Métodos modernos de Inteligencia Artificial

Grigori Sidorov (Ed.)

México D. F. 2011

Prefacio

Este libro contiene los trabajos de los especialistas prominentes en el área de la inteligencia artificial quienes en su mayoría trabajan en México.

La idea principal de este libro es dar a conocer a los lectores el estado actual de algunos de los métodos más utilizados de la inteligencia artificial moderna.

Los autores de este libro hicieron un gran esfuerzo para presentar a sus lectores un libro práctico y escrito de manera clara. Aquí se tratan los temas que es difícil encontrar en un solo libro especialmente redactado en el español. Además con diferencia a los libros clásicos de la inteligencia artificial aquí se presenta la información mucho más actualizada ya que es un libro mucho más reciente y presenta los últimos avances en el campo.

Por su diseño, es un libro de texto orientado a los estudiantes de todos los niveles —licenciatura, maestría y doctorado, y al público en general interesado en la inteligencia artificial. También el libro puede servir como un libro de consultas y ser útil para los especialistas de diferentes áreas —no solo en el área de la computación— quienes buscan la posibilidad de aplicar los métodos de la inteligencia artificial en sus áreas de conocimiento.

El libro tiene un enfoque práctico, es decir, está orientado a solución de los problemas prácticos que pueden presentarse tratando de aplicar algunos de los métodos de la inteligencia artificial. Cada capítulo tiene un número suficiente de los ejemplos prácticos de aplicaciones de los métodos y al final se proponen algunos ejercicios como un reto para los lectores a aplicar los conocimientos obtenidos durante su lectura.

Este libro no requiere conocimiento profundo previo ni de las matemáticas, ni de la computación, sin embargo se espera que el lector esté familiarizado con las ideas básicas de que es un algoritmo, cómo se escribe un programa, y sepa los conceptos fundamentales de las matemáticas.

Los temas principales abordados en el libro son los métodos de aprendizaje automático, optimización, razonamiento, y sus aplicaciones. Más específicamente se describen los métodos de búsqueda heurística, programación lógica, algoritmos genéticos, los paradigmas de optimización emergentes (evolución diferencial, optimización mediante cúmulo de partículas) las redes neuronales, aprendizaje por refuerzo, las ideas básicas de razonamiento temporal, solución de problemas usando agentes inteligentes y utilización de esos métodos en la robótica móvil. También se abarcan las ideas básicas relacionadas con el diseño de las ontologías, representación de conocimiento y su uso en práctica (sistemas expertos).

Después de leer este libro el lector sabrá cómo aplicar varios métodos de la inteligencia artificial a una amplia gama de problemas.

Grigori Sidorov

México D. F. a 10 de enero del 2011

Índice de alto nivel

Índice general

Introducción

Este libro describe los métodos que se encuentran en el foco de atención de la inteligencia artificial moderna. Por esta razón, nos centramos en la parte dinámica de la IA, la parte que se puede considerarse como estática, relacionada con representación de conocimiento, queda no cubierta en nuestro libro; sugerimos al lector consultar otros libros. También nótese, que este libro presenta el estado de arte de los métodos de la IA haciendo el hincapié en su estado moderno, sin embargo, por supuesto, existen las consideraciones clásicas que no pudimos omitir para tener un panorama completo.

Todos los capítulos contienen ejemplos detallados de funcionamiento de los algoritmos y métodos correspondientes.

El capítulo 1 presenta el análisis de los conceptos básicos de inteligencia artificial, tales como la propia inteligencia y la inteligencia artificial, implementando un análisis lingüístico basado en las definiciones que se encuentran en diccionarios. También se analiza la estructura del área de IA tomando en cuenta el flujo de información en nuestra interacción con el mundo externo.

En el Capítulo 2 se presentan las búsquedas heurísticas. Se discuten la búsqueda en el espacio de estados y la búsqueda basada en la reducción del problema. También se presentan el procedimiento minimax y el procedimiento alfa-beta.

El Capítulo 3 se dedica a los problemas relacionados con la representación del conocimiento. Se presentan sistemas expertos, manejo de incertidumbre, y se discuten los problemas relacionados con la creación y utilización de las ontologías.

Los temas relacionados con la programación lógica se presentan en el Capítulo 4. Se describe como desarrollar un programa un Prolog. Se introduce el concepto del árbol de resolución. Se discute semántica y control de los programas correspondientes.

En el capítulo 5 se da una introducción a las redes neuronales artificiales. Se presentan varias consideraciones prácticas de su uso. Se ven los casos de aplicaciones tales como reducción de ruido (redes auto-asociativas) y clasificación (redes usadas como clasificadores).

El capítulo 6 es una introducción a la computación evolutiva. El capítulo cubre tanto las bases teóricas que la sustentan como una interpretación intuitiva de los conceptos importantes del área.

En el capítulo 7 se presenta una introducción a los algoritmos bio-inspirados. Son un concepto de reciente creación. En primer término se presenta un panorama de las heurísticas para resolver problemas de búsqueda, se propone una clasificación basada en la motivación de cada una de ellas y se introducen los elementos que todo algoritmo bio-inspirado tiene. Posteriormente, los dos algoritmos bio-inspirados más recientemente propuestos en la literatura especializada se describen a nivel más detallado: La Evolución Diferencial (ED) y la Optimización Mediante Cúmulos de Partículas (PSO). Para cada uno de ellos se explica la forma de representar soluciones a un problema, la manera de escoger aquellas soluciones —llamadas individuos en ED y partículas en PSO— que guiarán a las demás soluciones en el proceso de búsqueda, los operadores que permiten generar nuevas soluciones a partir de las ya existentes y las técnicas para mantener aquellas mejores soluciones en la población (en el caso de la ED) y en el cúmulo (en el caso de PSO).

El capítulo 8 da una panorámica de los agentes inteligentes. Primero los agentes se presentan desde una perspectiva individual. Es decir, primero se caracterizan este tipo de agentes, se exponen algunos de los mecanismos más comunes de razonamiento práctico. Después, se presenta la descripción de los agentes inteligentes desde una perspectiva colectiva: los sistemas multiagentes. Se exponen los mecanismos de interacción y negociación más comunes; así como, los métodos de comunicación y coordinación en este tipo de sistemas.

El capítulo 9 presenta una introducción a los procesos de decisión de Markov (*Markov Decision Processes, MDP*, en inglés) y el aprendizaje por refuerzo. Se muestra como el problema de planeación puede ser representada como *MDP* y se describen los algoritmos básicos de su solución. También se presentan métodos de solución para problemas más complejos, mediante modelos factorizados y abstractos. Se ilustran estos métodos en tres aplicaciones: aprendiendo a volar un avión, controlando una planta eléctrica y coordinando a un robot mensajero.

El capítulo 10 discute la representación y razonamiento temporal. Muchos sistemas de la inteligencia artificial necesitan manejar la dimensión temporal de la información, tomar en cuenta los cambios en la información y poseer el conocimiento de cómo se cambia. Se presenta una descripción de algunos problemas fundamentales de las teorías y modelos temporales usando las consideraciones ontológicas del tiempo y de las relaciones de la orden temporal. Se da a conocer los tres enfoques principales a la representación de la información temporal en la IA y se proporciona una clasificación más fina de las proposiciones temporales.

En el capítulo 11 se describe el campo de la robótica basada en comportamiento. Para sentar las bases de la robótica basada en el comportamiento se presenta un enfoque reactivo donde la emergencia de las conductas hace posible resolver problemas relacionados con la selección de acción. Este es un problema recurrente donde varios subsistemas tratan de

acceder al mismo tiempo un recurso compartido. Se presentan también los elementos de robótica evolutiva, es decir, basada en el comportamiento animal. Existe una amplia gama de aplicaciones industriales de robots autónomos, por ejemplo: robots limpiadores de pisos en fábricas, sistemas de vigilancia, transporte de partes en fábricas, recolección y cosechas de frutos entre otros. A pesar de que el estado del arte actual todavía, aún no es posible desarrollar robots complejos para que realicen estas tareas con una entera autonomía. Sin embargo sí se puede diseñar y probar prototipos y algoritmos en plataformas comerciales de amplio uso en el área científica. Como un ejemplo, se describe el robot miniatura Khepera que es una de estas plataformas con la cual se pueden diseñar y llevar a cabo una serie de experimentos relacionados con la robótica reactiva.

Capítulo 1.

Conceptos básicos de inteligencia artificial y sus relaciones

1. Inteligencia Artificial

"*Pienso, luego, existo*" son las famosas palabras del filósofo francés René Descartes[1] las cuales se convirtieron en el elemento fundamental de la filosofía occidental. La interpretación más simple de esta frase es que si alguien empieza a pensar si él existe o no, este acto de pensamiento ya es una demostración que él sí existe. Sin embargo, lo que nos interesa en esta afirmación no son sus consecuencias filosóficas, sino la idea de que nuestra existencia como los seres humanos está directamente relacionada con nuestra capacidad de pensamiento.

Para proceder con la discusión siguiente sobre el tema de qué es la inteligencia artificial vamos a necesitar introducir un concepto muy importante para la descripción de cualquier ciencia —el concepto del "modelo científico".

1.1 Modelos científicos

Un modelo científico es una construcción mental que representa algunas características o propiedades del objeto correspondiente. Un modelo perfecto tendría todas las características y propiedades del objeto; y en este sentido sería completamente equivalente a éste. Sin embargo, prácticamente siempre los modelos omiten o ignoran algunas características de sus objetos de

[1] *René Descartes* (1596-1650) es el filósofo, matemático, científico e escritor francés (se pronuncia [*decárt*] según las reglas del francés). También tenía el seudónimo en latín *Renatus Cartesius*, por lo tanto el adjetivo *cartesiano* se refiere a las teorías o ideas relacionadas con él, por ejemplo, el sistema cartesiano de coordenadas.

modelado. Es completamente normal dado que los objetos son muy complejos, entonces, para poder utilizar algún modelo razonable estamos obligados a simplificar la representación del objeto hasta un cierto nivel permisible. Incluso, eso proporciona ciertos beneficios, ya que podemos ignorar las propiedades que no nos importan por el momento, y de tal manera nuestro objeto se convierte en algo más simple.

A menudo diferentes modelos representan diferentes características del mismo objeto y en este sentido no se puede decir que algún modelo es mejor que el otro. Esto también se conoce como diferentes *puntos de vista*, es decir, si se tienen dos puntos de vista diferentes y ambos representan algunas características del objeto, de antemano no se puede afirmar que un punto de vista es correcto y otro no lo es; quizá, simplemente las características que se representan en cada punto de vista son distintas. Claro, si son exactamente las mismas características del objeto, y las conclusiones son diferentes, se puede empezar a discutir de cuál punto de vista es mejor. Por ejemplo, el tiempo se puede describir con puntos en el tiempo o con los intervalos del tiempo (véase los capítulos siguientes). Son distintos puntos de vista, ambos modelos son aplicables, y no se puede decir que uno es "incorrecto". Sin embargo, se puede discutir sobre las ventajas de cada modelo, e incluso tratar de construir un modelo que va a combinar las ventajas de ambos.

Cualquier ciencia construye modelos de su objeto de estudio, visto de manera muy general; por ejemplo, la física construye modelos de la naturaleza, la biología construye modelos de la vida (de los seres vivos), la química trata las sustancias, etc. Más adelante vamos a definir que tipo de modelos pertenecen al dominio de la inteligencia artificial.

Del otro lado, nosotros como seres humanos también tenemos en nuestra cabeza un modelo específico —el modelo del mundo exterior. Éste ya no es un modelo científico, más bien es un modelo de sentido común; por ejemplo, sabemos que los objetos caen hacia abajo y no hacia arriba, pero no necesariamente conocemos que este fenómeno se explica con el concepto de la gravedad, etc. Es una representación del mundo exterior que cada uno de nosotros desarrolla usando sus habilidades cognitivas e interactuando con el mundo —incluyendo la interacción con otras personas.

Es decir, al percibir algo, estamos suponiendo qué cosa eso podría ser, y después actuamos en consecuencia. Si nos equivocamos, modificamos nuestra suposición y volvemos a intentar[2]. Así todos nosotros tenemos construido un modelo bastante completo del mundo exterior desde que éramos niños, que utilizamos en nuestra vida cotidiana.

[2] Esta manera de aprender se llama *aprendizaje por refuerzo*, véase más adelante.

1.2 Búsqueda de definiciones: inteligencia

El pensar es una característica muy importante de los seres humanos la que nos distingue de otros seres vivos, los que pueden reaccionar a los estímulos externos pero no pueden formular el problema y después resolverlo de manera racional, la actividad para cual usamos nuestra inteligencia natural. Entonces, nuestra inteligencia está relacionada con nuestro pensamiento. Pero ¿qué es la inteligencia?

El diccionario de la Real Academia Española [3] proporciona los siguientes sentidos de la palabra "*inteligencia*" que nos puede interesar:

1. *Capacidad de entender o comprender.*
2. *Capacidad de resolver problemas.*
3. *Conocimiento, comprensión, acto de entender.*

Es obvio que el sentido *3* es el resultado de utilización de de la capacidad mencionada en el sentido *1*.

Analicemos el sentido *1*. ¿Qué significa "*entender*"? El mismo diccionario nos ofrece lo siguiente:

1. *Tener idea clara de las cosas.*
2. *Saber con perfección algo.*

Parece que "*idea clara*" y "*saber*" es lo mismo. Consultemos la palabra "*saber*", la cual se define a través de "*conocer*", y si verifiquemos la definición de "*conocer*", allá aparece

1. *Averiguar por el ejercicio de las facultades intelectuales la naturaleza, cualidades y relaciones de las cosas.*
2. *Entender, advertir, saber, echar de ver.*

En el sentido *2* vemos un ejemplo de lo que se llama el círculo vicioso en definiciones, es decir, *entender* $\rightarrow$ *saber*, *saber* $\rightarrow$ *conocer*, y *conocer* $\rightarrow$ *entender*. (Le flecha se interpreta como "se define a través de".) Los círculos viciosos son inevitables si el diccionario no usa un conjunto de las palabras sin definiciones (las palabras primitivas, o el vocabulario definidor), y es un fenómeno completamente normal para un diccionario escrito para los lectores humanos. Aunque en general es deseable que la longitud de los círculos sea más larga que en este caso (véase la discusión en [1], Capítulo 4).

Ahora bien analicemos el sentido *1* de la palabra "*conocer*" (éste, a propósito, tiene su propio círculo vicioso, *averiguar* $\rightarrow$ *inquirir* $\rightarrow$ *averiguar*, pero eso no importa para la discusión siguiente). Y cómo ya vimos analizando el sentido *2*, *conocer* se define a través de *entender*. Lo que se dice en definición *1*, es que para entender un concepto debemos establecer relaciones de éste con otros conceptos; en otras palabras, incluir este concepto en nuestro modelo del mundo, y, agreguemos nosotros, sin que aparezcan las contradicciones. De hecho, el resultado de lo que acabamos de decir, tiene para

su descripción otra palabra importante —"*conocimiento*", pues, lógico, que el resultado de *conocer* es el *conocimiento*.

Resumiendo, nuestra interpretación de los conceptos "*conocer*" y "*entender*" es la siguiente:

- Cada uno de nosotros tiene un modelo interno del mundo exterior construido con las experiencias previas (conocimiento).
- Al encontrarnos con una experiencia nueva tratamos de ubicarla en nuestro modelo del mundo evitando las contradicciones.

Y ¿qué pasa con la palabra "*pensar*"? El diccionario de la Real Academia Española nos da los tres sentidos siguientes:

1. *Imaginar, considerar o discurrir.*
2. *Reflexionar, examinar con cuidado algo para formar dictamen.*
3. *Intentar o formar ánimo de hacer algo.*

De los cuales nos interesa más el sentido *2*.

Nótese que "*formar dictamen*" aquí se interpreta como "*construir un modelo*".

Analizando las definiciones del diccionario podemos ver que *examinar* → *inquirir* → *averiguar*. Acabamos de encontrarnos con la palabra "*averiguar*" viendo el concepto de "*inteligencia*". Del otro lado, la palabra "*reflexionar*" no nos ayuda mucho en el análisis de "*pensar*" debido al círculo *reflexionar* → *considerar* → *pensar*.

Entonces, viendo las relaciones entre las palabras, podemos concluir que "*entender, conocer*" es el resultado de "*pensar*", mientras que "*pensar*" es el proceso que nos lleva a "*entender, conocer*", y el resultado de "*entender, conocer*" es el "*conocimiento*"; la "*inteligencia*" es la capacidad de "*pensar*" o "*entender*". El significado del concepto "*entender*" acabamos de ver un par de párrafos más arriba —inclusión en el modelo interno del mundo exterior sin contradicciones.

Existe, sin embargo, el sentido *2* de la palabra "*inteligencia*": "*capacidad de resolver problemas*". ¿Es algo distinto de lo que acabamos de discutir? Creemos que no, ya que es un caso particular de "*pensar*", puesto que para resolver problemas hay que pensar. Lo que se cambia en este sentido específico es que se agrega una meta u objeto de pensamiento impuesto de afuera —el problema, es decir, el problema se impone desde afuera para el acto de pensamiento.

Nótese que en el sentido *2* no se incluye la pregunta de cómo encontrar los problemas para resolver. Es una pregunta también muy importante, incluso para la inteligencia artificial, sin embargo, para contestarla, es decir, para decidir qué tipos de problemas son de interés, se utiliza en una gran medida la intuición. Es más, se sabe que en una investigación científica, es mucho más difícil plantear un problema de manera correcta, que después encontrar su solución. En otras palabras, es más difícil decidir, sobre que vale la pena

pensar, que el propio pensar. Sin embargo, en este libro vamos a hablar de los problemas que ya están presentes y formulados, y hay que buscar una solución para ellos.

1.3 Búsqueda de definiciones: inteligencia artificial

Seguimos en nuestra búsqueda de definiciones con el concepto de la "*inteligencia artificial*". En el mismo diccionario de la Real Academia Española [3] podemos encontrar la siguiente definición de la IA:

Desarrollo y utilización de computadoras[3] *con los que se intenta reproducir los procesos de la inteligencia humana.*

Nos parece muy acertada esta definición con las siguientes precisiones. La inteligencia artificial es una ciencia, y como cualquier ciencia ella tiene su objeto de estudio —la inteligencia humana; pero a diferencia con el punto de vista tradicional, los modelos que construye la inteligencia artificial de su objeto deben ser exactos en el sentido que fuesen "entendibles" para las computadoras. Es la diferencia entre la IA y la otra ciencia que ha estudiado la inteligencia humana durante siglos —la filosofía— pero sin la necesidad de tanta exactitud. Claro que la filosofía a parte de la inteligencia estudia muchas otras cosas. Del otro lado, la parte computacional en la inteligencia artificial es primordial —debido a la necesidad de verificación de las teorías—, por eso tradicionalmente la inteligencia artificial se considera como parte de las ciencias computacionales.

Entonces, podemos reformular la definición del objeto de la inteligencia artificial de la siguiente manera:

Construcción de modelos de la inteligencia humana, tales que se puede implementarlos en las computadoras.

Se puede hacer una analogía con la física —antes de que se empezara a aplicar el aparato matemático en la física, ésta última era una ciencia descriptiva; después de aplicar los conceptos matemáticos, la física se convirtió en una ciencia exacta. Lo mismo sucedió con la inteligencia artificial, antes de que aparecieran las computadoras, la inteligencia artificial era parte de la filosofía —aunque no se utilizaba este nombre. Con las computadoras, la inteligencia artificial se convirtió en una ciencia exacta, ya que apareció la posibilidad de verificar sus teorías. Además, como siempre sucede con las computadoras, se agregó la posibilidad de procesar unas cantidades enormes de datos, lo que da una nueva calidad a la ciencia de la inteligencia artificial. Nótese que la palabra *artificial* precisamente subraya la idea de construir los

[3] En la definición original, en lugar de "*computadoras*" se usa la palabra "*ordenadores*" (según el uso del español en España).

modelos que se pueda implementar en algún artefacto, normalmente, la computadora.

Vamos a presentar algunas otras posibles definiciones de la inteligencia artificial. Cabe mencionar que cada definición representa un modelo, es decir, enfatiza algunas características e ignora otras.

Una definición conocida es "*sistemas basados en conocimientos*". Como ya sabemos, el conocimiento es precisamente el modelo del mundo externo, sin embargo, esa definición se centra en el conocimiento, e ignora los algoritmos que también son una parte importante de la IA.

Otra posible definición es "*sistemas basados en heurísticas*". Las heurísticas son las reglas o procedimientos que se aplican cuando no existe un método exacto para resolver un problema dado. Esa definición enfatiza la relación entre los métodos donde existe una solución exacta y los métodos donde no se sabe de antemano el camino a la solución, y para encontrar tal camino se usa la inteligencia, basada en el conocimiento y el razonamiento.

Entre otras distinciones frecuentemente aceptadas, que se presentan, por ejemplo, en [2], las definiciones de la inteligencia artificial se agrupan en cuatro categorías, a saber, la IA se define como:

- *Sistemas que piensan como humanos,*
- *Sistemas que actúan como humanos,*
- *Sistemas que piensan racionalmente,*
- *Sistemas que actúan racionalmente.*

Analizando esas definiciones, nos damos cuenta que todos se refieren a los sistemas, es decir, a los modelos implementados en las computadoras.

Del otro lado, las diferencias entre las definiciones están relacionadas con el uso de las palabras "*actuar*" vs. "*pensar*" y "*como humanos*" vs. "*racionalmente*". Nos parece que las diferencias entre ambas pares de palabras se refieren en la realidad al mismo rasgo. De hecho, tanto *actuar* como *pensar* se refieren al modelo del mundo, donde *actuar* implica la realización de ciertas acciones, mientras que en pensar no se realizan las acciones, pero se decide que hay que realizarlas; notemos que en *actuar* tampoco se realizan las acciones al azar, sino dado algunas consideraciones previas, es decir, existe algún tipo de pensamiento —sea "*racional*" o "*como humanos*". Entonces, la única diferencia entre *actuar* y *pensar* en este caso es la implementación física de las acciones, lo que nos parece irrelevante en el contexto de análisis ya que eso no está realmente relacionado con la propia inteligencia.

Ahora bien, ¿y qué significa *racionalmente*? Según el mismo diccionario es "*relativo a la razón*", y la *razón* es la "*facultad de discurrir*"; *discurrir* en su turno significa "*inferir, conjeturar, reflexionar, pensar, aplicar la inteligencia*". Sin embargo, nos parece que la clave de la diferencia entre "*racionalmente*" y "*como humanos*" está en las palabras "*como humanos*", porque está claro que este enfoque también es racional, por lo menos todas las cosas irracionales,

como sentimientos, emociones, etc. que poseemos los seres humanos no son objetos de estudio de la inteligencia artificial tradicional.

Es decir, las definiciones reflejan dos posibles enfoques en construcción de los modelos de la inteligencia. El enfoque que proclama "*como humanos*", quiere decir que debemos estudiar los procesos cognitivos e intelectuales en los humanos y después modelar estos procesos en las computadoras, lo que significa que las computadoras deben funcionar como humanos. El otro enfoque es más pragmático, y proclama que no importa cómo es el proceso de pensamiento de los seres humanos si el resultado va a ser el mismo.

El primer enfoque se llama cognitivo, mientras que el segundo tiene el nombre del enfoque de ingeniería. Un ejemplo: los aviones vuelan utilizando principios muy diferentes que las aves, —es el enfoque de ingeniería, cuando no importa como se hace en la naturaleza, pero que sí el resultado (funcionamiento). Si vamos a estudiar primero como vuelan las aves y después construir una máquina que lo haga igual, eso sería un enfoque similar al enfoque cognitivo.

Existe una famosa prueba llamada "*prueba de Turing*[4]", que consiste en que un ser humano puede interrogar a un interlocutor sin verlo, digamos, chateando, y la tarea es adivinar si es una computadora u otro ser humano. Las respuestas van a demostrar la inteligencia del interlocutor y analizándola se debe decidir si el interlocutor es una computadora o un ser humano. Obvio, para esta prueba no importa que enfoque se implemente en la computadora, lo que cuenta es solamente el resultado final; pero el desarrollador debe tomar alguna decisión referente al enfoque. Se puede criticar esta prueba con los argumentos que no se demuestra una inteligencia verdadera, sino una simulación, que no tiene nada que ver con la inteligencia de verdad. Otra parte débil de esta prueba es su antropocentrismo, es decir, no se verifica la inteligencia como tal, sino algunos rasgos específicos de los seres humanos que posiblemente no estén relacionados con la inteligencia. Sin embargo, si las preguntas se limitan a las áreas relacionadas con la inteligencia, esta prueba funcional permite entender mejor hasta dónde deben llegar los modelos computacionales para poder competir con los seres humanos.

[4] *Alan Turing* (1912-1954) era un matemático, lógico y criptógrafo inglés que se considera el fundador de la ciencia de la computación moderna. Durante la Segunda Guerra Mundial (1939-1945) empezó a utilizar una maquina electromecánica para descifrar los códigos de comunicación naval de los alemanes nazi. Después trabajó sobre el desarrollo de una de las primeras computadoras en el mundo (*Manchester Mark I*). Propuso una formalización muy importante del concepto de algoritmo —la *máquina de Turing* (no es una computadora, es un modelo matemático formal); y además la *prueba de Turing* mencionada.

2. Estructura del área de la inteligencia artificial

La inteligencia artificial está compuesta por varias áreas bastante grandes. Las diferencias entre las áreas se basan en qué parte de la inteligencia se modela. Para imaginar todo el proceso debemos evaluar como funcionamos nosotros mismos, lo que se hace con el método de la introspección —es decir, tratando de observar nuestros propios acciones, pensamientos, etc.

Nuestro funcionamiento e interacción con el mundo externo consiste en:

- Percepción basada en la información obtenida de nuestros sentidos — son cinco sentidos: la visión, la audición, el olfato, el gusto, el tacto; pero el sentido más importante es la visión, según algunas evaluaciones más del 90% de la información recibida es visual.
- Pensamiento basado en el modelo actual del mundo, que consiste en la ubicación de las percepciones en el modelo del mundo y del razonamiento sobre este modelo para resolver problemas.
- Acciones que realizamos a través de nuestro cuerpo y lenguaje.
- Uso de lenguaje natural para el pensamiento y para la transmisión/recepción de información en la sociedad. Lenguaje natural es una capacidad única de los seres humanos. El lenguaje nos hace lo que somos. Ningún animal tiene un lenguaje; claro, los animales tienen sus sistemas de comunicación, pero nunca llegan a nivel de abstracción que desarrollaron los humanos. Entonces, de un lado, los humanos reciben la información a través del lenguaje y absorben toda la experiencia de la sociedad grabada en los textos y en palabras de los otros individuos. Del otro lado, el lenguaje natural es nuestra principal manera de representar el conocimiento en nuestro modelo del mundo. También podemos reaccionar no solo con las acciones físicas, sino en forma verbal entrando en un diálogo.

Esta interacción determina la estructura de la inteligencia artificial como ciencia que modela la inteligencia humana, véase Ilustración 1. Esta ilustración representa el flujo de la información, con su entrada, procesamiento y salida en forma de las reacciones posibles.

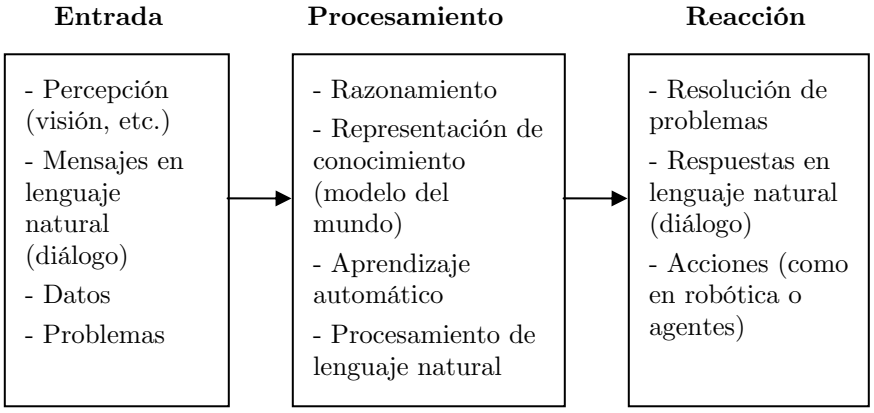


Ilustración 1. Flujo de información en la IA.

Algunos conceptos presentados en Ilustración 1 son pura información (como datos, mensajes o problemas), otros son reacciones (respuestas, acciones, resolución de problemas), el resto de los conceptos representan el núcleo de la IA como ciencia.

A la percepción —que es en su gran parte la visión, los otros sentidos casi no se utilizan en la IA en su estado actual— le corresponde el área de procesamiento de imágenes y reconocimiento de patrones en imágenes. Como está claro de su nombre, dentro de esta área se modela la percepción visual y se trata de detectar e identificar los objetos en imágenes.

El área de procesamiento de lenguaje natural (PLN) es una parte grande de la IA moderna. En el PLN se analizan la estructura de lenguaje humano y sus manifestaciones en los lenguajes específicos como el español, el inglés, el ruso, etc. para fines de la construcción de los modelos computacionales. También se estudian los métodos de cómo hacer que las computadoras entienden los textos, implementando análisis morfológico, sintáctico y semántico. El PLN es tan grande que a menudo se considera como una ciencia separada del área de la IA. Eso se fomenta también por el hecho que su objeto de estudio —el lenguaje natural— es fácilmente separable de los objetos de estudio de las otras áreas de la IA, como razonamiento, o representación del conocimiento, etc. Sin embargo, es necesario siempre tener en mente que el lenguaje es una parte esencial de nuestra inteligencia, aunque su procesamiento y los métodos que se usan pueden ser distintos de los métodos de las otras áreas de la IA.

Las tres otras áreas —razonamiento, representación de conocimiento, aprendizaje automático— son las áreas principales de la IA tradicional.

Mientras nos gustaría hacer notar la relación entre razonamiento y representación de conocimiento. El razonamiento siempre es una acción (o un procedimiento) que se realiza según ciertos algoritmos y debe realizarse sobre algo. Este algo es precisamente el conocimiento. Es decir, el conocimiento es la

base de razonamiento, y razonamiento son acciones sobre el conocimiento. De tal manera uno no puede existir sin el otro. Utilizando otra metáfora, se puede decir que el conocimiento es algo estático, mientras que el razonamiento es algo dinámico.

El aprendizaje automático es otro tipo especial de procedimientos (algoritmos) cuando a partir de los datos puros —es decir, de algo que no está explícitamente estructurado o está estructurado solamente hasta cierto grado— se genera el conocimiento, lo que ya tiene una estructura explícita y/o más completa.

Las demás áreas de la IA se pueden ser vistas o bien como una combinación de las áreas mencionadas, o como una aplicación específica en algún campo, o como la adición de algunas características específicas a las áreas mencionadas.

Por ejemplo, *planeación* es el razonamiento en el tiempo y/o espacio, es decir, se agregan las características relacionadas con el tiempo y/o espacio.

Otro ejemplo, las áreas de *robótica* y *agentes* modelan todo el proceso *entrada→procesamiento→reacción*. Entonces, esas áreas se pueden considerarse como una combinación de las áreas mencionadas. La diferencia principal entre esas dos áreas es la siguiente: en la robótica se supone la existencia de una máquina física (un robot) y los agentes existen como un modelo computacional (son programas).

Un ejemplo de una aplicación específica es, digamos, *procesamiento de la información geoespacial*, donde se usan las técnicas de procesamiento de imágenes combinadas con una representación del conocimiento específico aplicado a las imágenes de la superficie terrestre o los mapas.

3. Métodos de inteligencia artificial

En esta sección presentamos de manera muy general algunos métodos de la inteligencia artificial relacionados con el razonamiento y aprendizaje automático descritos a detalle en otros capítulos de este libro. Ya que el libro se centra en los métodos de la IA, es decir, en la parte de algoritmos, o en la parte dinámica, no discutiremos los problemas relacionados con la representación de conocimiento —la parte estática, estas cuestiones están descritas en otros libros, por ejemplo, en [2].

Antes que nada hay que examinar que conceptos son de mayor importancia en esta área y ver lar relaciones entre ellos.

Los conceptos son:

- información (datos),
- aprendizaje,
- clasificación,
- problema,
- razonamiento (enfocado en solución de problemas),

- optimización.

Igual que en las secciones anteriores vamos a basarnos en el concepto del modelo del mundo. Además, agreguemos los conceptos intuitivamente claros: estado actual del mundo y su estado deseado. Es importante agregar que no nos ocupamos en este libro de cómo determinar qué estado es el deseado, lo consideramos como algo externo.

Información son los datos que existen en el mundo. Nótese la diferencia de la información y el conocimiento: El *conocimiento* es la información que está incorporada en nuestro modelo del mundo.

Aprendizaje es un método de cómo convertir la información en el conocimiento, es decir, es una manera de incluirla en el modelo del mundo de manera no contradictoria.

Clasificación es un método de cómo llegar al saber el estado actual del modelo del mundo a partir de la información, es decir, entender, convertir la información que tenemos en el conocimiento dónde estamos en nuestro modelo. En otras palabras, basándose en la información disponible distinguir entre los posibles estados, es decir, clasificar.

Un *problema* es un estado deseado del modelo del mundo.

En este sentido *razonamiento* (enfocado en solución de problemas) son las acciones que hay que tomar y los cambios en el modelo del mundo que hay que realizar para alcanzar este estado deseado. Nótese que también se supone que se sabe el estado actual del mundo que se va a cambiar. Entonces, razonamiento son las acciones que nos llevan al cambio del estado actual al estado deseado.

Optimización es un método que nos lleva a base de lo que se sabe del estado actual al saber el estado deseado del mundo. Es decir, mi estado actual puede ser no es el mejor, pero a partir de ese estado, puedo alcanzar un estado mejor (optimizado). A diferencia de solución de problemas, no se dicen las acciones que hay que tomar. Sin embargo, está claro que ése puede ser un paso trivial cuando la acción es "cambiarse al estado óptimo", —en este caso la optimización va a ser equivalente a la solución de problemas.

Entre los conceptos mencionados —aprendizaje, clasificación, razonamiento, solución de problemas, optimización— son los conceptos que corresponden a los métodos de la inteligencia artificial. Todos esos métodos tienen su representación en este libro.

Referencias

[1] Alexander Gelbukh, Grigori Sidorov. Procesamiento automático del español con enfoque en recursos léxicos grandes. IPN, 2006, 240 p. http://www.gelbukh.com/libro-procesamiento/

[2] Stuart Russel, Peter Norwig. Inteligencia Artificial. Enfoque moderno. Prentice Hall, 1996, 979 p.

[3] Diccionario de la Real Academia Española. www.rae.es. Consultado 30.04.2008.
[4] Nils Nilsson. Problem-solving methods in artificial intelligence. NY, McGrow-Hill, 1971, 255 p.
[5] Elaine Rich, Kevin Knight. Artificial Intelligence. NY, McGrow-Hill, 1990, 621 p.

Capítulo 2.

Búsqueda heurística

1. Introducción

La teoría de búsqueda heurística es un área clásica y bien desarrollada de la Inteligencia Artificial, que sigue manteniendo su actualidad dado su aplicación en los problemas prácticos. Dos elementos fundamentales de resolución de problemas dentro de esta teoría son: *representación* del problema (formalización) y la propia resolución —*búsqueda*.

En este capítulo, se presentan dos enfoques para la resolución de problemas, y de modo correspondiente, dos maneras de su representación:

- Enfoque basado en el uso del espacio de estados, y
- Enfoque basado en la reducción de problemas.

Para ambos enfoques se presentan los algoritmos para la búsqueda de soluciones. Una propiedad importante de la gran mayoría de esos algoritmos es el uso de la información *heurística*.

Heurística es cualquier regla, estrategia o procedimiento que ayuda sustancialmente en la resolución de algún problema. Dentro del área de la Inteligencia Artificial y de la teoría de búsqueda, la información heurística es todo relacionado con el problema específico en cuestión y sirve para su solución más eficaz.

En este capítulo también presentamos los procedimientos básicos de búsqueda en los árboles de juegos, que se usan aplicando a los juegos el enfoque basado en reducción de los problemas. Ésos procedimientos actualmente se utilizan en programación de un espectro bastante amplio de juegos —juegos de dos participantes con la información completa.

Igual que en muchos otros libros dedicados a la Inteligencia Artificial, los conceptos principales y los métodos de la búsqueda heurística se explican usando como ejemplos los juegos, rompecabezas, y los problemas matemáticos famosos. Esa tradición que se sostiene por más de 30 años de existencia de la

teoría de búsqueda heurística tiene una razón importante. Como dice uno de los creadores de esta teoría N. Nilson (1971): "los juegos y problemas matemáticos no se usan porque son claros y simples, sino porque representan mayor complejidad con una estructura mínima... en los problemas relacionados con juegos y rompecabezas aparecieron y se pulieron muchas ideas que resultaron ser realmente útiles para los problemas realmente serios".

2. Búsqueda en el espacio de estados

2.1 Representación de los problemas en un espacio de estados

Un típico representante de una clase de problemas para los cuales es aplicable la representación con espacios de estados es un rompecabezas conocido como el juego de "quince", véase Ilustración 1a. En este rompecabezas se usan quince fichas numeradas de 1 a 15. Las fichas se encuentran dentro de un cuadrado 4x4. Una celda de este cuadrado siempre se queda vacía, de manera que una de las fichas vecinas puede moverse —vertical o horizontalmente— para ocupar este espacio vacío y dejando como vacío su posición anterior. Una variante más simple de este juego sería el juego de "ocho" —un cuadrado 3x3 y ocho fichas— véase el ejemplo en la Ilustración 1b.

En la Ilustración 1a se presentan dos configuraciones de las fichas. En este rompecabezas se requiere transformar la primera configuración —inicial— en la segunda configuración —configuración meta. Nótese que la configuración inicial se genera de manera aleatoria. La solución de este problema es una secuencia de jugadas de las fichas, por ejemplo, mover la ficha 8 arriba, mover la ficha 6 a la izquierda, etc.

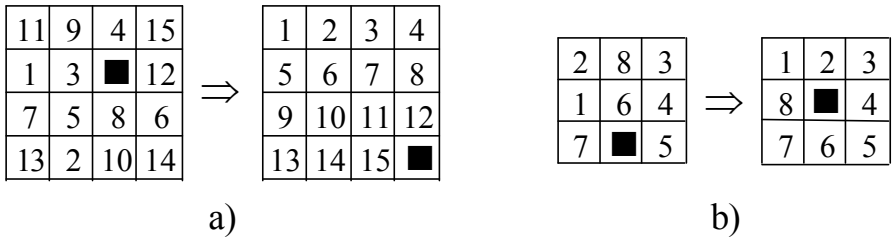


Ilustración 1. Juego de "quince" y juego de "ocho".

Una característica importante de este tipo de problemas (como juego de "quince" y juego de "ocho") es el hecho que la situación inicial y la situación meta están claramente definidas. También está presente un conjunto de

operaciones, o jugadas, que transforman una situación a otra. Precisamente en estas jugadas consiste la solución del problema, que en teoría, se puede obtener utilizando la estrategia de prueba y error. Realmente, empezando de una situación inicial se puede construir las configuraciones correspondientes a todas las posibles jugadas, después para cada configuración obtenida construir las configuraciones que corresponden a la siguiente jugada en esta configuración, y así sucesivamente hasta obtener la configuración meta.

Ahora vamos a introducir los conceptos que se usan para formalizar un problema en un espacio de estados. El concepto central es el concepto de **estado**. Por ejemplo, para el juego de "quince", el estado es una configuración específica de las fichas.

Entre todos los estados se distinguen un **estado inicial** y un **estado meta**. Esos dos estados juntos definen el problema que hay que resolver, véase los ejemplos en la Ilustración 1.

Otro concepto importante es el concepto de **operador**, es decir un movimiento permitido en un estado dado. El operador transforma un estado a otro, siendo en realidad una función definida sobre el conjunto de estados y obteniendo los valores del mismo conjunto. Para los juegos de "quince" u "ocho" es conveniente definir cuatro operadores que corresponden a las jugadas de la celda vacía (fichas sin número, ficha vacía) arriba, abajo, a la izquierda y a la derecha. En algunas situaciones el operador puede ser no aplicable a algún estado: por ejemplo los operadores de jugada a la derecha y abajo no son aplicables si la celda vacía se encuentra en la esquina derecha abajo. Entonces de manera generalizada un operador es una función parcialmente definida entre los estados.

En los términos de estados y operadores una **solución del problema** es una secuencia de operadores que transforma un estado inicial a un estado meta. La solución del problema se busca en un **espacio de estados**, es decir, en un conjunto de todos los estados que se puede alcanzar de un estado inicial dado utilizando operadores determinados. Por ejemplo en los juegos de "quince" u "ocho" el espacio de estados consiste en todas las configuraciones posibles de las fichas que pueden aparecer como resultado de las jugadas posibles de las fichas.

Un espacio de estados se puede representar como un grafo dirigido, donde los nodos corresponden a los estados y las aristas corresponden a los operadores aplicables. Las direcciones representadas con las flechas corresponden a las jugadas posibles empezando desde el nodo al cual se aplica el operador hasta el nodo que corresponde al resultado de la jugada. Entonces, la solución del problema es un camino en este grafo que lleva desde un estado inicial a un estado meta. En la Ilustración 2 se muestra una parte de espacio de estados para el juego de "quince". Cada nodo corresponde a una configuración de las fichas que este nodo representa. Las flechas son bidireccionales dado que en este rompecabezas cada operador tiene un

operador inverso, o más bien, el conjunto de los operadores consiste de dos pares de los operadores mutuamente inversos: arriba-abajo, izquierda-derecha.

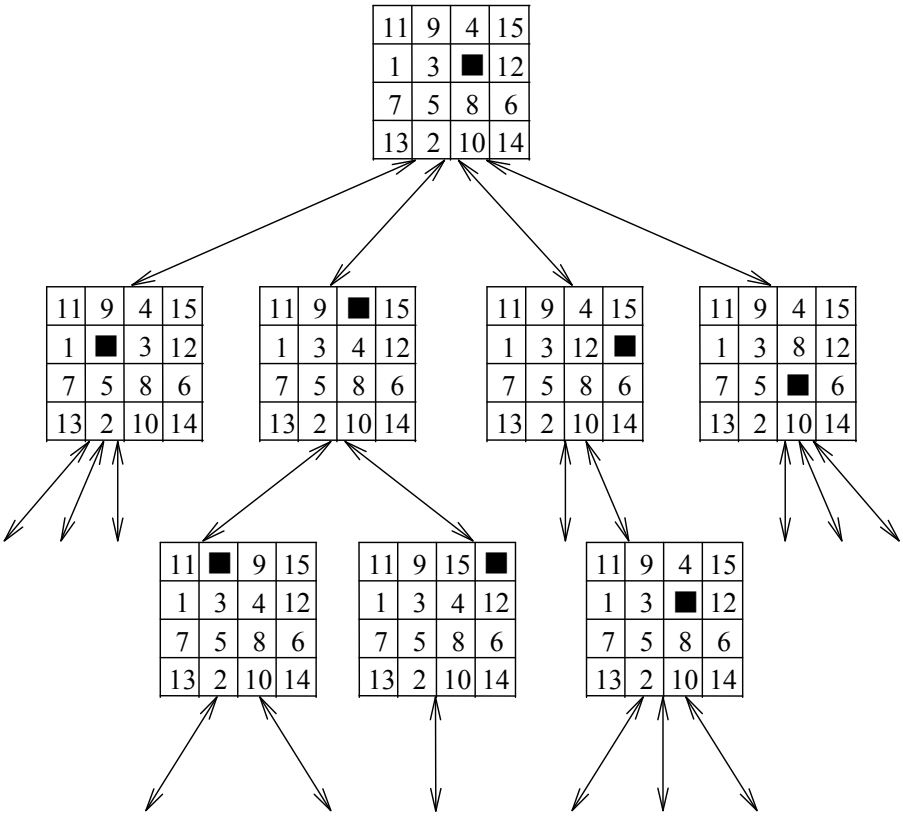


Ilustración 2. Una parte de espacio de estados para el juego de "quince".

Los espacios de estados pueden ser muy grandes y hasta infinitos, pero en todos los casos se supone que el conjunto de los espacios de estados es enumerable.

Entonces, en el enfoque de solución de los problemas utilizando un espacio de estados el problema se le presenta como una tupla (E_i, O, E_m), donde:

E_i es un estado inicial,

O es un conjunto finito de los operadores que son aplicables a no más que enumerable conjunto de estados

E_m es un estado meta.

El siguiente paso en la formalización de un problema supone la selección de algún ***modo de descripción*** de los estados del problema. Para eso se puede utilizar cualquier estructura aplicable de los lenguajes de programación

estándar: cadenas, listas, arreglos, árboles, etc. Por ejemplo, para los juegos de "quince" u "ocho" la estructura más natural para la representación de un estado es un arreglo bidimensional. Es importante mencionar, que de la selección de modo de representación de un estado depende la manera de la descripción de los operadores del problema, los que también deben ser definidos durante la formalización del problema en un espacio de estados.

Si vamos a utilizar algún lenguaje de programación estándar para la formalización del juego de "quince", entonces los operadores del problema pueden ser representados como cuatro funciones de este lenguaje. Si vamos a utilizar algún lenguaje de programación basado en reglas, entonces los operadores se representan como las reglas, por ejemplo, "estado inicial → estado resultante" (Nilsson, 1980).

En los ejemplos que hemos visto de los juegos de "quince" u "ocho" (Ilustración 1) cada estado meta se representa de manera directa, es decir, se conoce la posición de cada ficha en el estado meta. En los casos más complejos puede haber más de un estado meta, o bien, estado meta puede estar definido de manera indirecta, es decir, tener cierta propiedad, por ejemplo, un estado donde la sumatoria de los números de las fichas en la primera fila es menor de 10. En estos casos, la propiedad que tiene el estado meta debe ser definida de manera exacta, digamos, definiendo una función booleana que implemente la verificación de esta propiedad de un estado.

Resumiendo, para la representación de un problema en un espacio de estados es necesario definir lo siguiente:

- Modo de descripción de los estados del problema, incluyendo un estado inicial.
- Conjunto de operadores y resultados de su aplicación a los estados.
- Conjunto de estados meta o bien descripción de sus propiedades.

Estas características definen de manera implícita un grafo (espacio de estados), en el cual es necesario encontrar una solución del problema. La solución del problema en un espacio de estados se realiza como una ***búsqueda*** secuencial en el ***espacio de estados.*** En el punto inicial, al estado inicial se le aplica algún operador y se construye un nuevo nodo (correspondiente a algún estado), y las aristas que lo conectan con el nodo raíz (padre). En cada paso posterior de la búsqueda, a uno de los nodos (correspondientes a estados) se les aplica un operador permitido y se construye otro nodo del grafo y las aristas correspondientes. Si el nodo construido corresponde al estado meta, el proceso se termina.

2.2 Ejemplos de los espacios de estados

Vamos a ver dos ejemplos típicos de representación de problemas en un espacio de estados, que muestran que esta representación se aplica para

diferentes tipos de problemas. Es necesario recalcar que aunque la representación propuesta es bastante natural, no es la representación única posible, existen otras maneras de representar el problema.

De manera general, la selección de representación es muy importante, ya que de eso depende la dimensión de espacio de estados y por lo tanto, la efectividad de búsqueda en él. Obviamente, la representación con espacio de estados pequeño es muy deseable, sin embargo, selección de las representaciones que disminuyen el espacio de la búsqueda normalmente requiere un análisis adicional del problema en cuestión.

Consideremos la formalización en un espacio de estados del problema conocido el viajero. El viajero tiene un mapa de los caminos que interconectan a siete ciudades, véase la Ilustración 3. Él debe planear su viaje de tal manera que partiendo de la ciudad A, visitara cada uno de las otras seis ciudades B, C, D, E, F, G exactamente una vez y después se regresara a su ciudad de partida inicial. Existe otra variante más complejo del mismo problema, donde se requiere que la ruta tuviera una longitud mínima.

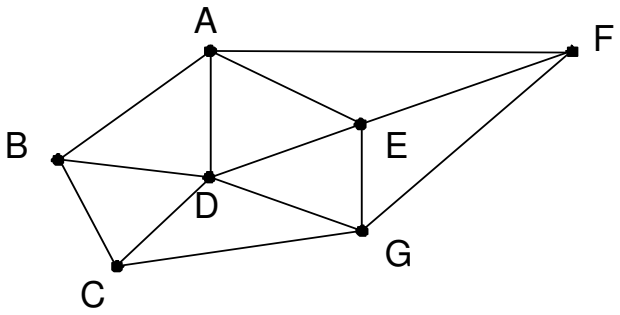


Ilustración 3. Problema del viajero: mapa de caminos entre las ciudades.

El estado del problema en cuestión se puede representar como una lista de las ciudades ya visitadas por el viajero. Entonces, a los posibles estados corresponden las listas de los elementos A, B, C, D, E, F, G sin repeticiones (con la excepción de la ciudad A, que puede aparecer en la lista dos veces, en su inicio y en su final). Ejemplo de la lista que corresponde a algún estado: (A B C F). El estado inicial se define como la lista que contiene solamente el elemento A: (A). El estado meta se define como cualquier lista permitida que contenga cada uno de los elementos por lo menos una vez y que inicia y termina con el elemento A.

Para los estados definidos de esta manera, los operadores del problema corresponden a las jugadas entre las ciudades, es decir, a las aristas del grafo representado en la Ilustración 3. De esta manera tenemos 13 operadores. En la Ilustración 4 se muestra un fragmento del espacio de estados resultante representado como un grafo, que también incluye la ruta que representa la

solución del problema. Este grafo es un árbol, es decir, al aplicar cualquier operador la lista de las ciudades ya visitadas solamente puede aumentarse, por lo tanto, al construir un nuevo nodo no podemos encontrarnos con un nodo ya procesado anteriormente.

Los operadores en este problema pueden ser definidos de otra manera. Por ejemplo, para cada de las siete ciudades vamos a introducir un operador que corresponde a la jugada a esta ciudad (de cualquier otra ciudad). Este operador es aplicable a algún estado si él corresponde al mapa de los caminos y la lista que corresponde al estado actual no contenga la ciudad al cual hay que moverse. Por ejemplo, el operador de moverse a la ciudad B no es aplicable al estado (A B C D), pero sí es aplicable al estado (A D).

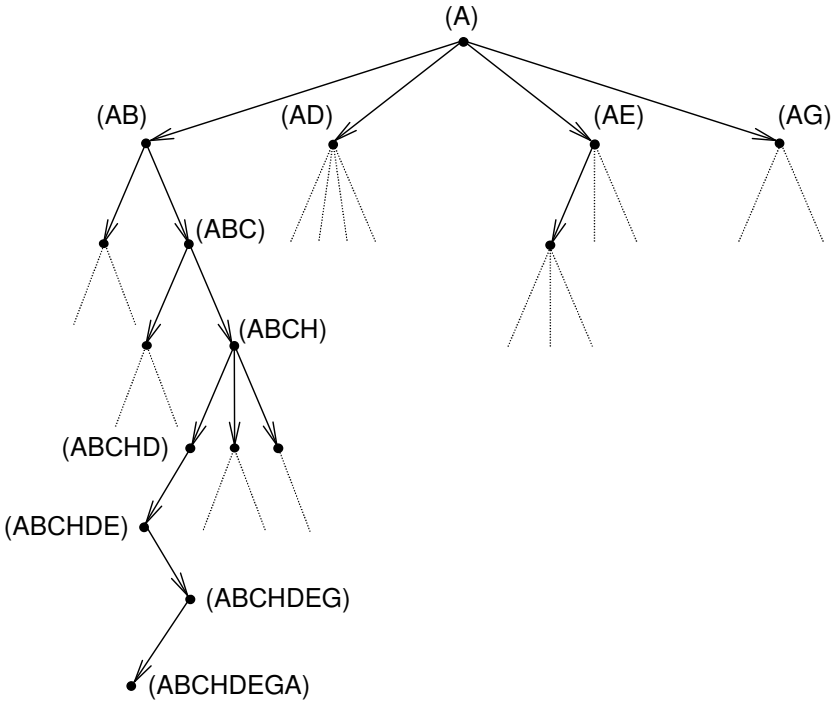


Ilustración 4. Problema del viajero:
un fragmento del espacio de estados.

Ahora vamos a analizar otro problema bien conocido sobre un mono y un plátano. En un cuarto se encuentran un mono, una caja y un plátano que está colgado en el techo, tan alto que el mono puede alcanzarlo solamente parándose sobre la caja. Hay que encontrar una secuencia de acciones que permitieran al mono alcanzar el plátano. Se supone que el mono puede moverse en el cuarto, empujar la caja por el piso, subir a la caja y agarrar el plátano.

Claro está, que representación de los estados de este problema debe incluir los siguientes elementos: posición del mono en el cuarto, tanto horizontalmente —respecto al piso—, como verticalmente (es decir, si el mono se encuentra en el piso o se encuentra sobre la caja), posición de la caja en el piso y el hecho si el mono ya tiene el plátano o no. Todo eso se puede presentar como una lista de cuatro elementos ($posición_{mono}$ $vertical_{mono}$ $posición_{caja}$ *meta*) donde

$posición_{mono}$ corresponde a la posición del mono en el piso; eso puede ser un vector bidimensional de coordenadas.

$posición_{caja}$ es posición de la caja en el piso

$vertical_{mono}$ corresponde a la posición del mono en el piso (denotémoslo como PISO), o a la posición del mono sobre la caja (denotémoslo como CAJA).

meta es un número 0 o 1 que corresponde al hecho de que si el mono ya tiene el plátano o no.

También vamos a fijar tres puntos importantes en la superficie del piso.

P_{mono} es el punto de la posición inicial del mono.

P_{caja} es el punto de la posición inicial de la caja.

$P_{plátano}$ es el punto del piso ubicado inmediatamente debajo del plátano.

Entonces, el estado inicial del problema se describe con la lista (P_{mono} P_{caja} PISO 0), y el estado meta se representa como cualquier lista cuyo último elemento es igual a 1.

Los operadores en este problema se pueden definir de la manera siguiente:

1. *Moverse (W)* = el mono se mueve al punto W en la superficie del piso.
2. *Mover_ caja (V)* = el mono mueve la caja al punto V del piso.
3. *Subir* = el mono sube a la caja.
4. *Agarrar* = el mono agarra el plátano.

Las condiciones de aplicación de sus cuatro operadores pueden ser definidas como las reglas del siguiente tipo: "argumentos del operador → resultados del operador".

Recordemos que usamos la representación de un estado con la lista de cuatro elementos ($posición_{mono}$ $vertical_{mono}$ $posición_{caja}$ *meta*) donde el significado de cada elemento se define por la posición en esta lista: primer lugar en la lista corresponde a la posición del mono sobre el piso, etc.

Las letras X, Y, Z, W, V corresponden a las variables que pueden tener cualquier valor.

1. *Moverse (W)* : (X PISO Y Z) → (W PISO Y Z).
2. *Mover_ caja (V)* : (X PISO X Z) → (V PISO V Z).
3. *Subir* : (X PISO X Z) → (X CAJA X Z).
4. *Agarrar* : ($P_{plátano}$ CAJA $P_{plátano}$ 0) → ($P_{plátano}$ CAJA $P_{plátano}$ 1)

Si vamos a considerar que los puntos especiales mencionados anteriormente P_{mono} P_{caja}.$P_{plátano}$ son importantes para resolver el problema entonces vamos a

obtener el espacio de estados presentado en la Ilustración 5. Este espacio de estados contiene solamente 13 estados, las aristas están marcadas con el número del operador aplicable. El espacio de estados contiene cuatro ciclos de las jugadas del mono entre los tres puntos especiales importantes, con o sin la caja. En el espacio de estados también existen dos caminos que no llevan a la solución, cuando el mono sube a la caja, pero la caja no está debajo del plátano. El camino de la solución está marcado en la ilustración. El camino contiene cuatro operadores:

Moverse (P_{caja}); Mover_ caja ($P_{plátano}$); Subir, Agarrar

La notación que corresponde a la Ilustración 5 es la siguiente: $P_m = P_{mono}$, $P_c = P_{caja}$, $P_p = P_{plátano}$, P = PISO, C = CAJA.

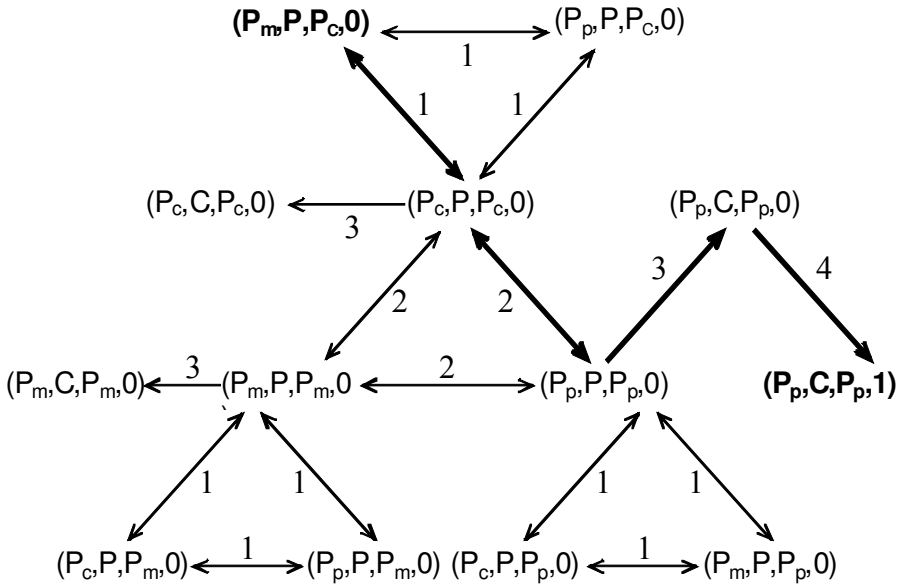


Ilustración 5. Espacio de estados del problema sobre el mono.

El ejemplo presentado demuestra que la selección de una representación natural es muy importante para la solución eficaz del problema. Este espacio de estados tan pequeño fue obtenido como consecuencia de que se ignoraron casi todos los puntos del piso menos los tres puntos importantes: P_{mono}, P_{caja}, $P_{plátano}$.

Una estrategia muy poderosa para la reducción de los espacios de estados es la aplicación de *esquemas de estados* o *esquemas de operadores,* en los cuales se utilizan variables para la descripción de estados y de operadores. De esta manera un esquema de estados describe un conjunto de estados, y no solo un estado, y un esquema de operadores describe un conjunto de acciones de cierto

tipo. En el ejemplo presentado utilizamos los esquemas de operadores, pero no utilizamos los esquemas de estados.

2.3 Estrategias de búsqueda en los espacios de estados

Como ya hemos mencionado la búsqueda en un espacio de estados se basa en la construcción de una ruta secuencial sobre los nodos de un grafo de estados hasta llegar a un estado meta. Vamos a introducir algunos términos que se utilizarán al describir varios algoritmos de búsqueda.

Es natural llamar al nodo de un grafo que corresponde al estado inicial un *nodo inicial*, y al nodo que corresponde al estado meta un *nodo meta*. Los nodos que siguen inmediatamente a algún otro nodo vamos a llamarlos *nodos hijos*, y al mismo nodo un *nodo padre*. La operación principal que se realiza durante la búsqueda en un grafo es la ***apertura del nodo***, lo que significa que se construyen (se generan) sus nodos hijos por medio de aplicación de todos los operadores aplicables al estado correspondiente.

La búsqueda en un espacio de estados se puede imaginar como un proceso de la apertura de los nodos y verificación de las propiedades de los nodos construidos. Es importante mencionar que durante este proceso se deben mantener los *apuntadores* de todos los nodos hijos hacia sus nodos padre. Precisamente esos apuntadores permitirán reconstruir la ruta hacia el nodo inicial después de que se generó un nodo meta. Esta ruta en la dirección inversa, o más bien, la secuencia de aplicación de los operadores que corresponden a las aristas de esta ruta, será la *solución del problema*.

Los nodos y los apuntadores construidos durante el proceso de la búsqueda representan un sub-árbol del grafo que corresponde al espacio de estados del problema; es decir, de todo que fue definido al formalizar el problema en cuestión. Este sub-árbol se llama ***árbol de la búsqueda.***

Los procedimientos de la búsqueda en un espacio de estados pueden tener varias características adicionales:

- Uso de la información heurística.
- Orden de apertura de nodos.
- Búsqueda completa vs. búsqueda incompleta.
- Dirección de la búsqueda.

Según la primera característica se defieren la búsqueda ***ciega*** y la búsqueda ***heurística***. En caso de la búsqueda ciega, la posición de cada nodo no influye al orden en que se abren los nodos. A diferencia, la búsqueda heurística usa la información a priori —heurística— sobre la estructura general del espacio de estados y/o donde en este espacio de estados puede estar un nodo meta, por eso se abre primero un nodo más prospectivo. En caso general eso permite reducir la búsqueda.

Existen dos tipos básicos de la búsqueda ciega que dependen del orden de apertura de los nodos: búsqueda en ***profundidad*** y búsqueda en ***amplitud***.

Tanto búsqueda ciega como búsqueda heurística pueden ser completas o incompletas. En una búsqueda ***completa*** se verifica cada nodo del grafo que corresponde al espacio de estados del problema. De esa manera, el hecho de encontrar la solución, si ésta existe, está garantizada. En una búsqueda ***incompleta*** se verifica solamente una parte del grafo, y si esta parte no contiene los nodos meta, la solución no será encontrada.

En cuanto a la dirección de la búsqueda se distinguen entre la búsqueda ***directa***, que se hace desde un nodo inicial hasta un nodo meta; y la búsqueda ***inversa*** que se realiza desde un nodo meta hacia un nodo inicial. Otra posibilidad es una búsqueda ***bidireccional***, donde se aplica la búsqueda directa y la búsqueda inversa en turnos. Se puede utilizar la búsqueda inversa cuando los operadores del problema son inversos.

Usualmente para resolver los problemas se aplica la búsqueda directa, la que vamos a considerar en el resto del capítulo.

2.4 Búsqueda ciega

Los algoritmos de búsqueda ciega —en profundidad (*depth first*) y en amplitud (*breadth first*)— se distinguen por la manera de seleccionar al nodo que se abre en el siguiente paso del algoritmo. En el algoritmo de búsqueda en amplitud los nodos se abren en el mismo orden que se construyen. A cambio, en el algoritmo de búsqueda en profundidad se abren primero los nodos construidos al último.

Primero vamos a analizar esos algoritmos para los espacios de estados que son árboles —la raíz del árbol es el nodo inicial. Después presentamos las modificaciones necesarias de esos algoritmos para las búsquedas en cualquier tipo de grafos (no solo árboles). De manera general, es mucho más fácil organizar una búsqueda en un árbol, ya que al construir un nuevo estado (y el nodo correspondiente) podemos estar seguros que este estado nunca se ha construido antes ni se construirá en el futuro.

Los algoritmos de búsqueda que estamos viendo utilizan las listas que llamaremos `Open` (abiertos) y `Closed` (cerrados), que contienen los nodos abiertos y los nodos cerrados del espacio de estados correspondientemente. Llamemos al nodo actual, es decir, el nodo que está siendo abierto, `Current`. Al formular los algoritmos suponemos que el nodo inicial no es al mismo tiempo el nodo meta.

El algoritmo base de **búsqueda en amplitud** consiste en los siguientes pasos:

Paso 1. Colocar el nodo inicial en la lista de los nodos cerrados `Closed`.

Paso 2. Si la lista `Closed` está vacía, terminar la búsqueda y mandar un mensaje de error, en caso contrario (la lista `Closed` contiene algún elemento) continuar al paso 3.

Paso 3. Tomar el **primer** nodo de la lista `Closed`. Vamos a llamar este nodo `Current` (actual). Mover este nodo a la lista de los nodos abiertos `Open`.

Paso 4. Abrir el nodo `Current` construyendo sus nodos hijos. Si no existen nodos hijos, regresar al paso 2. Si nodos hijos existen, moverlos (en cualquier orden) **al final** de la lista `Closed` y construir *apuntadores* que indican que su nodo padre era el nodo `Current`.

Paso 5. Verificar si algún nodo hijo es un nodo meta. De ser así, terminar la búsqueda y presentar la solución del problema que se obtienen pasando por los apuntadores del nodo meta hacia el nodo inicial. En caso contrario continuar con el paso 2.

Fin del algoritmo.

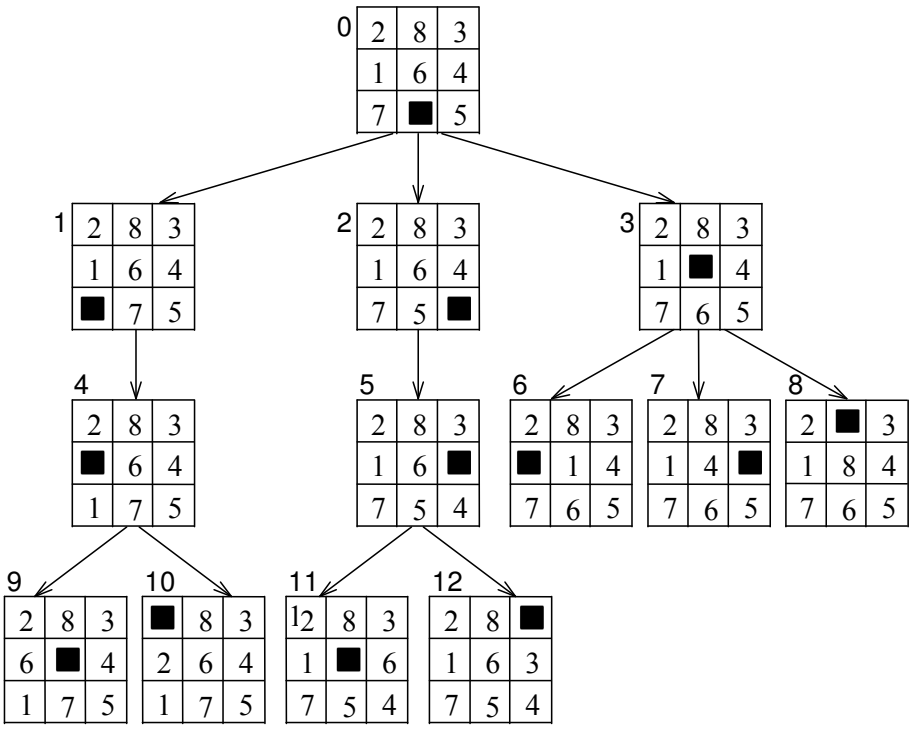


Ilustración 6. Búsqueda en amplitud para el juego de "ocho".

Las bases de este algoritmo es un ciclo de la apertura consecutiva de los nodos terminales (hojas) del árbol de la búsqueda que se guardan en la lista

`Closed`. El algoritmo de búsqueda en amplitud presentado es completo. Se puede demostrar que al aplicar el algoritmo de búsqueda en amplitud necesariamente se encontrará la ruta más corta hacia algún nodo meta si esta ruta existe. Si la ruta que soluciona el problema no existe, entonces en caso de un espacio de estados finito, el algoritmo terminará su funcionamiento y mandará un mensaje de error; en caso de un espacio de estados infinito el algoritmo no terminará.

En la Ilustración 6 se presenta una parte del árbol construida al aplicar el algoritmo de búsqueda en amplitud a alguna configuración inicial del juego de "ocho". La ejecución del algoritmo fue interrumpida después de construir los primeros 12 nodos, de los cuales 6 nodos fueron abiertos. Los nodos del árbol contienen descripciones de los estados correspondientes, las flechas indican los nodos hijos. Los nodos están numerados según el orden en el cual ellos fueron construidos durante la búsqueda. Se considera que el orden de la construcción de los nodos hijos corresponde a una manera fija de mover la ficha vacía: izquierda, derecha, arriba, abajo.

También se supone que la operación de apertura de un nodo está organizada de tal manera que no genera un estado idéntico del nodo hijo al nodo padre del nodo que estamos abriendo. Por ejemplo, en el estado 1 no vamos a generar un estado idéntico al estado 0.

Mencionamos que en el caso que el algoritmo continuara su funcionamiento, se abrirían los nodos 6, 7, u 8, dado que ellos se encuentran al inicio de la lista de los nodos que no están abiertos todavía (`Closed`).

Antes de presentar el algoritmo de **búsqueda en profundidad** es necesario definir el concepto de ***profundidad del nodo*** en un árbol de la búsqueda. Se puede hacerlo de siguiente manera recursiva:

- Profundidad del nodo que corresponde a la raíz del árbol es igual a cero.
- Profundidad de cada nodo diferente de la raíz es igual a profundidad de su nodo padre más uno.

En el algoritmo de búsqueda en profundidad hay que abrir primero el nodo que tiene mayor profundidad. Este principio puede llevar a un proceso que nunca termina, en caso si el espacio de estados es infinito y la búsqueda continúa en una rama infinita del árbol que no contenga un nodo meta. Por eso es necesario introducir uno u otro límite de la búsqueda; lo que se hace normalmente es limitar la profundidad de búsqueda en el árbol. Eso significa que se puede construir solamente los nodos que tienen la profundidad que es menor de algún ***umbral de profundidad*** (o ***profundidad límite***). De esa manera, se abren los nodos con mayor profundidad pero solamente si esta profundidad es menor que el umbral mencionado. El algoritmo de búsqueda correspondiente se llama ***búsqueda en profundidad limitada***.

Los pasos del algoritmo base de búsqueda en profundidad limitada (con umbral de profundidad D) son los siguientes.

Paso 1. Colocar el nodo inicial en la lista de los nodos cerrados `Closed`.

Paso 2. Si la lista `Closed` está vacía, terminar la búsqueda y mandar un mensaje de error, en caso contrario (la lista `Closed` contiene algún elemento) continuar al paso 3.

Paso 3. Tomar el **primer** nodo de la lista `Closed`. Vamos a llamar este nodo `Current` (actual). Mover este nodo a la lista de los nodos abiertos `Open`.

Paso 4. Si la profundidad del nodo actual (`Current`) es igual al umbral de profundidad D, regresar al paso 2; en caso contrario continuar con el siguiente paso.

Paso 5. Abrir el nodo `Current` construyendo sus nodos hijos. Si no existen nodos hijos, regresar al paso 2. Si nodos hijos existen, moverlos (en cualquier orden) **al inicio** de la lista `Closed` y construir *apuntadores* que indican que su nodo padre era el nodo `Current`.

Paso 6. Verificar si algún nodo hijo es un nodo meta. De ser así, terminar la búsqueda y presentar la solución del problema que se obtiene pasando por los apuntadores del nodo meta hacia el nodo inicial. En caso contrario continuar con el paso 2.

Fin del algoritmo.

Este algoritmo se parece bastante al algoritmo de la búsqueda en amplitud, siendo la única diferencia el límite de la profundidad (Paso 4), y el lugar en la lista `Closed` donde se agregan los nodos hijos (Paso 5).

Dado que en la profundidad es limitada, este algoritmo de búsqueda aplicado a los árboles siempre termina su funcionamiento. A diferencia del algoritmo de búsqueda en amplitud, su búsqueda es incompleta, dado que un nodo meta puede encontrarse debajo del umbral de la profundidad; en este caso este nodo meta no será encontrado.

En la Ilustración 7 se muestra un fragmento del árbol de la búsqueda construido por el algoritmo de búsqueda en profundidad limitada con el umbral igual a 4. Como un estado inicial tomamos el mismo estado que tiene el ejemplo presentado en la Ilustración 6. Durante la búsqueda también fueron construidos 12 nodos, de los cuales fueron abiertos 7. Los nodos están numerados según el orden en el cual se construyeron. Es fácil verificar comparando las dos ilustraciones mencionadas, que los algoritmos de búsqueda en profundidad limitada y de búsqueda en amplitud construyeron dos diferentes árboles de búsqueda.

Se nota que el algoritmo de búsqueda en profundidad primero realiza la búsqueda siguiendo una ruta hasta llegar a la profundidad límite; después se analizan las rutas de la misma o menor profundidad, cuya diferencia de la ruta anterior consiste en el último nodo; después se consideran las rutas que tienen diferencia en los dos últimos nodos, etc.

Si vamos a continuar la ejecución de los algoritmos de búsqueda en profundidad y en amplitud para el estado inicial del juego de "ocho" presentado en las ilustraciones 6 y 7, teniendo como propósito encontrar la configuración presentada en la Ilustración 1 (b), entonces ésta será encontrada

en la profundidad 5; el algoritmo de búsqueda en amplitud abrirá 26 nodos y construirá 46 nodos, mientras que el algoritmo de búsqueda en profundidad abrirá 18 nodos y construirá 35 nodos (Nilsson 1973, sección 3.2 o Nilsson 1985, sección 2.4).

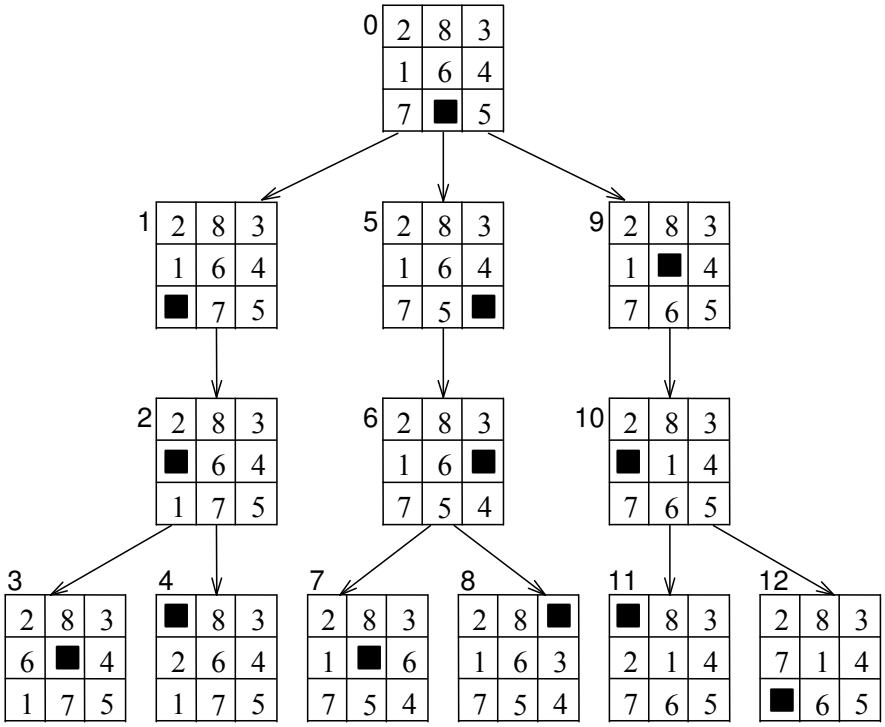


Ilustración 7. Búsqueda en profundidad limitada para el juego de "ocho."

En general, los algoritmos de búsqueda en profundidad y en amplitud son comparables respecto a su efectividad, más específicamente respecto al número de los nodos construidos. Al mismo tiempo, el algoritmo de búsqueda en profundidad tiene cierta ventaja: en los casos cuando la búsqueda empezó en la rama del árbol que contenga el estado meta, la solución se encontrará más rápidamente que utilizando la búsqueda en amplitud.

Se puede observar que en el ejemplo que hemos visto relacionado con el juego de "ocho", los algoritmos de búsqueda en profundidad y en amplitud formulados para los espacios de estados que se representan por árboles, se aplicaban al espacio de estados que puede ser representado por grafo. La diferencia es que este grafo puede tener ciclos, ya que el mismo estado del

juego puede ser alcanzado como resultado de aplicación de varias secuencias de los operadores.

En algunos casos los ciclos no son obstáculos para los algoritmos base de búsqueda ciega para encontrar la solución, si la solución existe, y terminar su funcionamiento. Sin embargo, en otros casos los algoritmos pueden entrar en un ciclo infinito dado la reduplicación múltiple de espacios de estados en el árbol de la búsqueda.

Es necesario subrayar, que tanto para la búsqueda en profundidad, como por la búsqueda en amplitud se construye exactamente un árbol de la búsqueda, aunque si el espacio de estados es un grafo con ciclos. Es así, porque al abrir los nodos los apuntadores de los nodos hijos apuntan solamente a un nodo padre. En caso de un grafo (espacio de estados) aleatorio el árbol de la búsqueda puede contener los estados duplicados. Durante la búsqueda el grafo se abre y se convierte en un árbol dado que algunos estados se repiten varias veces. Por ejemplo, para el juego de "ocho", según el convenio de la apertura de los nodos, al construir el árbol de la búsqueda se excluyen solamente los estados repetidos que se encuentran a lo máximo en dos pasos más arriba en el árbol; mientras que en los estados repetidos los estados más lejanos son posibles.

Para evitar esta repetición de los estados en caso de la búsqueda en los grafos de manera general es necesario agregar algunas modificaciones bastante obvias en los algoritmos base de búsqueda en profundidad y en amplitud.

En el algoritmo de búsqueda en amplitud es necesario verificar adicionalmente si cada nodo construido (más bien la descripción del estado correspondiente) se encuentre en las listas `Open` y `Closed`, dado que ya se construyó anteriormente como el resultado de la apertura de algún otro nodo. De ser así, no es necesario meter este nodo en la lista `Closed` otra vez. En el algoritmo de búsqueda en profundidad limitada, además de la verificación mencionada, es necesario realizar un recálculo de la profundidad de sus nodos hijos construidos, que ya están presentes en la lista `Open`, o bien en la lista `Closed`.

Generalmente hablando, los algoritmos de búsqueda ciega no son métodos efectivos de búsqueda de soluciones. En caso de algunos problemas no triviales es imposible utilizarlos dado un número grande de los nodos construidos. Realmente, si `L` es la longitud de la ruta de la solución, y `B` es el número de ramas (nodos hijos) de cada nodo, entonces en un caso general debemos explorar B^L rutas desde el nodo inicial. Este número se aumenta exponencialmente al aumentar la longitud de la ruta solución, lo que se llama la explosión combinatoria.

Una de las maneras de aumentar la efectividad de la búsqueda está relacionada con el uso de la información que tome en cuenta las características específicas del problema en cuestión y que permita moverse hacia la meta con cierta inteligencia. Este tipo de información específica del problema se llama

información *heurística*, y los algoritmos y métodos correspondientes también se llaman heurísticos.

2.5 Búsqueda heurística

La idea principal que está detrás de la búsqueda heurística está relacionada con la posibilidad de evaluar cada nodo que todavía está cerrado utilizando la información heurística desde el punto de vista de que tan prometedor es, y escoger para la apertura y continuación de la búsqueda un nodo más prospectivo.

El modo más común de utilizar la información heurística es introducir la función heurística de evaluación `Est(V)`. Esta función se define sobre un conjunto de nodos del espacio de estados y obtiene valores numéricos. El valor de la función heurística de evaluación se interpreta como un prospecto de apertura del nodo, o a veces, como la probabilidad de su permanencia en la ruta de solución. Usualmente el convenio es que el menor valor de la función `Est(V)` corresponde a un nodo más prospectivo, y los nodos se ordenan según el orden de aumento de los valores de la función heurística de evaluación.

La secuencia de los pasos del algoritmo de la búsqueda heurística (ordenada) parece a la secuencia de los pasos de los algoritmos de la búsqueda ciega; la diferencia consiste en el uso de la función heurística de evaluación. Después de la construcción de un nuevo estado se hace su evaluación, es decir se calcula el valor de dicha función. Las listas de los nodos abiertos y cerrados contienen tanto los nodos, como sus evaluaciones que se usan para ordenar la búsqueda.

En el ciclo cada vez se elige un nodo más prometedor para su apertura en el árbol de la búsqueda. Igual que en los casos de los algoritmos de la búsqueda ciega, el conjunto de los nodos y apuntadores generados por el algoritmo representa un árbol; las hojas de este árbol son los nodos todavía cerrados.

Suponemos que el espacio de estados utilizado por el algoritmo es un árbol. Los pasos principales del **algoritmo base de la búsqueda heurística** (*best_first_search*) son los siguientes:

Paso 1. Colocar el nodo inicial en la lista de los nodos cerrados `Closed` y calcular el valor de la función de evaluación.

Paso 2. Si la lista `Closed` está vacía, terminar la búsqueda y mandar un mensaje de error, en caso contrario (la lista `Closed` contiene algún elemento) continuar al paso 3.

Paso 3. Tomar de la lista `Closed` un nodo con el menor valor de la función de evaluación (en caso de que sean varios nodos con el mismo valor mínimo de evaluación, se puede tomar cualquier nodo de esos). Vamos a llamar este nodo `Current` (actual). Mover este nodo a la lista de los nodos abiertos `Open`.

Paso 4. Si el nodo actual (`Current`) es un nodo meta, terminar la búsqueda y presentar la solución del problema que se obtiene pasando por los apuntadores del nodo meta hacia el nodo inicial. En caso contrario continuar con el paso 5.

Paso 5. Abrir el nodo `Current` construyendo sus nodos hijos. Si no existen nodos hijos, regresar al paso 2. Si nodos hijos existen pasar al paso 6.

Paso 6. Para cada nodo hijo calcular su valor de la función de evaluación, mover todos los nodos hijos a la lista `Closed` y construir los apuntadores que contienen la referencia de los nodos hijos al nodo padre `Current`. Continuar con el paso 2.

Fin del algoritmo.

Es necesario mencionar que la búsqueda en profundidad puede ser interpretada como un caso particular de la búsqueda heurística con la función de evaluación `Est(V) = d(V)`; lo mismo se aplica para la búsqueda en amplitud donde la función de evaluación es `Est(V) = 1/d(V)`. `d(V)` es la profundidad del nodo `V`.

Para adaptar el algoritmo mencionado al caso general de la búsqueda en cualquier grafo (espacio de estados), es necesario prever la reacción del algoritmo en caso de la construcción de los nodos hijos que ya están presentes o bien en la lista de los nodos abiertos, o bien en la lista de los nodos todavía cerrados.

En general, la función heurística de evaluación puede depender no solamente de las propiedades internas del estado que se está evaluando, es decir, de las propiedades de los elementos que pertenecen a este estado, sino de las características de todo el espacio de estados, por ejemplo, de la profundidad del nodo que se evalúa dentro del árbol de la búsqueda u otras propiedades de la ruta hacia este nodo. Por eso el valor de la función de evaluación para un nodo hijo apenas construido, pero que ya se encuentre en la lista `Open` o `Closed` lo que significa que ya se había construido anteriormente, puede ser menor. Por eso hay que corregir el valor anterior de evaluación de este nodo cambiándolo por el valor nuevo, si éste es menor. Si el nodo construido con el valor corregido pertenece a la lista `Open`, es necesario volver a colocarla en la lista `Closed`, pero con el valor de evaluación menor. También es necesario cambiar los apuntadores de todos los nodos en las listas `Open` y `Closed`, la regulación de los cuales se disminuyó, dirigiendo los al nodo `Current`.

Sin embargo si la función de evaluación solamente toma en cuenta las propiedades internas de los nodos (estados), entonces para prevenir formación de ciclos es suficiente una modificación más simple del algoritmo: hay que excluir el duplicado de estados en las listas `Open` y `Closed`, dejando solamente un estado (es decir, prohibiendo la repetición de los estados).

Damos un ejemplo de funcionamiento del algoritmo de la búsqueda heurística usando los mismos datos del juego de "ocho", que en las ilustraciones 6 y 7, donde se representa el estado inicial, y el estado meta

mismo que ha representado en la ilustración 1 (b). Vamos a utilizar como la función de evaluación de siguiente simple función:

Est(1) = d(V) + k(V)

donde d(V) es la profundidad del nodo V, o el número de las aristas en la ruta de este nodo hacia el nodo inicial;

k(V) número de él las fichas de la posición (estado, nodo) V que se encuentran en un lugar incorrecto. Un lugar de una ficha es incorrecto si es diferente del lugar de esta ficha en el nodo meta.

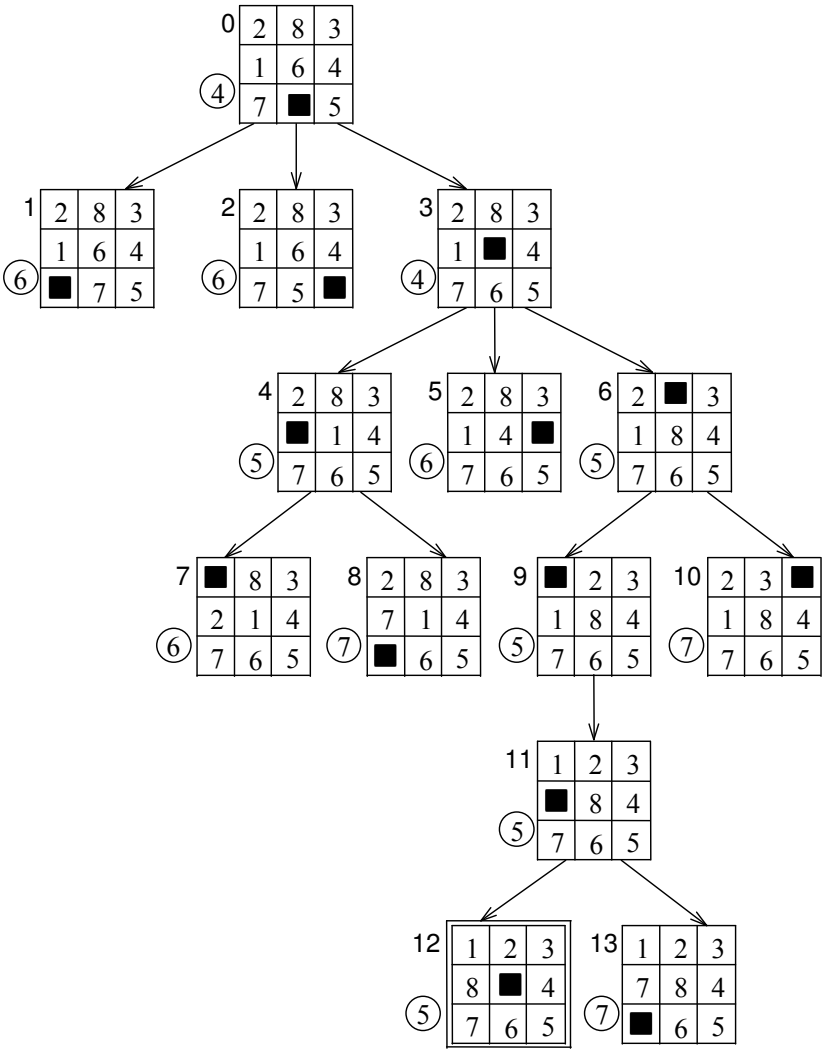


Ilustración 8. Búsqueda heurística para el juego de "ocho".

En la Ilustración 8 se muestra un árbol construido por el algoritmo de la búsqueda heurística usando la función de evaluación mencionada. La emulación de cada nodo se encuentra dentro de un círculo cerca del nodo. Los números al lado de cada estado sin alguna marca especial, igual que antes corresponden al orden de construcción de los nodos. El nodo 12 está marcado ya que corresponde al nodo meta.

Se puede observar, que dado que la selección de un nodo con la menor evaluación se hace dentro de todo el árbol construido en cada momento, los nodos que se abren uno tras otro pueden estar en las partes diferentes de este árbol. La función de evaluación es tal que si las dos posiciones tienen el mismo peso, se da la ventaja a un nodo con menor profundidad.

La solución del problema con longitud de cinco jugadas fue encontrada abriendo 6 nodos y construyendo 13 en nodos; es mucho menos que con la aplicación de la búsqueda ciega: búsqueda en amplitud, 26 y 46 nodos, búsqueda en profundidad: 18 y 35 nodos. De esa manera el uso de la información heurística permite deducir significativamente el espacio de la búsqueda.

La efectividad de los algoritmos de la búsqueda se puede evaluar utilizando el siguiente concepto P, que corresponde al *acercamiento a la meta*. Se calcula según la siguiente fórmula:

P = L / N

Donde L es la longitud de la ruta encontrada (es equivalente a la profundidad del nodo meta);

N es el número total de nodos construidos durante la búsqueda.

Se puede observar que P = 1 si solamente se construyen los nodos de la ruta de solución, en cualquier otro caso P $<$ 1. En general, este número es menor si vamos construyendo mayor número de los nodos inútiles. De esta manera este criterio muestra que tanto el árbol construido durante la búsqueda está alargado, y no ancho. Para los ejemplos mencionados relacionados con el juego de "ocho" este criterio tiene los valores: 5/13 para la búsqueda heurística, 5/46 para la búsqueda en amplitud, 5/35 para la búsqueda en profundidad.

Claro está que el algoritmo de búsqueda heurística con una bien seleccionada función de evaluación encuentra la solución más rápido que los algoritmos de la búsqueda ciega. Sin embargo una buena selección de una función de evaluación es un paso más difícil durante la formalización del problema; a menudo esta selección se hace de manera experimental.

En el mismo problema se puede comparar diferentes funciones de evaluación según su *fuerza heurística*, es decir, que tan más rápido se hace la búsqueda. Es necesario mencionar que la fuerza heurística debe tomar en cuenta un total de los cálculos realizados durante la búsqueda, por eso además de tomar en cuenta el número de los nodos abiertos o construidos es necesario tomar en cuenta la complejidad de cálculo de la misma función de evaluación.

Para el juego de "ocho" podemos sugerir otra función heurística de evaluación:

Est2(V) = d(V) + s(V)

d(V) de igual manera que en la función anterior Est1 corresponde a la profundidad del nodo. s(V) es la sumatoria que se obtiene calculando para cada de las ocho fichas las dos distancias —vertical y horizontal— entre la celda en el estado actual y en el estado meta. Después se suman esas distancias para las ocho fichas. De esta manera el valor de s(V) para reflejar la distancia total para todas las fichas de su posición meta.

Por ejemplo, para la configuración inicial en la ilustración 8 la distancia de la ficha 8 de su posición en la configuración meta es 1 tanto verticalmente como horizontalmente, y su sumatoria es igual a 2. La sumatoria total para todas las fichas es igual a 5, dado que las fichas 3, 4, 5, 7 ya se encuentran en su lugar, y su aportación a esta sumatoria es igual a 0. Es intuitivamente claro, y se puede mostrar con ejemplos, que esta función tiene mayor fuerza heurística, es decir, dirige la búsqueda de manera más efectiva.

2.6 Consideraciones de la búsqueda heurística

Queda una pregunta importante, si el algoritmo de la búsqueda heurística con la función de evaluación general —es decir, sin algún tipo de limitaciones—, garantiza que encontrará la solución haciendo un número finito de pasos, en los casos cuando la solución existe; esto es lo que garantiza el algoritmo de búsqueda en amplitud. Claro está, que no podemos estar seguros de eso, especialmente para los problemas con espacios de estados infinitos. En general, a menudo se presenta la situación cuando una heurística que reduce significativamente la búsqueda para algún tipo de problemas (estados iniciales y estados meta), para otro tipo de problemas no reduce el espacio de la búsqueda, incluso se puede buscar la solución haciendo más pasos que en la búsqueda ciega, o no permite encontrar la ruta solución.

La exploración matemática del algoritmo de la búsqueda heurística, específicamente las condiciones que garanticen que el algoritmo encuentre la solución, fue realizada para las funciones de evaluación heurística de equipo especial y para un problema más complejo que hemos considerado hasta el momento —búsqueda de cualquier ruta solución hasta un nodo meta (Nilson, 1971; Winston, 1984).

Suponemos que sobre el conjunto de las aristas del espacio de estados está definida una función de costo:

$c(V_A, V_B)$ es un costo de transición (arista) del nodo V_A hasta el nodo V_B.

Vamos a definir el ***costo*** de cualquier ruta en el espacio de estados (grafo), como la sumatoria de las aristas que forman la ruta. Que nuestro propósito sea no solamente encontrar la ruta solución, sino ***la ruta solución óptima***, es decir, con el menor costo.

Suponemos que la función heurística de evaluación Est(V) está construida de tal manera que permite evaluar el costo de la ruta solución óptima, que van desde el nodo inicial hasta uno de los nodos meta, con la condición que esta ruta pasa a través del nodo V. Entonces, el valor de esta función de evaluación se puede representar como la sumatoria:

Est(V) = g(V) + h(V)

donde g(V) es la evaluación de la ruta óptima desde el nodo inicial hasta el nodo V, y h(V) es la evaluación de la ruta óptima desde el nodo V hasta el nodo meta.

Si durante el proceso de la búsqueda del nodo V ya está construido, entonces existe la ruta hasta este nodo, y se puede calcular su costo. Esta ruta no exactamente debe ser la ruta óptima. Es posible que exista otra ruta con menor costo que todavía no está encontrada. Sin embargo, el costo de la ruta encontrada g(V) puede ser usado como una evaluación de la ruta con el costo mínimo. Mientras h(V) puede ser sugerido tomando en cuenta las consideraciones heurísticas, que dependen del problema, como una característica de evaluación del nodo V que describe su proximidad hacia el nodo meta. De esta manera, la información heurística solamente se usa para el cálculo de h(V) en la función de evaluación.

Una variante de la búsqueda heurística, que se usa para la búsqueda de la ruta óptima de solución y utiliza la función heurística de evaluación del tipo mencionado, se conoce como A-algoritmo (Nillson, 1980). Se demostraron varias propiedades importantes de este algoritmo, específicamente su admisibilidad.

Un algoritmo de búsqueda es ***admisible*** (*válido*), si para un grafo aleatorio él siempre termina su funcionamiento encontrando una ruta óptima hacia un nodo meta, en caso que este ruta exista.

Denotamos como h*(V) el costo de la ruta óptima de un nodo arbitrario V hacia un nodo meta. Entonces se cumple el siguiente **teorema de admisibilidad de A-algoritmo:**

A- algoritmo que usa alguna función heurística

Est(V) = g(V) + h(V)

Donde g(V) es el costo de la ruta desde el nodo inicial hasta el nodo V en el árbol de la búsqueda, y h(V) es la evaluación heurística de la ruta óptima del nodo V al nodo meta. El algoritmo es admisible si h(V) h*(V) para todos los nodos V del espacio del estados.

A-algoritmo de la búsqueda de heurística con la función h(V) que cumple con la condición mencionada se llama A*-algoritmo (Nilsson, 1971).

El significado práctico de este teorema consiste en el hecho que para determinar si A-algoritmo es admisible es suficiente encontrar el límite inferior de la función h*(V) y utilizarla como función h(V), entonces se garantiza que la ruta encontrada será óptima.

Si se toma el límite inferior trivial, es decir establecer que h(V)=0 para todos los nodos del espacio de estados, la admisibilidad está garantizada. Sin

embargo este caso corresponde a la completa ausencia de la información heurística sobre el problema, la función Est no tiene ninguna fuerza heurística, es decir, no disminuye la búsqueda. A*-algoritmo se comporta de manera igual a la algoritmo de búsqueda en amplitud.

Hablando con mayor exactitud, cuando Est(V)=g(V), donde g(V) es el costo de la ruta del nodo inicial hasta el nodo V, es el algoritmo conocido como el algoritmo de costos iguales. El algoritmo de costos iguales representa una variante general del algoritmo de búsqueda en amplitud donde los nodos se abren según el orden de aumento del costo g(V), es decir, primero se abren los nodos de la lista de los nodos cerrados, para los cuales la función g tiene menor valor.

Además si vamos a suponer un costo c(VA, VB)=0 para todas las aristas de espacio de estados, entonces el A*-algoritmo se convierte en una ineficaz búsqueda ciega en amplitud.

Ambas funciones propuestas anteriormente Est1 y Est2 para el juego de "ocho" cumplen con el criterio de admisibilidad del A*-algoritmo. El componente d(V) es el costo de la ruta al nodo V dado que todas las aristas tienen el costo de c(VA, VB)=1. La diferencia entre esas dos funciones es el segundo componente; se puede mostrar que los valores de la segunda función siempre (es decir, para todos los estados), es mayor que los valores de la primera función, Est1(V) Est2(V), lo que es equivalente a k(V) s(V). Realmente, en la segunda función la aportación de cada ficha en la evaluación (sumatoria) general s(V), o bien es 0 —la ficha y a ésta en su lugar, o bien no es menor que 1 —en caso contrario. En la primera función, la aportación de cada ficha en la evaluación k(V) o bien es igual a 0 o a 1.

De la última desigualdad se deduce que es suficiente demostrar la admisibilidad solamente para la segunda función Est2. Lo que la condición correspondiente s(V) h*(V) se cumple, se demuestra de manera siguiente. Si las fichas no hubieran tenido otras fichas como obstáculos en su jugada y podrían moverse como si el campo de juego hubiera sido vacío, entonces la sumatoria de longitudes de las rutas para todas las fichas hubiera sido igual a s(V). En realidad, las fichas a menudo no pueden moverse utilizando la ruta más corta, dato que hay otras fichas en el campo de juego. Entonces, la longitud (el costo) de la ruta óptima h*(V) no es menor que s(V).

Es importante notar que la función s(V) no toma en cuenta qué tan difícil es el cambio de posiciones entre las dos fichas, y entonces, su fuerza heurística puede ser aumentada en principio. En algunos casos la fuerza heurística de una función puede ser aumentada multiplicándola por una constante positiva mayor que 1, sin embargo a veces este aumento causan la pérdida de admisibilidad del algoritmo. Por ejemplo, para el juego de "ocho" como otro componente de la fórmula de la función heurística podemos tomar h(V)=2*s(V); en algunos casos esta función puede acelerar la búsqueda y permitir resolver problemas más difíciles, sin embargo se dejará de cumplir la

condición de admisibilidad; por ejemplo para el estado inicial de la ilustración 15, h*(V) s(V).

Si la desigualdad h1(V) h2(V) se cumple para todos los nodos del espacio de estados, que no son nodos meta, entonces A*-algoritmo que usa el componente heurístico h2(V) se llama el algoritmo más informado A*-algoritmo que usa el componente heurístico h1(V). En (Nilsson 1971) está demostrado que si esas funciones son estáticas (es decir, no se cambian durante la búsqueda), entonces el algoritmo más informado siempre abren menor número de nodos para encontrar la ruta con el costo mínimo. Eso significa que el algoritmo más informado realiza una búsqueda mejor dirigida, y por lo tanto más eficaz del nodo meta. De esa manera el concepto de ser más informado refleja uno de los aspectos del concepto de la fuerza heurística de una función de evaluación durante la búsqueda en un espacio de estados.

Resumiendo, es necesario buscar una función heurística h(V), que sea el límite inferior de h*(V) para garantizar la admisibilidad del algoritmo, y que sean lo más cerca posible a h*(V) para tener la efectividad de la búsqueda. Claro está que existen problemas donde no se puede encontrar en todos los casos una función de evaluación que garantizara tanto la efectividad como la admisibilidad de la búsqueda heurística. Entonces en muchos casos estamos obligados a utilizar las funciones heurísticas que reducen la búsqueda en muchos casos pero no garantizan que la ruta óptima será encontrada.

En el caso ideal cuando conocemos la evaluación h*(V) y se usa como la función h(V), el algoritmo A*-algoritmo encuentra la ruta óptima directamente, sin abrir los nodos no necesarios.

Una fuerte simplificación del algoritmo base de la búsqueda heurística con una función de evaluación general es el algoritmo de "subida a la colina" (Nilsson 1980, Winston 1984). Este algoritmo al abrir cada nodo reordena (según el valor de la función de evaluación) los nodos hijos generados y escoge para la apertura subsecuente un nodo hijo con el menor valor de evaluación; lo hace así, y no busca un nodo con el menor valor de evaluación entre todos los nodos no abiertos del árbol de la búsqueda, como lo hace el algoritmo base de búsqueda heurística. Es obvio que esta selección local es mucho más fácil que la selección global en todo el árbol de la búsqueda.

La idea principal de este método de búsqueda se basa en el método de "subida a la colina" conocido fuera del área de inteligencia artificial para encontrar un máximo (mínimo) de una función. Según este método para encontrar un máximo de la función, en cada paso se hace la jugada hacia mayor inclinación de la función (colina). Para cierto tipo de funciones —que tengan un máximo único y algunas otras propiedades de crecimiento— este uso de la información local, es decir, conocimiento de la dirección de mayor inclinación en cada punto, permite encontrar una solución global, es decir el máximo de la función.

En el algoritmo de "subida a la colina" en el espacio de estados, la función que determina la dirección es la función heurística de evaluación tomada con

el signo opuesto. La búsqueda continúa siempre del aquel nodo hijo que tenga menor valor de la función heurística; se deben evitarse las situaciones cuando hay varios nodos con la misma evaluación.

Es importante mencionar que el algoritmo de "subida a la colina" produce el mismo resultado que el algoritmo base de la búsqueda heurística en casos cuando la función de evaluación tiene ciertas propiedades, en particular tiene un solo máximo global. Este algoritmo no funciona si la función heurística tiene varios máximos locales. De esta manera la aplicación de este algoritmo está restringida, sin embargo, a veces, se puede resolver los problemas correspondientes construyendo una función heurística más sofisticada.

3. Búsqueda basada en reducción de problemas

3.1 Método de reducción de problemas

Para resolver los problemas del área de la inteligencia artificial, además del enfoque ya descrito basado en la representación del problema en un espacio de estados, existe otra posibilidad más compleja. Utilizando este nuevo enfoque se hace análisis del problema inicial para detectar un conjunto de subproblemas; al resolver todos los subproblemas se encontrará la solución del problema inicial. Cada subproblema en el caso general es más simple que el problema inicial y puede ser resuelta utilizando algún método, en particular, el método basado en un espacio de estados. Sin embargo, se puede continuar este proceso de búsqueda de subproblemas en cada subproblema de manera consecutiva hasta llegar a los ***problemas elementales*** para los cuales se conoce la solución. Esta metodología basada en la sustitución de un problema con un conjunto de subproblemas se llama ***reducción de problemas***.

Para dar ejemplo de este enfoque, vamos a considerar una variante de un conocido rompecabezas —Torre de Hanoi (o pirámide). Se utilizan tres postes, vamos a utilizar las letras A, B y C para marcarlos, y tres discos del diámetro diferente que se colocan sobre los postes utilizando los espacios en el centro de cada disco. Al empezar todos los discos se encuentran en el poste A, y los discos de menor diámetro se encuentran sobre los discos de mayor diámetro, véase la ilustración 9. Los discos se enumeran según se aumenta su diámetro. La tarea es mover los discos del poste A al poste C utilizando un conjunto de reglas (restricciones): Se puede mover solamente el disco que se encuentra arriba sobre todos los discos en un poste y ningún disco se puede colocar sobre un disco del menor diámetro. En la ilustración 9 se muestra el estado inicial y el estado meta del problema de Torre de Hanoi.

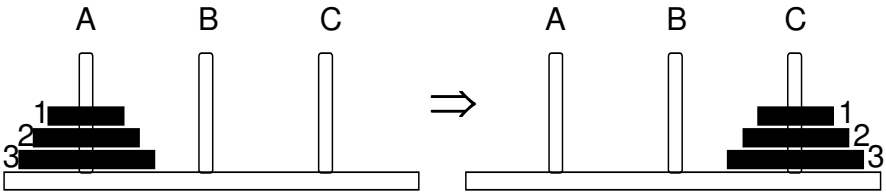


Ilustración 9. Problema de Torre de Hanoi.

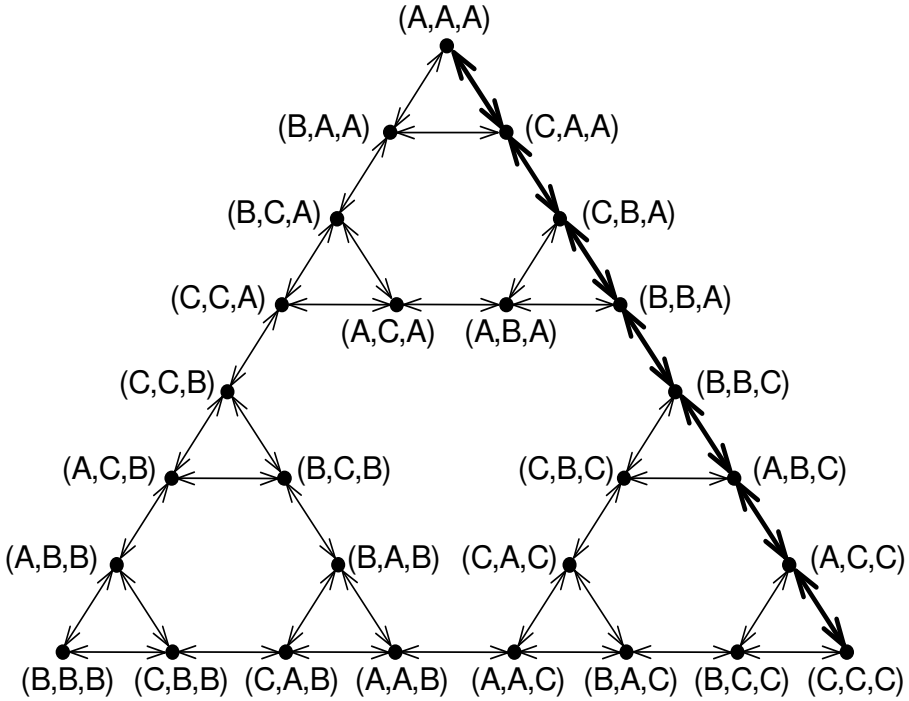


Ilustración 10. Espacio de estados del problema de Torre de Hanoi.

Es relativamente fácil formalizar este problema en un espacio de estados: un estado corresponde a una lista de tres elementos cada uno de los cuales representan la posición del disco correspondiente, es decir, el primer elemento corresponde al primer disco, segundo elemento al segundo, y tercer elemento corresponde al tercer disco. El estado inicial se describe como la lista (AAA), mientras que el estado meta es la lista (CCC). Se supone que si en el mismo poste se encuentran más de un disco, entonces cualquier disco de mayor diámetro se encuentra por debajo de cualquier disco de menor diámetro.

El espacio de estados completo del problema de Torre de Hanoi se representa con el grafo de la ilustración 10, el que contiene 27 nodos. La ruta en el grafo marcada con negrita representa la solución del problema que incluye siete jugadas de los discos.

Se puede encontrar esta solución realizando todas las posibles jugadas de los discos, sin embargo el método de reducción de problemas permite encontrar una solución más rápida. La idea clave de reducción consiste en que para mover todo la pirámide es necesario mover el disco 3, que es el disco más abajo; sin embargo eso es posible solamente si los discos de menor diámetro 1 y 2 se movieron al poste B, como en la ilustración 11(a).

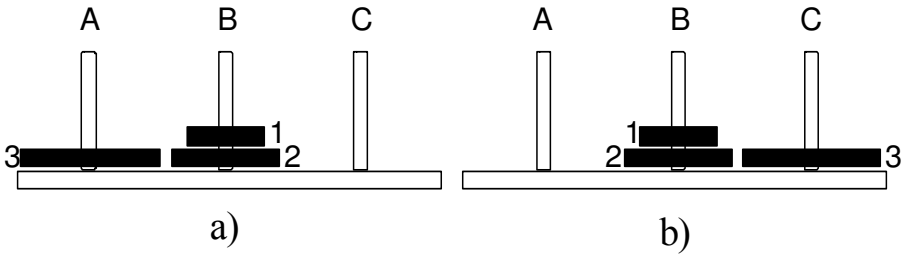


Ilustración 11. Problema de torres de Hanoi: dos estados cruciales

De esa manera, se puede identificar tres subproblemas que permiten solucionar el problema en total:

1. Mover los discos 1 y 2 del poste A al poste B.
2. Mover el disco 3 del poste A al poste C.
3. Mover los discos 1 y 2 del poste B al poste C.

En la ilustración 11(b) se muestra el estado inicial del subproblema 3.

Cada uno de los tres subproblemas es más simple que el problema en total. Realmente, en los subproblemas 1 y 3 solamente se mueven dos discos; y el subproblema 2 debe considerarse como elemental, ya que su solución es una sola jugada —cambiar el disco 3 al poste C. Se puede aplicar el mismo método de reducción de problemas a los subproblemas 1 y 3, y convertirlos en problemas elementales. Todo el proceso de reducción se puede representar esquemáticamente como un árbol, véase la ilustración 12. Los nodos del árbol correspondan a los problemas o subproblemas, mientras que los nodos relación el problema que se está reduciendo con sus subproblemas.

Es importante notar que la idea de reducción de un problema a un conjunto de subproblemas puede ser aplicada en el caso cuando la confirmación inicial del problema de Torre de Hanoi contiene no solamente tres, sino mayor número de discos. De esa manera, en el caso del enfoque basado en la reducción, también obtenemos un espacio, pero este espacio no contiene los estados, sino los problemas o subproblemas (sus descripciones). En este caso, el papel de los operadores que se utilizaban en un espacio de estados, juegan

los operadores que convierten problemas a subproblemas. Más exactamente, cada ***operador de reducción*** transforma descripción del problema en la descripción de un conjunto de subproblemas; y este conjunto es tal, que la solución de todos los subproblemas garantiza la solución del problema que se reduce.

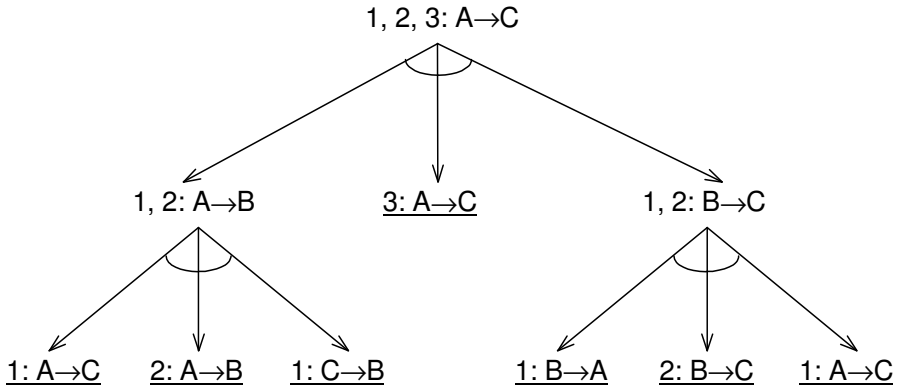


Ilustración 12. Reducción del problema de Torre de Hanoi.

Durante la resolución de un problema utilizando el método de reducción, igual que en caso de un espacio de estados, puede surgir la necesidad de búsquedas. Realmente, en cada etapa de reducción pueden existir varios operadores aplicables, es decir, varias maneras de convertir el problema a un conjunto de subproblemas; y, por lo tanto, varios conjuntos diferentes de subproblemas. Algunos caminos posiblemente no lleven a la solución del problema inicial, ya que pueden encontrarse los subproblemas sin solución; mientras que otros caminos permitan resolver el problema inicial. En el caso general, para la reducción del problema inicial es necesario intentar la aplicación de diferentes operadores. El proceso de reducción continúa hasta que el problema inicial no se reduce a un conjunto de los problemas elementales para los cuales se conoce la solución.

Igual que en el caso de la representación en un espacio de estados, la formalización de un problema con el enfoque basado en reducción incluye la definición de los siguientes componentes:

- Manera de descripción de los problemas y subproblemas, incluyendo la descripción del problema inicial.
- Conjunto de operadores y sus efectos a las descripciones de los problemas.
- Conjunto de los problemas elementales.

Esos componentes definen de manera implícita un ***espacio de problemas***, en el cual hay que realizar la búsqueda de la solución.

En cuanto a la manera de descripción de problemas y subproblemas, a menudo es cómodo utilizar el modelo de un espacio de estados, es decir definiendo un estado inicial y un conjunto de operadores, igual que un estado meta o sus propiedades. En este caso, se consideran como los problemas elementales, los problemas que pueden ser resueltos en un solo paso en el espacio de estados.

Si vamos a adoptar esta manera de descripción en el problema de Torre de Hanoi, entonces el problema elemental de jugadas del disco de mayor diámetro del poste A al poste C se puede escribir como (B,B,A)→(B,B,C); y el problema inicial se represente como (A,A,A)→(C,C,C). Adicionalmente, dado que el conjunto de operadores es el mismo para todos los problemas y subproblemas, entonces no es necesario repetirlo en la descripción de cada problema de manera explícita. Con esta manera de descripción de los problemas, los subproblemas se interpretan de manera natural como problemas de búsqueda de la ruta entre ciertos estados "marcados" (importantes) en el espacio de estados. Por ejemplo, los estados "marcados" son los estados (B,B,A) y (B,B,C) en el problema de Torre de Hanoi representados en la ilustración 11. Su propiedad importante es que la ruta de solución tiene que pasar por ellos.

3.2 Grafos Y/O. Grafo de solución

Para la representación del proceso de reducción de problemas y de los conjuntos alternativos de subproblemas se usan normalmente unas estructuras parecidas a los grafos. Los nodos de esas estructuras representan las descripciones de problemas y subproblemas, mientras que las aristas entre dos nodos corresponden al problema que se reduce y a una de los subproblemas construidos; las flechas de las aristas representan la dirección de reducción. Un ejemplo de esa estructura se encuentra en la ilustración 13(a): el problema G puede ser resuelta a través de la solución de los problemas D_1 y D_2; o E_1, E_2 y E_3; o del problema F. En este caso, las aristas que corresponden al mismo conjunto de subproblemas se unen por medio de un arco especial. Para hacer esta estructura más comprensible se introducen los nodos intermedios, y cada conjunto de subproblemas se agrupan debajo de su nodo padre. En este caso la estructura de la ilustración 13(a) se convierten en la estructura de la ilustración 13(b): para dos de tres conjuntos alternativos de subproblemas se introdujeron los nodos D y E.

Suponiendo que los nodos D y E corresponden a las descripciones de las rutas alternativas de la solución del problema inicial, se puede llamar el nodo G "O-nodo", dado que el problema G puede ser resuelta de manera D, o de manera E, o de manera F. De modo similar, se puede llamar los nodos D y E "Y-nodos" dado que para alcanzar su solución se requiere la solución de todos los subproblemas que dependen de ellos, lo que está marcado con un arco en

especial. Esta es la razón porque las estructuras parecidas a las estructuras presentadas en las ilustraciones 13(b) y 14 se llaman "***grafos Y/O***", o "***grafos de metas***".

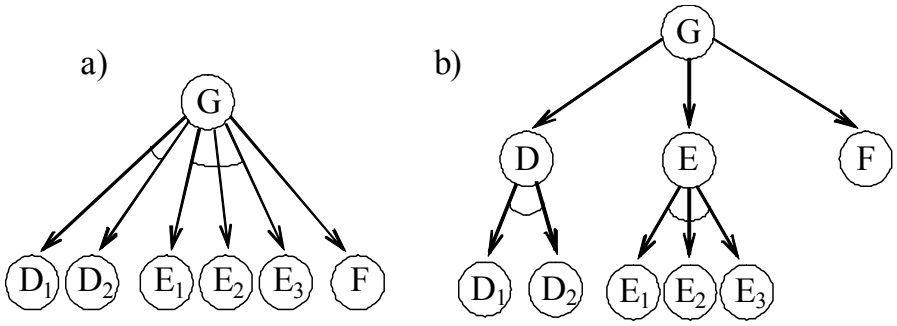


Ilustración 13. Representación esquemática de reducción de problemas.

Si algún nodo de este grafo tiene los nodos que le siguen —nodos hijo, entonces uno de los dos: todos esos nodos son Y-nodos, o todos esos nodos son O-nodos. Es necesario mencionar si algún nodo de un grafo Y/O genera exactamente un nodo hijo, se puede considerar este nodo tanto Y-nodo, como O-nodo. Un ejemplo de este tipo de nodo es el nodo F es la ilustración 13(b).

Utilizando el lenguaje de los grafos Y/O, la aplicación de varios operadores de reducción del problema significa que primero se construirá un Y-nodo intermedio, y después todos O-nodos que le siguen. La única excepción consiste en la situación cuando el conjunto de problemas contiene exactamente un nodo, vamos a considerarlo como O-nodo.

Un nodo de un grafo Y/O que corresponde a la descripción del problema inicial vamos a llamar el ***nodo inicial***. Los nodos que corresponden a las descripciones de los problemas elementales vamos a llamarlos ***nodos terminales***. En el grafo representado en la ilustración 14 el nodo P_0 es el nodo inicial, y los nodos P_1, P_4, P_5, P_7 y P_8 son los nodos terminales; ellos están marcados con negrita.

La búsqueda de solución del problema basada en caminos en los grafos, es aplicable también al enfoque que utiliza reducción de problemas. El propósito de la búsqueda en un grafo Y/O es demostrar que existe la solución del problema inicial, es decir, del nodo inicial. La existencia de la solución de todo el problema depende de si existe una solución para los otros nodos del grafo.

Vamos a formular la definición recursiva de existencia de la solución de un nodo en un grafo Y/O:

- Para los nodos terminales la solución existe, dado que aquellos corresponden a los problemas elementales.
- Para un O-nodo la solución existe, si y sólo si existe la solución para por lo menos un nodo hijo.

- Para un Y-nodo la solución existe, si y sólo si existe la solución para cada nodo hijo.

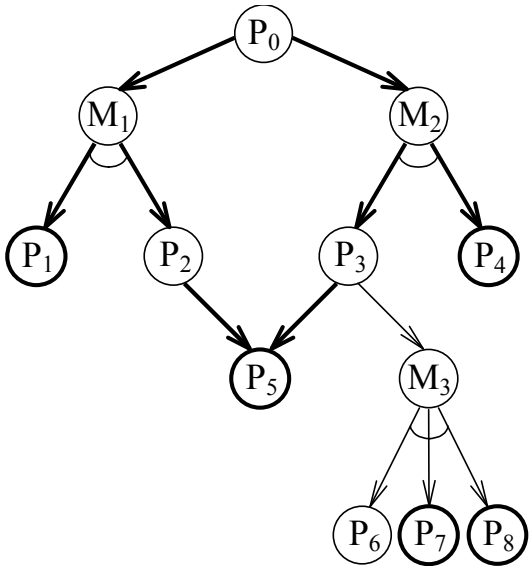


Ilustración 14. Ejemplo de grafo Y/O.

Si durante el proceso de la búsqueda pudimos mostrar que existe la solución para el nodo inicial, eso significa que encontramos la solución del problema que consiste en el grafo de solución. Un ***grafo de solución*** es un subgrafo de un grafo Y/O que consiste solamente de los nodos para los cuales existen sus soluciones y que demuestra que existe la solución para el nodo inicial.

Para el grafo Y/O presentado una ilustración 14, existe la solución para los nodos M_1, M_2, P_2 y P_3 (además de los nodos terminales). Este grafo contiene dos grafos de solución: el primero contiene los nodos P_0, M_1, P_1, P_2 y P_5; mientras que el segundo contiene P_0, M_2, P_3, P_4 y P_5. Para los nodos M_3 y P_6 no existen las soluciones (M_3 es así dado que no existe solución para P_6).

3.3 Ejemplo: el problema de integración simbólica

Como un ejemplo del método de reducción se considerará la solución del problema de la integración simbólica, es decir, encontrar una integral indefinida $\int F(x)dx$. Por lo general, este problema se resuelve mediante de transformaciones sucesivas de la integral a una expresión que contiene las integrales tabuladas conocidas. Para ello se usan algunas reglas de la integración, entre ellas: la regla para integración de la suma de funciones, la regla de integración por partes, la regla de sacar una constante afuera del

símbolo de la integración, y aplicación de las sustituciones algebraicas y trigonométricas y el uso de varias fórmulas algebraicas y trigonométricas.

Para formalizar este problema en el marco del enfoque basado en reducción de problemas, es necesario determinar la forma de descripción de problemas y subproblemas, los operadores de reducción y problemas elementales. Como una posible forma de descripción de los problemas se puede utilizar una cadena de caracteres que contiene el integrando y la variable de integración (si esta última no ha sido fijada de antemano). Los operadores de reducción se basarán, obviamente, sobre las reglas mencionadas de la integración. Por ejemplo, la regla de integración por partes $\int u dx = u \int dv - \int v du$ reduce el problema inicial (la integral indefinida del lado izquierdo de la formula) a los dos subproblemas de integración: las dos integrales correspondientes del lado derecho.

Es necesario tomar en cuenta que algunos operadores de reducción así obtenidos —como, por ejemplo, los operadores de integración por partes o de integración de la suma de funciones—, en realidad reducen el problema original y generan un Y-nodo en el grafo Y/O; mientras que las transformaciones y sustitución algebraicas y trigonométricas —como la división del numerador por el denominador, o complemento hasta el cuadrado completo—, sólo reemplazar una expresión por la otra, generando así O-nodos en el grafo Y/O.

Los problemas elementales de integración corresponden a las integrales tabuladas, por ejemplo:

$$\int \sin x \, dx = -\cos x + C$$

Es necesario tomar en cuenta que cada una de estas fórmulas tabuladas contiene una variable, de hecho es un patrón que representa un número infinito de problemas elementales.

La peculiaridad del problema de la integración es que, por ejemplo, al integrar por partes puede haber varias maneras de dividir el original integrando y, en consecuencia, varias maneras de aplicar esta regla de integración. Esto significa que en el caso general para una regla existen varias maneras para reducir el problema, es decir, varias maneras de utilizar el mismo operador de reducción.

Otra característica del problema, radica en el hecho de que en cada etapa de reducción se aplica por lo general un gran número de operadores (incluido varias aplicaciones del mismo operador), y, por lo tanto, el grafo Y/O del problema es demasiado grande, incluso para problemas simples de la integración. Entonces, para hacer una búsqueda eficaz en este grafo, es necesario de alguna manera limitar y/u ordenar el conjunto de los nodos generados durante la búsqueda. Por ejemplo, se puede ordenar a los operadores de reducción según su grado de utilidad, y asignar una mayor prioridad a los operadores que corresponden a la integración de la suma y de la integración por partes.

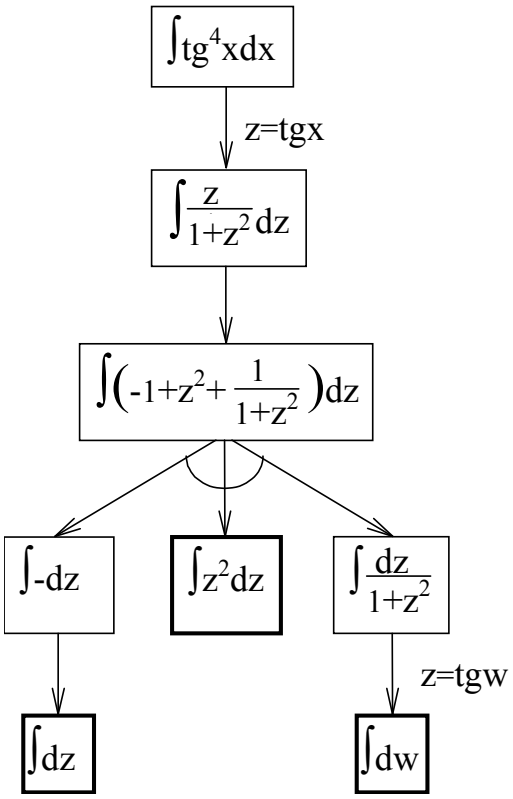


Ilustración 15. Grafo de solución para un problema de integración.

En la ilustración 15 se muestra el grafo de solución para un problema de integración. En los nodos del grafo se encuentran los problemas/subproblemas, los nodos terminales están marcados con negrita. La solución puede ser ensamblada a partir de información contenida en el grafo y consiste en los siguientes pasos:

1. Uso de la sustitución z = tg (x);
2. Transformación equivalente del integrando;
3. Aplicación de la regla de integración de la suma de funciones; uno de los tres subproblemas resultantes es elemental, y los otros dos se resuelven en un solo paso. El primer problema se resuelve al sacar la constante afuera del símbolo de la integración y el segundo con la sustitución z = tg (w).

El enfoque basado en reducción de problemas, es aplicable en la práctica y tiene ventajas sobre el basado en la representación de un espacio de estados, cuando los subproblemas obtenidos después de la reducción pueden ser

resueltos de forma independiente el uno del otro, como en el ejemplo de integración. Sin embargo, esta condición de independencia mutua de los subproblemas resultantes no tiene que ser muy rígida. El método de reducción también es aplicable si las soluciones de subproblemas son dependientes el uno del otro, pero existe un procedimiento para su reducción, en el cual las soluciones de subproblemas anteriores no se destruyen durante la solución de otros, que se resuelven más tarde; como en el ejemplo de Torre de Hanoi, donde el orden de los subproblemas es importante, pero no existe dependencia entre ellos.

3.4 Específica de búsqueda en grafos Y/O

Un grafo Y/O, llamado también el ***grafo de metas*** (problemas), se define de forma implícita durante la formalización del problema utilizando el método de reducción, por medio de la descripción del problema inicial, definición de los operadores de reducción del problema a los subproblemas y detección del conjunto de los problemas elementales.

Obviamente, si no existe un Y-nodo en este grafo, obtenemos un grafo de búsqueda en espacio de estados habitual en el cual se puede utilizar los algoritmos de búsqueda en espacio de estados presentados anteriormente. Sin embargo, debido a la presencia de Y-nodos en el caso general en este grafo es necesario modificar los algoritmos, lo que los hace sustancialmente más complejos.

Las causa de mayor complejidad de los algoritmos de búsqueda en grafos Y/O está relacionada con el hecho de que la verificación de condiciones de terminación de la búsqueda se vuelve más compleja, es decir, la verificación de existencia de solución para el problema original. Después de la construcción de cada nuevo nodo se debe determinar de alguna manera si ya existe la solución para el nodo inicial. Si la solución ya existe, debemos buscar el grafo de solución, para saber cuál es esta solución. Si todavía no sabemos si existe la solución para el nodo inicial, es necesario determinar si tiene sentido continuar la búsqueda, ya que puede estar demostrado que no existe la solución para algunos subproblemas que son indispensables para la solución del problema original, lo que significa que el problema original no tiene solución. *Ausencia de solución* de un nodo (y del problema correspondiente) en un árbol de búsqueda se puede definir de manera recursiva del mismo modo que se definió anteriormente la existencia de solución:

- el nodo que no es terminal y no tiene nodos hijos, no tiene solución;
- O-nodo no tiene solución si y sólo si todos sus nodos hijos no tienen solución;
- Y-nodo no tiene solución si y sólo si al menos uno de sus nodos hijos no tiene solución.

Igual que en el caso de la búsqueda en un espacio de estados, todos los algoritmos de búsqueda en un grafo Y/O se puede describir utilizando la operación de apertura del nodo corriente, lo que significa la aplicación a este (más precisamente, el estado correspondiente) de todos los posibles operadores de reducción, lo que genera los nodos hijos. El proceso de la búsqueda, representado como un proceso de apertura gradual de los nodos de un grafo Y/O, corresponde a la transformación de una manera explícita de una parte de este grafo Y/O, definido de forma implícita durante la formalización del problema. Esta transformación implica, en términos generales, existencia de apuntadores de los nodos hijos hacia sus nodos padres; estos apuntadores son necesarios para determinar los nodos para los cuales existe o no existe la solución y para la determinación del grafo de solución (que contiene sólo los nodos que tienen solución).

La estructura de nodos y apuntadores que se forma durante la búsqueda, se conoce como un ***grafo de búsqueda***; al terminar la búsqueda este grafo contiene el grafo de solución. En general, el grafo de búsqueda contiene nodos tanto con solución como sin solución, y los nodos sobre los cuales se desconoce si tienen solución o no.

Así, en los algoritmos de búsqueda para grafos Y/O existe la necesidad de una organización más compleja de las listas de los nodos abiertos y cerrados (los nodos pueden ser de dos tipos, y para el nodo de cada tipo puede ser determinado la existencia o no de la solución). Además, durante la búsqueda es adecuado de alguna manera mantener subgrafos que son candidatos para el grafo solución; al terminar la búsqueda uno de ellos será la solución del problema.

Los principales tipos de búsqueda en grafos Y/O se diferencian según el orden de apertura de nodos. Igual que en el espacio de estados se utilizan:

- Búsqueda en amplitud —los nodos se abren en el orden en que fueron construidos;
- Búsqueda en profundidad —primero se abren los nodos construidos en el último paso; generalmente la profundidad de la búsqueda se limita a alguna profundidad específica;
- "Subida a la colina" y búsqueda heurística (selección del nodo para su apertura, se basa en las evaluaciones heurísticas de los nodos).

Algoritmos de búsqueda en amplitud en grafos Y/O que son árboles, se formulan de manera más simple: la organización de iteraciones es mucho más simple, ya que cualquier problema (meta) puede presentarse en la apertura de los nodos exactamente una vez. La profundidad de los nodos en el árbol Y/O que se construye por los algoritmos de búsqueda se puede definir de la misma manera que en el caso de los árboles de búsqueda en un espacio de estados.

Algoritmo de búsqueda en amplitud en un árbol Y/O necesariamente encuentra el árbol de solución, el nodo terminal del cual tiene la profundidad mínima (a menos que, por supuesto, la solución existe).

Algoritmo de búsqueda en profundidad sin límite, puede utilizarse sin el peligro de entrar en un ciclo infinito, sólo para los árboles finitos. Una búsqueda en profundidad más segura, es búsqueda en profundidad con un límite establecido, en el cual los nodos no terminales que tienen la profundidad igual al límite, se consideran los nodos sin solución. Al igual que con la búsqueda en un espacio de estados, la limitación de la profundidad puede impedir encontrar las soluciones, pero un árbol de solución que se encuentra dentro del límite establecido será encontrado necesariamente.

Una búsqueda ciega en un árbol Y/O, igual como en un espacio de estados, es ineficiente, por lo que es mejor utilizar el algoritmo de búsqueda heurística. En este algoritmo se elige el nodo más prometedor en el árbol de búsqueda para su apertura, más precisamente, un nodo más prometedor en el subárbol, que es el candidato para ser el árbol de solución.

Para la búsqueda heurística en un grafo Y/O se propuso y se estudió un algoritmo, que es análogo del A*-algoritmo en el espacio del estados; mismo que se nombró AO*-algoritmo (Nilsson 1971, Nilsson 1984). Si se establece un costo de las aristas en un árbol Y/O, y se aplica la función heurística de evaluación de una forma especial, entonces este algoritmo puede encontrar el árbol de solución con un costo mínimo, en el número finito de pasos. En el caso más simple se puede considerar como el costo de un árbol, el valor total que se obtiene al sumar el costo de sus aristas. En otras palabras, para AO*-algoritmo igualmente se cumple el teorema de admisibilidad que garantiza la posibilidad de encontrar la solución con un costo mínimo (el *árbol óptimo*), siempre que exista un árbol de solución.

3.5 Búsqueda en profundidad en los árboles Y/O

Para evitar numerosos detalles de la búsqueda en grafos Y/O generalizados, consideramos solamente la búsqueda en los grafos que son árboles. La búsqueda en profundidad para la apertura de nodos y la formación de un árbol Y/O de solución se organiza más fácil usando tres procedimientos mutuamente recursivos, los podemos llamar, respectivamente, funciones AND/OR, AND y OR; y vamos a describir sus pasos básicos [Winston 1984].

Función AND/OR, al recibir en la entrada un nodo (problema) Goal, o bien produce un fallo, si este nodo no tiene solución, o bien devuelve un árbol de solución, mostrando que la solución existe y cuál es.

Paso 1. Comprobar si el nodo Goal es el nodo meta. En caso afirmativo, terminar el procedimiento, retornando este mismo nodo como resultado, de lo contrario proceder al siguiente paso.

Paso 2. Comprobar si se puede reducir (abrir) el nodo Goal. En caso afirmativo, continuar con el paso 3, de lo contrario salir del procedimiento con un mensaje de fallo.

Paso 3. Abrir el nodo Goal, formando una lista de sus nodos hijos G_List, continuar con el paso 4.

Paso 4. Si el nodo Goal es un Y-nodo, aplicar a él y la lista G_List la función AND, finalizar el procedimiento actual con el resultado recibido de la función AND.

Paso 5. Si el nodo Goal es un O-nodo, aplicar a él y la lista G_List la función OR, finalizar el procedimiento actual con el resultado recibido de la función OR.

Fin de la función AND/OR.

Las funciones AND y OR se parecen entre sí. Cada una de ellas recibe en la entrada el nodo Goal y la lista de sus nodos hijos G_List; el resultado de las funciones o bien es un árbol de solución con la raíz en el nodo Goal, o bien el mensaje de fallo (inexistencia de solución para este nodo).

Pasos principales de la función AND:

Paso 1. Establecer como el grafo de solución del problema Goal al mismo nodo Goal y continuar al paso siguiente.

Paso 2. Si la lista G_List no está vacía, entonces continuar al paso 3, de lo contrario terminar el procedimiento, retornando como resultado el grafo de solución construido para el nodo (meta) Goal.

Paso 3. Seleccionar de la lista G_List un nodo siguiente (llamémoslo Current) y verificar si existe alguna solución para Current llamando a la función AND/OR.

Paso 4. Si no existe la solución para el nodo Current, terminar la función actual con un mensaje de fallo, de lo contrario continuar al paso siguiente.

Paso 5. Conectar el grafo de solución obtenido por la función AND/OR para el nodo Current al grafo de solución del nodo Goal y continuar con el paso 2.

Fin de la función AND.

Pasos principales de la función OR:

Paso 1. Establecer como el grafo de solución del problema (nodo) Goal al mismo nodo Goal y continuar al paso siguiente.

Paso 2. Si la lista G_List no está vacía, entonces continuar al paso 3, de lo contrario terminar el procedimiento con un mensaje de fallo.

Paso 3. Seleccionar de la lista G_List un nodo siguiente (llamémoslo Current) y verificar si existe alguna solución para Current llamando a la función AND/OR.

Paso 4. Si no existe la solución para el nodo Current, continuar con el paso 2, de lo contrario continuar al paso siguiente.

Paso 5. Conectar el grafo de solución obtenido por la función AND/OR para el nodo Current al grafo de solución del nodo Goal y terminar el procesamiento retornado el grafo de solución obtenido como su resultado.

Fin de la función OR.

En caso de una selección arbitraria del nodo en el paso 3 de la función OR la búsqueda se convierte en una simple búsqueda en profundidad. Sin

embargo, este paso puede convertirse en un paso más complejo utilizando alguna función heurística, que evalúa que tan prospectivos son los nodos. Si la lista G_List está ordenada por ascendencia de la función de evaluación heurística, y en el paso 3 de la función OR se selecciona el nodo con la evaluación más baja, se obtiene un algoritmo que es análogo del algoritmo "Subida a la colina" en los espacios de estados. Es necesario tomar en cuenta que para una búsqueda heurística el esquema presentado de recorrido de un árbol Y/O con las tres funciones recursivas no es adecuado dado la necesidad de acceso a los nodos terminales del subárbol que es el candidato para ser el árbol de solución; se requiere una organización mucho más compleja de recorrido de los nodos.

3.6 Diferencias y operadores clave

Al final es importante discutir algunas ideas generales, que pueden ser fundamentales en el proceso de reducción de problemas. Las ideas más importantes son la idea de definición de los llamados *operadores clave* y la idea de *diferencias* de estados. Suponemos que los problemas y subproblemas se resuelven en un espacio de estados y se representan como (S_I, O, S_G), donde SI es un estado inicial y SG es un estado meta; O es el conjunto de operadores de transformación entre estados, que puede ser omitido si es constante (no sufre cambios).

A menudo, durante la búsqueda en el espacio de estados de algún problema se puede encontrar un operador que necesariamente forma parte de la solución del problema (de hecho, la aplicación de este operador es un paso necesario para resolver este problema). En el grafo (espacio de estados) a este operador corresponde normalmente una arista, que conecta las dos partes del grafo que son prácticamente independientes. Este operador se llama un ***operador clave*** en el espacio de estados. Por ejemplo, el problema de Torre de Hanoi, el operador clave era el operador de jugada al poste deseado del disco más grande (disco 3), véase la ilustración 11.

El operador clave puede ser usado para la siguiente reducción del problema original a los subproblemas. Denominamos

Op: el operador clave encontrado en el espacio de estado del problema (Op$\in$ O);

S_{Op}: un conjunto de estados donde se puede aplicar el operador Op;

Op(s): el estado obtenido después de la aplicación del operador clave Op al estado s ($s \in S_{Op}$);

Entonces, el problema original se reduce a los tres subproblemas:

- El primer problema (S_I, S_{Op}) es encontrar un camino desde el estado inicial del problema original a uno de los estados donde se aplica el operador clave Op;

- El segundo problema es elemental y consiste en la aplicación de este operador clave;
- El tercer problema (Op(s), S_G) es encontrar el camino desde el estado obtenido después de la aplicación del operador clave, hasta el estado meta del problema original.

El orden de solución de estos problemas es importante: el tercer problema no puede ser resuelto antes del primero y del segundo, así como el segundo problema no se resuelve antes del primero. Para resolver a los problemas primero y tercero se puede aplicar el método de reducción de nuevo; o bien su solución se puede encontrar realizando directamente la búsqueda en el espacio de estados.

Para la mayoría de los problemas no se puede identificar de forma única el operador clave, sino se puede encontrar un conjunto de *candidatos para ser operadores clave*, es decir, operadores que con una mayor probabilidad podrían ser operadores clave. Así, en general, se requiere el proceso iterativo de aplicación de los operadores clave candidatos; cada uno de ellos genera su conjunto de problemas resultantes (esta aplicación significa exactamente la búsqueda en el grafo Y/O del problema).

Con este método de reducción de problemas la pregunta más importante es, cómo encontrar los candidatos a los operadores clave. Uno de los métodos propuestos y probados por primera vez en uno de los sistemas de inteligencia artificial famoso GPS (Rich and Knigh, 1994), es identificar las diferencias de los estados inicial y meta del problema. Es más fácil formalizar la diferencia comparando los estados inicial y meta. Si el estado inicial es igual al estado meta, entonces no hay diferencias entre ellos, y el problema ya está resuelto.

Por ejemplo, en el problema anterior del mono y del plátano la diferencia se encentra en la desigualdad de los elementos en las listas de descripción de dos estados. Luego, al comparar los estados (P_{mono}, $P_{plátano}$, P_{caja}, 0) y (P_{mono}, $P_{plátano}$, P_{caja}, 1) se revelan tres diferencias, respectivamente, en el primero, tercero y cuarto elementos de estas listas.

Cada diferencia en el sistema GPS se conectaba con uno o varios operadores destinados a eliminar o reducir estas diferencias. Estos operadores eran precisamente los candidatos para ser operadores clave. En cada etapa el sistema determina la diferencia entre el estado actual (objeto) del problema y el estado meta (objetivo) y después selecciona y trata de aplicar operadores para reducir la diferencia. Los operadores también pueden incluir las *condiciones previas* (las condiciones de su aplicación), cuyo cumplimento es necesario para su aplicación exitosa; en este caso GPS reduce el problema original a otro problema de conseguir las condiciones deseadas.

En el problema del mono es natural asociar las diferencias y los operadores candidatos para ser operadores clave de la siguiente manera:

- Las diferencias en el primer elemento de las listas que describen el estado (la posición del mono en el piso): los operadores Moverse y Mover_caja.

- Las diferencias en el segundo elemento (la posición del mono en el plano vertical): el operador Subir.
- Las diferencias en el tercer elemento (la posición de la caja): el operador Mover_caja.
- La diferencia en el cuarto elemento (tener o no el plátano): el operador Agarrar.

Es evidente que para la implementación de esas ideas en un programa es necesario, en primer lugar, un procedimiento especial para la comparación de las descripciones de los estados y el cálculo de las diferencias. En segundo lugar, se necesita un medio para conectar los operadores clave con diferencias relevantes —en el caso de GPS se utiliza una tabla de correspondencia entre operadores y diferencia. Esta tabla (o procedimiento) también debe establecer la prioridad de diferencias y operadores, es decir, el orden de su aplicación en los casos en que se encontró varias diferencias y se puede aplicar varios operadores clave. Las diferencias deben ser ordenadas por su relevancia para la solución del problema original. El sistema GPS empieza con un intento de resolver las diferencias más difíciles, pasando después a las diferencias más fáciles.

En el problema del mono y el plátano la prioridad (valor) de las diferencias se puede establecer, por ejemplo, de la siguiente manera: la diferencia en el cuarto elemento, después, en el segundo, después, en el tercero y en el primero elemento de la descripción del problema. La misma prioridad pueden tener los operadores que reducen estas diferencias.

Hacemos hincapié en que la información presentada en los procedimientos y tablas es específica y depende del problema específico, es decir, es la información heurística orientada a los problemas. Es precisamente una de las debilidades del enfoque empleado en el sistema GPS: los procedimientos de cálculo de diferencias e implementación de los operadores que los reducen deben ser analizados e implementados para cada problema específico (o para un área muy restringida que incluye algunos tipos de problemas), de lo contrario la eficacia de la solución de problemas se reduce.

Al mismo tiempo, el principal mecanismo para la solución de problemas en el sistema GPS no fue orientado a problemas: es una realización de un método universal heurístico para resolver problemas, a menudo usados por los humanos, y conocido como el análisis de medios y objetivos (*means-ends analysis*) (Winston 1984). La idea clave de esta heurística es la siguiente:

- Encontrar las diferencias entre lo que está dado en el problema y lo que es necesario obtener;
- Eliminar gradualmente las diferencia, utilizando los medios (las operaciones) adecuados.

Trabajando con esta heurística, el GPS aplica varios planes de reducción del problema, y de forma recursiva va formando el árbol de solución de metas (problemas y subproblemas) mediante la identificación de las diferencias entre los objetos y aplicación de los operadores de reducción de estas diferencias.

4. Búsqueda en árboles de juegos

Consideramos un conjunto de ***juegos de dos personas con la información completa***. En estos juegos participan dos jugadores que turnan sus jugadas. En cualquier momento, cada jugador sabe todo lo que ha sucedido en el juego hasta este momento y qué jugadas se puede realizar en el momento dado. El juego termina con la victoria de un jugador (y la derrota del otro), o con un empate.

Así, a esta clase de juegos no pertenecen los juegos cuyo resultado depende, aunque parcialmente del azar: juego de las cartas, dados, "batalla naval" etc. Sin embargo, esta clase es bastante amplia e incluye juegos como el ajedrez, damas, reversi, kalah, gato (tres en línea) y otros juegos.

4.1 Árboles de juego. Búsqueda de estrategia ganadora

Para formalizar y estudiar las estrategias de juego en la clase de juegos con información completa, se puede utilizar el enfoque basado en reducción de problemas. Recordemos que en este caso se debe definir los siguientes componentes: una forma de descripciones de problemas y subproblemas, los operadores, que reducen un problema a subproblemas; problemas elementales, así como una descripción del problema inicial.

El problema más interesante es encontrar una estrategia ganadora para uno de los jugadores, a partir de algunas posiciones específicas del juego (no necesariamente a partir de la posición primaria). Utilizando el enfoque basado en reducción de problemas, una estrategia ganadora se busca durante el proceso de la demostración, que se puede ganar el juego. Del mismo modo, la estrategia de búsqueda de empate en alguna posición particular del juego, se realiza demostrando que el juego puede ser llevado a un empate.

Es evidente que la descripción del problema debe contener una descripción de la configuración del juego para la cual se busca la estrategia. Por ejemplo, en un juego de damas la descripción de posición incluye ubicación sobre el tablero de todas las piezas, incluyendo las reinas. Por lo general, la descripción de la configuración también incluye una indicación de qué lado es la jugada siguiente.

Los nombres de los jugadores serán MAS y MENOS. Usamos la siguiente notación:

X^S y Y^S son configuraciones del juego, y S toma los valores "+" o "-" indicando quien debe realizar la siguiente jugada (es decir, en la configuración de X^+ la próxima jugada es del jugador MAS), en la configuración de X^- la próxima jugada es del jugador MENOS.

$W(X^S)$ es el problema de demostración que un jugador MAS puede ganar empezando en la configuración X^S;

$V(X^S)$ es el problema de demostración que un jugador MENOS puede ganar empezando en la configuración X^S.

En primer lugar, consideramos el problema de juego $W(X^S)$. Se puede determinar los operadores de reducción de este problema a los subproblemas usando como base las jugadas permitidas en el juego:

Si en una determinada configuración X^+, el jugador MAS realiza una jugada de N jugadas permitidos, los que generan las configuraciones X_1^-, X_2^-,..., X_n^-, entonces para la solución del problema $W(X^+)$ es necesario resolver al menos un subproblema $W(X_i^-)$. Realmente, ya que el jugador MAS selecciona su jugada, entonces él va a ganar el partido si al menos una jugada lleva a la victoria, véase la ilustración 16(a).

Si en una determinada configuración Y^-, el jugador MENOS realiza una jugada de K jugadas permitidos, los que generan las configuraciones Y_1^+, Y_2^+,..., Y_k^+, entonces para la solución del problema $W(Y^-)$ es necesario resolver todos los subproblemas $W(Y_i^+)$. Realmente, ya que el jugador MENOS selecciona su jugada, entonces el jugador MAS va a ganar el partido si después de cada jugada de su contrincante él tiene garantizada la victoria, véase la ilustración 16(b).

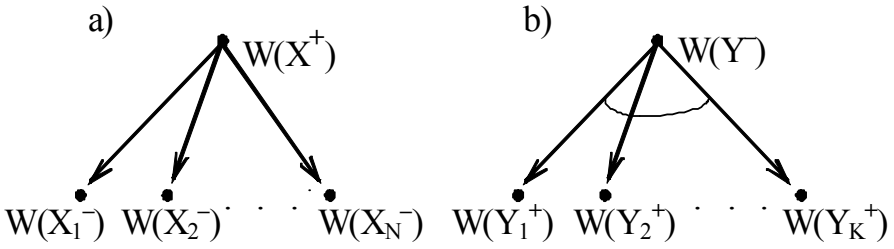


Ilustración 16. Reducción de problemas de juego.

La aplicación sucesiva de este esquema de reducción a la configuración inicial del juego genera un árbol Y/O (un grafo Y/O), que se llama un ***árbol (grafo) de juego***. Las aristas del árbol de juego corresponden a jugadas, los nodos representan a las configuraciones del juego, donde los nodos terminales (hojas del árbol) son posiciones en las que el juego termina con victoria, derrota o empate. Algunas hojas son posiciones que corresponden a problemas elementales donde el jugador MAS gana. Tomamos en cuenta que para las configuraciones, donde el turno es del jugador MAS, en el árbol de juego existe el O-nodo, y para las posiciones en las que el turno es del MENOS, la hoja es Y-nodo.

El objetivo de construcción de un árbol de juego es obtener un subárbol de solución del problema $W(X^S)$, que muestra cómo el jugador MAS puede ganar el juego desde la posición X^S independientemente de las respuestas de su contrincante. Para eso se puede aplicar diferentes algoritmos de búsqueda en

grafos Y/O. El árbol de solución termina en las posiciones donde el jugador MAS gana e incluye la estrategia completa para alcanzar la victoria: para cada posible continuación del juego elegido por el oponente, en el árbol existe una jugada de respuesta que permite ganar.

Para el problema $V(X^S)$ la estrategia de reducción de los problemas de juego a los subproblemas es similar: a las jugadas del jugador MAS corresponden Y-nodos, a las jugadas de MENOS corresponden O-nodos, y los nodos terminales corresponderán a las configuraciones que contengan la manera de ganar para el jugador MENOS.

Por supuesto, tal reducción de problemas también es aplicable en el caso cuando se necesita probar la existencia de una estrategia de empate en el juego. Obvio que las definiciones de los problemas elementales deben cambiarse como corresponde.

Consideremos como otro ejemplo un juego simple, conocido como "el último pierde". Dos jugadores se turnan en tomar una o dos monedas de un montón, que originalmente contiene siete monedas. El jugador que recoge la última moneda pierde.

La ilustración 17 muestra el grafo del juego completo para el problema $V(7^+)$, los arcos en negrita están marcando el grafo Y/O de solución, que demuestra que el segundo jugador (es decir, el jugador MENOS que inicia como segundo), siempre gana. La configuración del juego se representa como el número restante de las monedas, y la indicación de quien es el turno. Los nodos terminales que corresponden al problema elemental $V(1^+)$, es decir, al hecho que el jugador MENOS gana, están marcados.

Representada en un grafo de solución la estrategia ganadora puede ser formulada de la siguiente manera: si en su jugada el jugador MAS toma una moneda, entonces en la próxima jugada el MENOS debe tomar dos, y si el MAS toma dos monedas, el MENOS debe agarrar una. Tomamos en cuenta que para un problema similar $W(7^+)$ no se puede construir un grafo de solución (el nodo inicial no tiene solución), así que el jugador MAS no tiene una estrategia ganadora en este juego.

En la mayoría de los juegos que tienen mayor interés, como las damas y el ajedrez, no es posible construir un árbol de solución completo o encontrar una estrategia de juego completa. Por ejemplo, para las damas el número de nodos en el árbol del juego completo se estima alrededor de 10^{40}, y prácticamente es imposible recorrer un árbol así. En su caso, los algoritmos de búsqueda que utilizan heurísticas, no reducen tanto la parte del árbol que se está considerando para dar un resultado importante, disminuyendo el tiempo de búsqueda.

Sin embargo, por los juegos incompletos de las damas y del ajedrez (por ejemplo, de finales), igual que para todos los juegos simples, como el gato en un tablero no muy grande, se puede aplicar con éxito a los algoritmos de búsqueda en grafos Y/O que permiten detectar estrategias de juego ganadoras o de empate.

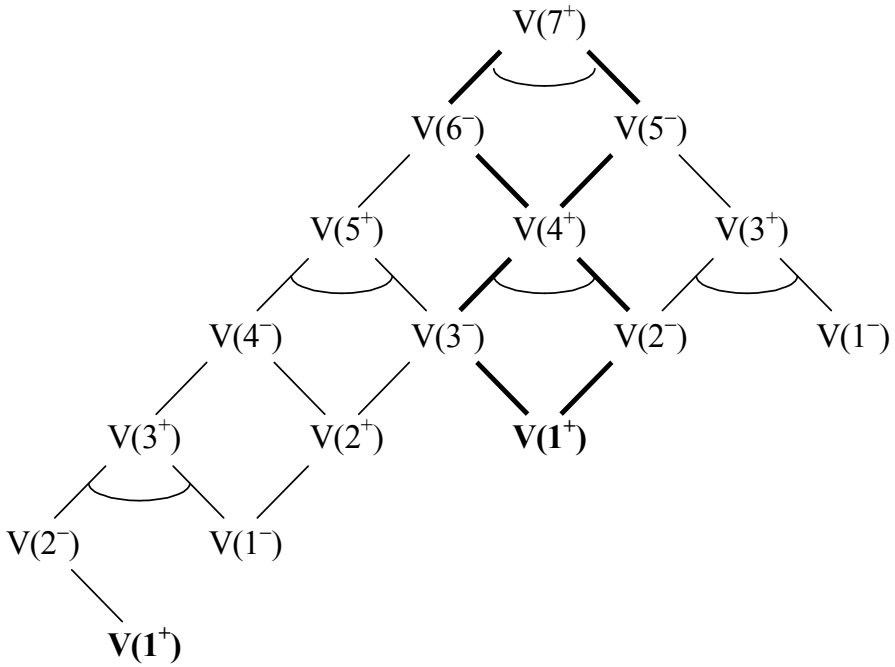


Ilustración 17. Grafo del juego "el último pierde".

Consideremos, por ejemplo, el juego del gato en el tablero de 3x3. Jugador MAS empieza y pone "X", jugador MENOS pone "0". El juego termina cuando se construye una fila, o una columna, o diagonal de las "X" (en este caso gana MAS) o "0" (gana MENOS). Estimamos el tamaño del árbol de juego completo: el nodo inicial tiene 9 nodos hijo, cada uno de los cuales a su vez cuenta con 8 nodos hijo, cada nodo de la profundidad dos cuenta con 7 hijos, etc. Así, el número de nodos finales en el árbol de juego es 9! = 362,880, pero muchos caminos en este árbol se cortan antes en los nodos terminales. Entonces, en este juego es posible recorrer todo el árbol y encontrar una estrategia ganadora. Sin embargo, la situación va a cambiar sustancialmente con un aumento significativo del tamaño del tablero, o en el caso del tablero de juego sin límites.

En estos casos, como en todos los juegos complejos, en lugar de la tarea poco realista de la búsqueda de una estrategia de juego completo, se resuelve, por lo general, una tarea más simple: para una posición determinada en el juego encontrar una *primera jugada aceptable*.

4.2 Procedimiento minimax

Con el fin de encontrar una primera jugada aceptable normalmente se hace un recorrido de una parte del árbol de juego construido a partir de una configuración dada. Se utiliza uno de los algoritmos de búsqueda (en profundidad, en amplitud, o heurístico) y alguna restricción artificial para que se finalice la búsqueda en el árbol de juego: por ejemplo, se limita el tiempo o la profundidad de la búsqueda.

En el árbol parcial de juego construido de esta manera se evalúan sus nodos terminales (hojas), y según esas estimaciones se determina la mejor jugada en una configuración determinada del juego. Por otra parte, para la evaluación de los nodos terminales del árbol obtenido se utiliza una función llamada ***función de evaluación estática***, y para la evaluación de los nodos restantes —el nodo raíz (inicial) y los nodos intermedios entre la raíz y el nodo final— se utiliza lo que se llama ***un principio minimax***.

La función de evaluación estática cuando se aplica a algún nodo que corresponde a una configuración del juego, regresa un valor numérico que toma en cuenta las diversas ventajas de esta posición de juego. Por ejemplo, para las damas pueden ser considerados tales elementos (estáticos) de una configuración de juego, como que tan lejos se promovieron las piezas, su movilidad, el número de los reinas, su control del centro, entre otras. De hecho, la función estática realiza una evaluación heurística de las posibilidades de ganar de uno de los jugadores. Consideremos el problema de ganar del jugador MAS y en consecuencia el problema de búsqueda de su buena primera jugada en una configuración dada.

En este caso acordemos que si la función de evaluación estática tiene mayor valor, mayores son las ventajas del jugador MAS (sobre el jugador MENOS) en la posición dada. Muy a menudo, la función de evaluación se selecciona de la manera siguiente:

- Función de evaluación estática es positiva en las configuraciones de juego, donde el jugador MAS tiene las ventajas;
- Función de evaluación estática es negativa en configuraciones donde el jugador MENOS tiene las ventajas;
- Función de evaluación estática es cercana a cero en las posiciones que no da ventajas a ninguno de los jugadores.

Por ejemplo, para las damas la función estática muy simple puede tomar en cuenta la ventaja en el número de piezas (y reinas) del jugador MAS. Para el juego de "gato" (tres en línea) en un tablero cuadrado fijo es posible la función estática siguiente:

$$E(P)= \begin{cases} +\infty & \text{si P es una posición donde el jugador MAS gana,} \\ -\infty & \text{si P es una posición donde el jugador MENOS gana,} \\ (N_L^+ + N_C^+ + N_D^+) - (N_L^- + N_C^- + N_D^-) & \text{en otros casos.} \end{cases}$$

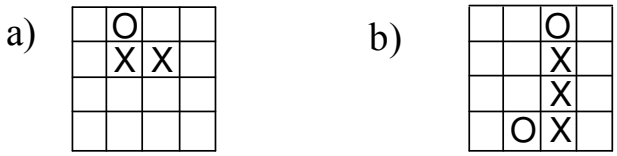
donde $+\infty$ es un número positivo muy grande,

$-\infty$ es un número negativo muy pequeño,

N_L^+ , N_C^+ , N_D^+ son cantidades de filas, columnas y diagonales "abiertas" para el jugador MAS, es decir, donde él puede poner tres "X" en línea,

N_L^- , N_C^- , N_D^- son cantidades de filas, columnas y diagonales "abiertas" para el jugador MENOS.

La ilustración 18 muestra dos posiciones de juego (en un cuadrado 4x4) y los correspondientes valores de la función de evaluación estática.



E (P) = 8 - 5 = 3 E (P) = 6 - 4 = 2

Ilustración 18. Ejemplo de evaluación estática en el juego de gato.

Hacemos hincapié que la función de evaluación estática se aplica solamente para los nodos terminales del árbol de juego mientras que para los nodos intermedios y el nodo inicial se utiliza el principio minimax basado en la siguiente idea sencilla. Si el jugador MAS tendría que elegir entre varias jugadas posibles, entonces él habría elegido la jugada más fuerte, es decir, la jugada que conduce a una posición con la estimación más alta. Del mismo modo, si la jugada tuviera que elegir el jugador MENOS, él habría elegido la jugada que conduce a la posición con una evaluación mínima.

Ahora podemos formular el ***principio minimax***:

O-nodo en un árbol de juego tiene la evaluación igual a la evaluación máxima de sus nodos hijos;

Y-nodo en un árbol de juego tiene la evaluación igual a la evaluación mínima de sus nodos hijos.

El principio minimax es la base del procedimiento minimax diseñado para identificar las mejores jugadas (más precisamente, relativamente buenos) de un jugador en la configuración dada del juego, para una profundidad de búsqueda N en el árbol de juego. Se supone que el jugador quien juega primero es MAS; es decir, el nodo inicial es un O-nodo. Los principales pasos de este procedimiento son los siguientes:

1. El árbol de juego se construye (se recorre) con uno de los algoritmos conocidos de búsqueda (por lo general, con el algoritmo de búsqueda en profundidad) de la posición inicial S hasta la profundidad N;
2. Todos los nodos terminales obtenidos de este árbol, es decir, los nodos que se encuentran en la profundidad N, se evalúan utilizando la función de evaluación estática;

3. De conformidad con el principio minimax se calculan las evaluaciones de todos los otros nodos: en primer lugar la estimación de los nodos que son padres para los nodos terminales, después, para los nodos padres de estos nodos, etc.; esta evaluación continúa en el movimiento hacia arriba en el árbol de búsqueda hasta el momento de evaluar los nodos hijo del nodo inicial, es decir para la configuración inicial S;
4. Entre los nodos hijo del nodo inicial se elige el nodo con la más alta evaluación: la jugada que conduce a este nodo es la mejor forma de continuar en la configuración del juego S.

En la ilustración 19 se muestra el uso del procedimiento minimax para un árbol de juego, construido a una profundidad de N=3. Los nodos terminales no tienen nombres, se marcan con valores de la función de evaluación estática. Los números índices de los nombres de otros nodos muestran el orden en que estos nodos fueron construidos por el algoritmo de búsqueda en profundidad. Al lado de cada nodo se presentan sus evaluaciones minimax obtenidas en el proceso de movimiento en la dirección opuesta con respecto a la construcción del árbol. Por lo tanto, la mejor jugada es la primera de las dos posibles.

En este árbol de juego está marcada una ruta (rama) —secuencia de jugadas de ambos jugadores—, que representa el juego denominado *minimax óptimo* en el cual cada jugador siempre elige la mejor jugada para él. Tomemos en cuenta que las evaluaciones de todos los nodos de en esta ruta (rama) del árbol son las mismas (coinciden), y la evaluación del nodo inicial es igual a la evaluación del nodo terminal de esta rama.

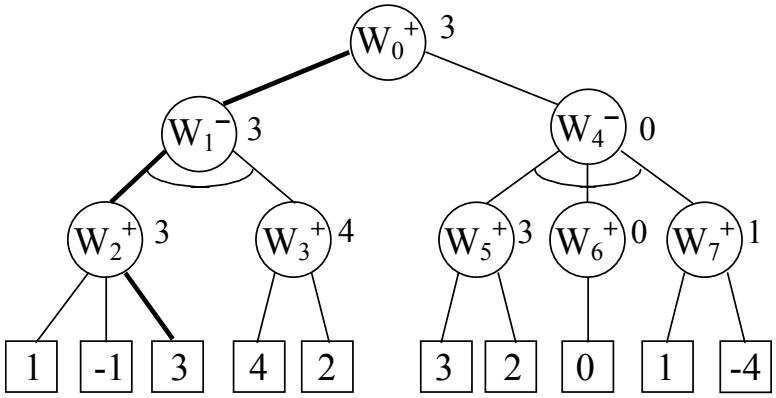


Ilustración 19. Árbol de juego construido con el procedimiento minimax.

En principio, la función de evaluación estática se podría aplicar a los nodos intermedios, y basándose en estas evaluaciones seleccionar la mejor primera jugada; por ejemplo, elegir la jugada que maximiza el valor de la función de evaluación estática entre los nodos hijo y el nodo inicial. Sin embargo, se considera que las estimaciones obtenidas mediante el procedimiento minimax, es una medida más confiable de las propiedades relativas de los nodos

intermedios; en comparación con las estimaciones obtenidas de la aplicación de la función de evaluación estática. De hecho, las estimaciones que se basan en el principio minimax están tomando en cuenta el posible desarrollo del partido en el futuro y ponderan diferentes características que pueden surgir más adelante; mientras que la aplicación directa de la función de evaluación toma en cuenta solamente las propiedades de la propia posición como tal. Esta diferencia de las estimaciones estáticas y estimaciones minimax es muy significativa para los juegos "activos" con las posiciones dinámicas; por ejemplo, en las damas y en el ajedrez, estas son las configuraciones en las que existe la amenaza de tomar una o más piezas. En el caso de los juegos "pasivos" (tranquilos), las posiciones de la evaluación estática pueden no tener muchas diferencias con las estimaciones usando el principio minimax.

4.3 Procedimiento alfa-beta

En el procedimiento minimax, el proceso de construcción de un árbol de juego parcial se separa del proceso de evaluación de los nodos, y esto conduce al hecho de que, en un caso general, el procedimiento minimax es una estrategia ineficaz de búsqueda de buena primera jugada. Para mejorar la efectividad de búsqueda es necesario calcular las evaluaciones (estática y minimax) de los nodos de forma simultánea con la construcción del árbol de juego y no considerar los nodos que son peores de los nodos ya construidos. Esto conduce al procedimiento llamado ***procedimiento alfa-beta*** para encontrar la mejor primera jugada en una posición determinada. Este procedimiento se basa en una consideración bastante obvia para la reducción de la búsqueda: si hay dos variantes de jugadas de un jugador, la peor jugada en algunos casos puede ser inmediatamente descartada, sin determinar que tan exactamente peor es esta jugada.

Consideremos primero la idea del procedimiento alfa-beta en el ejemplo de un árbol de juego que se muestra en la ilustración 19. El árbol de juego se construye a una profundidad de N=3 usando el algoritmo de búsqueda en profundidad. Y tan pronto como se hace posible no sólo se calculan las evaluaciones estáticas de los nodos terminales, pero también la *estimación previa minimax* de los nodos intermedios. Las estimaciones preliminares se determinan como un mínimo o máximo de los nodos hijo que ya se conocen en un momento dado. En general, esta estimación se puede obtener si se conoce la evaluación de al menos un nodo hijo. En el proceso de construcción del árbol de juego y obtención de los nuevos nodos, las estimaciones preliminares se refinan paulatinamente, igual utilizando el principio minimax.

Suponemos que de esta manera se construyen los nodos W_1^-, W_2^+, y los tres primeros nodos terminales (hojas), véase la ilustración 20. Estas hojas son evaluadas por una función estática, y el nodo W_2^+ recibió la estimación minimax exacta de 3, y el nodo W_1^- la evaluación preliminar 3. A

continuación, durante la construcción y la apertura del nodo W_3^+, la evaluación estática de su primer nodo hijo tiene el valor 4, que se convierte en la estimación preliminar del mismo nodo W_3^+. Esta evaluación preliminar después de la construcción de su segundo nodo hijo será recalculada. De acuerdo con el principio minimax la evaluación de un nodo puede solamente aumentar (dado que se toma el valor máximo de los nodos hijo), pero incluso si este valor se aumenta, eso no va a afectar a la evaluación del nodo W_1^- , ya que este durante el refinamiento sólo puede disminuir (dado que en este caso se toma el valor mínimo de los nodos hijo). En consecuencia, podemos ignorar el segundo nodo hijo W_3^+, no construirlo y no evaluarlo, ya que el refinamiento de evaluación del nodo W_3^+ no afecta a la evaluación de nodo W_1^+. Esta reducción de búsqueda en un árbol de juego se llama ***cortar las ramas***.

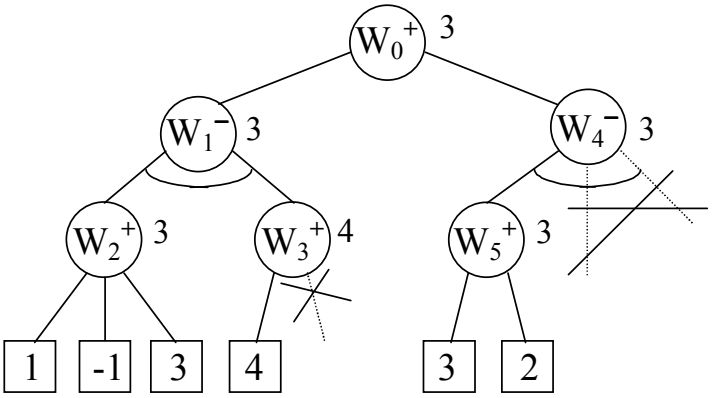


Ilustración 20. Árbol de búsqueda construido con el procedimiento alfa-beta.

Continuamos en nuestro ejemplo, el proceso de búsqueda en profundidad con el cálculo simultáneo de las evaluaciones preliminares (y exactas donde es posible) de los nodos hasta el momento en que ya se construyen los nodos W_4^- y W_5^+, igual que dos nodos hijo del último que se estiman con la función estática. Dado el valor obtenido de la evaluación del primer nodo hijo del nodo inicial W_0^+ (que corresponde a la posición inicial de juego), este nodo tiene un valor de evaluación preliminar igual a 3. El nodo W_5^+ recibió la estimación exacta minimax de 3, y su padre W_4^- obtuvo la estimación preliminar de 3. Esta estimación preliminar del nodo W_4^- puede cambiarse, pero de acuerdo con el principio minimax sólo puede disminuir, y esta disminución no afectará a la evaluación del nodo W_0^+, dado que este último, igual según el principio minimax, solo puede aumentarse. Por lo tanto, la construcción del árbol se puede cortar por debajo del nodo W_4^-, eliminando por completo su segunda y tercera ramas y dejando en el árbol su evaluación preliminar.

Con eso se termina la construcción del árbol de juego. El resultado obtenido que corresponde a la mejor primera jugada, es el mismo que en el

procedimiento minimax. Algunos de los nodos mantuvieron su evaluación preliminar, sin embargo, estas estimaciones aproximadas fueron suficientes para determinar la evaluación exacta minimax del nodo inicial y la mejor primera jugada. Al mismo tiempo se produjo una reducción significativa de la búsqueda: en lugar de 17 nodos se construyeron 11, y se llevó acabo sólo 6 llamadas a la función de evaluación estática en lugar de 10.

Generalicemos la idea de reducción de búsqueda considerada. Formulamos primero las reglas para el cálculo de evaluaciones de nodos de un árbol de juego, incluyendo las evaluaciones preliminares de los nodos intermedios, que vamos a llamar ***valores alfa*** y ***beta***:

- Un nodo terminal (hoja) de un árbol de juego se evalúa usando la función de evaluación estática tan pronto como se construye;
- Un nodo intermedio se evalúa preliminarmente usando el principio minimax, tan pronto cuando se conozca la evaluación de uno de sus nodos hijo; cada evaluación preliminar se recalcula cada vez que se obtenga una nueva evaluación de uno de sus nodos hijo;
- Una evaluación preliminar de los O-nodos (valor alfa) es igual a la mayor de las evaluaciones calculadas hasta el momento de sus nodos hijo;
- Una evaluación preliminar de los Y-nodos (valor beta) es igual a la menor de las evaluaciones calculadas hasta el momento de sus nodos hijo.

Podemos señalar una consecuencia obvia de estas reglas de cómputo: los valores alfa no pueden disminuir, y los valores beta no pueden crecer.

Ahora formulamos dos reglas cuando hay que terminar la búsqueda, o cortar las ramas del árbol de juego:

- Se puede terminar la búsqueda por debajo de cualquier Y-nodo, cuyo valor beta no supere (es menor o igual) el valor alfa de uno de los O-nodos que se encuentren arriba en el árbol (incluyendo el nodo raíz del árbol);
- Se puede terminar la búsqueda por debajo de cualquier O-nodo, cuyo valor alfa es mayor o igual que el valor beta de uno de los Y-nodos que se encuentren arriba en el árbol.

En el primer caso se dice que ha habido un ***corte alfa***, ya que se cortan las ramas del árbol empezando con un O-nodo que tienen asignados valores alfa; el segundo caso es un ***corte beta*** dado que se cortan las ramas empezando con valores beta.

Es importante notar que la reglas consideradas se aplican durante la construcción del árbol de juego en profundidad, lo que significa que la evaluación preliminar de los nodos intermedios sólo aparece según se hace la subida en el árbol de los nodos terminales hasta la parte superior y realmente el corte comience sólo después de que se obtuvo al menos una estimación minimax exacta de un nodo intermedio.

En el ejemplo anterior de la ilustración 20 el primer corte fue el corte beta, y el segundo el corte alfa. En ambos casos, el corte era poco profundo dado que el valor que se necesita para la aplicación de la regla correspondiente se encontraba en el nodo inmediatamente anterior al punto del corte. En el caso general, el punto de corte puede encontrarse significativamente más alto en el árbol, incluyendo el nodo inicial.

Después de la terminación de la búsqueda en alguna rama del árbol, la evaluación preliminar de sus nodos permanece sin especificar, pero, como ya se señaló, esto no impide una evaluación preliminar correcta de todos los nodos que se encuentran arriba, así como una evaluación exacta del nodo raíz y sus nodos hijo, lo que permite encontrar la mejor primera jugada.

En las reglas anteriores de cálculo de evaluaciones de los nodos y realización de cortes (siempre que sea posible) se basa el ***procedimiento alfa-beta***, que es una realización más eficaz del principio minimax. En principio, aplicando la inducción matemática y el razonamiento como se mostró en el ejemplo anterior, fácilmente podemos probar la siguiente afirmación:

El procedimiento alfa-beta siempre conduce al mismo resultado —la mejor primera jugada—, que el procedimiento minimax de la misma profundidad.

Una cuestión importante es hasta qué punto en promedio el procedimiento alfa-beta es más eficaz que el procedimiento minimax. Es fácil observar que el número de cortes en el procedimiento alfa-beta depende del grado en que las evaluaciones preliminares (valores alfa y beta) obtenidas como primeras aproximan las evaluaciones minimax finales: cuanto más cerca están estas estimaciones, más cortes existen y la búsqueda es menor. Esta situación se ejemplifica en la ilustración 20, donde la rama del árbol de juego que es minimax óptima se construye casi al principio de la búsqueda.

Por lo tanto, la eficacia del procedimiento alfa-beta depende del orden de la construcción y apertura de los nodos en un árbol de juego. Puede suceder un orden poco beneficioso, en el cual se presentara la necesidad de recorrer todos los nodos del árbol, y que, en el peor de los casos, el procedimiento alfa-beta no tendrá ninguna ventaja en comparación con el procedimiento minimax.

El mayor número de cortes se presenta cuando la búsqueda en profundidad descubre primero el nodo con la evaluación estática máxima que después se convertirá en la evaluación minimax del nodo inicial. Cuando el número de cortes es máximo, se construye y se evalúa un número mínimo de los nodos terminales. Se mostró (Nilsson 1971), que cuando las jugadas más poderosas se consideran primero, el número de nodos terminales de la profundidad N, que serán construidos y evaluados por el procedimiento alfa-beta, aproximadamente es igual al número de nodos terminales, que serían construidos y evaluados en una profundidad N/2 por un procedimiento minimax convencional. Así, con un tiempo y la memoria fijos el procedimiento alfa-beta puede buscar dos veces más profundo que el procedimiento minimax estándar.

En conclusión, mencionamos que la función de evaluación estática y el procedimiento alfa-beta son dos componentes indispensables de casi todos los juegos computacionales, incluidos los comerciales. A menudo durante la búsqueda en el árbol de juego se utilizan adicionalmente los métodos heurísticos (Russell, Norvig 1996):

- *Cortes direccionales* (heurísticos) de ramas poco prometedoras: por ejemplo, la construcción de un árbol de juego se interrumpe en una posición "pasiva" y se utiliza la función de evaluación estática; mientras que para las posiciones "activas" la búsqueda continúa hasta la profundidad deseada;
- *Ordenamiento dinámico* de los nodos, en el que después de cada ciclo de refinamiento de los valores minimax y de los cortes, se realiza reordenamiento de los nodos del árbol de juego construido para el momento actual (usando alguna función heurística), y se elige para la apertura el nodo más prometedor;
- *Profundización progresiva*, en la que el procedimiento alfa-beta se aplica varias veces: primero, para alguna profundidad pequeña (por ejemplo, profundidad 2), después la profundidad se incrementa en cada iteración; y después de cada iteración se hace reordenamiento de todos los nodos hijo, con el fin de aumentar el número de cortes en iteraciones posteriores.

5. Conclusiones

Durante mucho tiempo el desarrollo de algoritmos de búsqueda heurística sigue como una línea de la investigación pertinente en el área de inteligencia artificial. Esto se debe al hecho de que en muchos problemas combinatorios interesantes el espacio de búsqueda se aumenta de manera exponencial, es decir, el número de estados posibles crece exponencialmente al aumentar la profundidad de búsqueda. Por esta razón, la búsqueda heurística a menudo es el único método práctico para solucionar esos problemas. Entre los problemas que no pueden ser resueltos en un tiempo razonable sin el uso de las heurísticas, son dos de las más antiguas aplicaciones en el campo de la inteligencia artificial: demostración de teoremas y juegos.

Un papel clave en la búsqueda heurística, es la evaluación heurística. El principal criterio para su elección es su eficacia para resolver problemas específicos. El desarrollo de buenas heurísticas puede reducir drásticamente espacio de búsqueda, pero no es un problema fácil. Se requiere un análisis exhaustivo previo de las maneras de formalizar el problema, igual que las experiencias de su solución. En el trabajo (Pearl, 1984) se describen en detalle las cuestiones relacionadas con el diseño y análisis de heurísticas.

Inevitable costo de la eficacia de las heurísticas es el hecho de que en algunos casos, una heurística no permite encontrar una solución óptima o no permite encontrar una solución aunque exista. Esta limitación es inherente a la búsqueda heurística y no puede ser eliminada mediante algoritmos de búsqueda más eficiente.

Un aspecto importante que debe tomarse en cuenta en el diseño y la selección de la heurística es su complejidad computacional, ya que ésta debe participar en el cálculo del coste global computacional de una solución. Si la complejidad computacional de alguna función heurística eficaz es muy alta, puede ser preferible una heurística más fácil de calcular, aun cuando el espacio de búsqueda generado con su aplicación sea mucho más grande. Por lo tanto, es necesario optimizar el balance entre el tamaño de espacio de búsqueda y la evaluación heurística. La combinación óptima de esos dos elementos todavía es una cuestión abierta en muchos problemas incluyendo el juego de ajedrez y demostración de teoremas.

Tareas

1. Considerar el problema del mono y el plátano con elementos adicionales: En la sala hay dos cajas (en diferentes lugares), y el mono no sólo puede subir a la caja, sino también bajar de ella (sin el plátano). Formalizar el problema en el espacio de estados; incluir descripción de los estados inicial y final, así como los operadores del problema. Dibujar el espacio de estados obtenido como un grafo y marcar la ruta solución.

2. Formalizar en el espacio de estado el problema de ocho reinas (especificar la forma de descripciones de los estados, describir los operadores, los estados inicial y final).

Colocar 8 reinas en un tablero de ajedrez normal 8 reinas de tal manera que en cada horizontal, vertical y diagonal quedase no más de una reina.

Construir una parte del árbol de búsqueda para resolver este problema usando uno de los algoritmos de búsqueda ciega.

3. Formalizar el siguiente problema para la búsqueda en el espacio de estados.

Un padre y sus dos hijos tienen que cruzar en un barco de una orilla del río a otra. ¿Cómo hacerlo, si el peso del padre es de 80 kg, el peso de los hijos es 40 kg de cada uno, y el barco puede cargar no más de 80 kg?

Construir para la formalización propuesta el árbol de búsqueda en profundidad y el árbol de búsqueda en amplitud. Marcar en cada uno de los árboles la solución encontrada. Enumerar los nodos de cada árbol según su orden de construcción.

4. Para la configuración siguiente del juego en “ocho” construir el árbol de búsqueda usando el algoritmo de búsqueda heurística con una función de evaluación que es igual al número de fichas que no estén en sus lugares.

1	2	3
	7	4
6	8	5

$\Rightarrow$

1	2	3
8		4
7	6	5

El orden de aplicación de los operadores que mueven la ficha "vacía" es fijo: izquierda, derecha, arriba, abajo. Entre los nodos con la misma evaluación se selecciona el nodo de menor profundidad para continuar la búsqueda. Calcular para cada nodo construido del árbol su evaluación, enumerar todos los nodos según el orden de su construcción, señalar en el árbol la ruta solución.

5. Sugerir algunas funciones de evaluación para la búsqueda heurística en el espacio de estados para:

a) El problema del mono y del plátano.

b) El problema del viajero.

6. Para la variante del problema de La Torre de Hanoi con cuatro discos, dibujar el grafo Y/O que muestra la solución de este problema con el método de reducción.

7. Consideramos el juego "el último pierde", en el que dos jugadores (MAS y MENOS) en turnos toman 1, 3, o 5 monedas de un montón que inicialmente contiene 9 monedas. El jugador que toma la última moneda, pierde. Para el problema de encontrar una estrategia ganadora en las posiciones del juego $V(9^+)$, dibujar un árbol de juego completo (grafo) y mostrar en él, un grafo de solución que demuestre que el segundo jugador MENOS puede ganar siempre. Formular la estrategia ganadora del MENOS verbalmente. Si al inicio del juego el montón de monedas tendría K monedas (K es un número natural), ¿con qué valores de K la estrategia formulada será aplicable?

8. Sugerir una función de evaluación estática para el juego de damas.

9. Para la posición determinada siguiente de juego del "gato" (tres en línea) encontrar la mejor jugada basada en el procedimiento minimax con la profundidad de búsqueda en el árbol de juego igual a 2. Utilizar cualquier función de evaluación estática adecuada. Dibujar el árbol de juego resultante, asignar a todos los nodos su evaluación.

	X	
X	X	O
O	O	

10. Para la posición determinada siguiente de juego del "gato" (tres en línea) encontrar la mejor jugada basada en el procedimiento alfa-beta con la profundidad de búsqueda en el árbol de juego igual a 2, utilizando la función de evaluación estática presentada en la sección 4. Dibujar el árbol de juego resultante, enumerar los nodos según el orden de su construcción, asignar a los nodos su evaluación.

X	O	
	X	

¿Cuántos cortes se hicieron? ¿Cuántos nodos del árbol de juego no se construyeron en comparación con el procedimiento minimax? ¿Cómo cambiarían estos valores si los nodos hijo se evaluaran por el procedimiento alfa-beta en un orden diferente?

11. ¿Qué es una función de evaluación heurística? ¿Sobre qué conjunto se define, que valores toma, cuál es la interpretación de estos valores? ¿Para qué sirve? Dar ejemplos de funciones de evaluación.

Referencias

[1] Luger, G. F. Artificial Intelligence. Structures and Strategies for Complex problem Solving (Fourth Edition). Addison-Wesley, 2001.

[2] Nilsson, N. J. Problem-Solving Methods in Artificial Intelligence. New York: McGraw-Hill, 1971.

[3] Nilsson, N. J. Principles of Artificial Intelligence. Morgan Kaufmann, 1980.

[4] Nilsson, N. J. Artificial Intelligence: A New Synthesis. Morgan Kaufmann, 1998.

[5] Pearl, J. Heuristics. Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, 1984.

[6] Slagle, J. R. Artificial Intelligence: The Heuristic Programming Approach. New York: McGraw-Hill, 1971.

[7] Rich, E., Knight K. Inteligencia Artificial. McGraw-Hill/InterAmericana de España, S.A., 1994.

[8] Russel, S. Norvig P. Inteligencia Artificial. Un enfoque moderno. Prentice Hall Hispanomericana, 1996.

[9] Winston, P. Artificial Intelligence (Third Edition). Addison-Wesley, 1993.

Ontologías y Representación del Conocimiento

Manuel Vilares Ferro
vilares@uvigo.es

Santiago Fernández Lanza
sflanza@uvigo.es

Milagros Fernández Gavilanes
mfgavilanes@uvigo.es

Francisco José Ribadas Pena
ribadas@uvigo.es

Área de Ciencias de la Computación e Inteligencia Artificial
Escuela Superior de Ingeniería Informática
Universidade de Vigo

1.- Introducción

Desde la antigüedad, se ha intentado analizar, definir o descubrir las leyes que rigen lo que cae bajo el concepto de conocimiento. A lo largo de la historia del pensamiento se han realizado propuestas más o menos explicativas que tenían por objetivo clarificar esta compleja noción. Por regla general, cuando alguna de estas propuestas tiene cierto carácter formal o aplicado suelen venir asociadas a intentos más o menos exitosos de representación del conocimiento. Tanto la tarea de clarificación de la noción de conocimiento como la de su representación no son tarea fácil; ello se pone de manifiesto en la multitud de discusiones que, aún hoy día, se tienen al respecto en disciplinas como la Psicología, la Filosofía, la Lingüística, las Ciencias de la Computación, etc.

Conscientes de esta dificultad, comenzaremos el presente capítulo recordando, de una forma sintética, algunos de los intentos más relevantes de definición del término "conocimiento" y estableciendo una clásica distinción, propia del ámbito de las Ciencias de la Computación, entre las representaciones procedimentales, estructuradas y declarativas del conocimiento. A continuación se presentará la conexión existente entre la representación del conocimiento y diversos modos de representación del razonamiento como la Lógica Clásica (Proposicional y de Primer Orden), los Sistemas Basados en Reglas y representaciones estructuradas como son las Redes Semánticas, los Grafos Relacionales de Quillian, los Grafos de Dependencia Conceptual de Schank, la Jerarquía de Conceptos o los Marcos. Posteriormente se explicitarán algunas propuestas de representación de conocimiento incierto como la Teoría de las Probabilidades, el Modelo de los Factores de Certidumbre o la Lógica Difusa.

Finalizaremos el capítulo desarrollando de forma un poco más extensa el tema de las ontologías; uno de los métodos de representación del conocimiento que ha cobrado gran importancia con el advenimiento de las grandes redes de información como Internet y más concretamente, dentro del ámbito de la World Wide Web, con el desarrollo de lo que se conoce con el nombre de "Web Semántica". De éstas propondremos una definición, estudiaremos los tipos que existen, analizaremos propuestas metodológicas para su diseño, haremos una breve incursión en el lenguaje OWL específico para la implementación de ontologías y haremos referencia a unas cuantas herramientas de desarrollo como Protégé, Jena o DAMLdotnetAPI, para terminar haciendo referencia a unos cuantos ejemplos concretos de ontologías.

1.1.- Una aproximación a la definición de "conocimiento"

A grandes rasgos, las definiciones que se han dado de "conocimiento" podrían clasificarse en dos grupos: las que lo consideran como una cualidad o capacidad de un sujeto o agente, sea éste humano o no, o las que lo consideran como algún tipo de entidad abstracta como puede ser la creencia, la información, etc.

Entre las primeras se encuentra la que figura como segunda acepción del término en el Diccionario de la Real Academia Española: "Entendimiento, inteligencia, razón natural". También la de Newell: "Lo que puede atribuirse a un agente de tal modo que su comportamiento pueda computarse de acuerdo con el principio de racionalidad" [Newell, A (1981)]. Basta echar un breve vistazo a la web para encontrarnos propuestas de definición que ahondan esta forma de concebir el conocimiento. Así encontramos, "conocimiento es la capacidad de resolver un determinado conjunto de problemas con una efectividad determinada"[1], "conocimiento es la capacidad para convertir datos e información en acciones efectivas"[2], etc. Todas ellas consideran al conocimiento como una capacidad que hace que un sujeto o agente se comporte, opere, actúe o resuelva problemas de determinada manera, a la cual, en ocasiones se pone la etiqueta de "racional".

Entre el segundo grupo de definiciones, destaca la utilizada tradicionalmente en Epistemología o Teoría del Conocimiento, disciplina filosófica que estudia este concepto, definiéndolo en principio como "Creencia verdadera y justificada". Esta definición tiene su origen en el *Teeteto* platónico [Platón (2003)] pero aún es defendida hoy día por muchos epistemólogos. En contextos más científico-técnicos y menos filosóficos se suele considerar al conocimiento como un conjunto organizado de datos o información que se utilizan con un determinado propósito, ya sea resolver o contribuir a la resolución de un determinado problema, o como ayuda en la toma de una decisión.

En todo caso, podemos estar de acuerdo con Hanson y Bauer cuando afirman que conceptos como el de conocimiento son conceptos polimorfos para los cuales no se puede dar una definición cuya validez sea universal [Hanson, S.; Bauer, M. (1989)]. Más bien, lo que se puede dar son definiciones contextualmente adecuadas en ámbitos concretos. Lo que se hará en el presente apartado será proporcionar una definición de "conocimiento" sin pretensiones de universalidad pero adecuada al contexto de la representación del conocimiento en el ámbito de las Ciencias de la Computación y la Inteligencia Artificial. Para ello, conviene previamente establecer los componentes que juegan algún tipo de papel en este escenario:

- *Ambiente o entorno*: Todos los componentes están incardinados dentro de lo que podríamos denominar "ambiente" o "entorno". La definición puede parecer muy amplia pero dado que estamos hablando de sistemas informáticos, el ambiente o entorno será aquello que envuelve a cualquier sistema informático de tal forma que éste interactúe con el. Por lo tanto, será aquello de lo que un sistema informático recibe datos y aquello sobre lo que tal sistema actúa.
- *Agente*: Denominaremos "agente" al sistema informático al que se hacía referencia en el punto anterior. Este sistema obtendrá datos del ambiente o entorno y actuará sobre éste, pero lo hará de forma limitada y con un determinado propósito. Es decir, los sistemas informáticos son herramientas limitadas que no capturan cualquier tipo de datos sino una serie de datos específicos que son de interés para el propósito para el que fue creado. Del

[1] http://www.gestiondelconocimiento.com/conceptos_conocimiento.htm
[2] http://www.daedalus.es/inteligencia-de-negocio/gestion-del-conocimiento/que-es-el-conocimiento/

mismo modo no realizan cualquier tipo de acción sobre el entorno sino acciones concretas con el fin de llevar a cabo un propósito concreto.

- *Datos:* Los datos son aquello que entra en el sistema desde el ambiente o entorno. Sin embargo, los datos por sí mismos no son de utilidad para el sistema si este no es capaz de capturar su significado o interpretarlos.
- *Información*: Cuando los datos son interpretables por el agente, entonces éste puede captar su significado. Cuando esto sucede, estamos hablando de datos significativos, datos interpretados, datos informativos o simplemente información.
- *Proceso de razonamiento*: Para que el agente sea capaz de conseguir su propósito u objetivo, debe realizar un proceso de razonamiento que, en función de la información recibida, le indique cuál es la acción que debe llevar a cabo sobre el entorno. Dicha acción deberá ser adecuada a su propósito si el agente está correctamente diseñado e implementado.

Considerados así los componentes, podremos definir "conocimiento" como la información que es utilizada por los procesos de razonamiento del sistema informático y que se usa con un determinado objetivo o propósito.

Ahora bien, todos los conceptos que se están manejando aquí como "ambiente", "agente", "datos", "información", "razonamiento" y por supuesto el propio concepto de "conocimiento" son lo suficientemente enigmáticos como para que en el campo de la computación no se puedan manejar en su "estado puro". Lo que solemos manejar son maneras de representar tales cosas como "información", "razonamiento", "conocimiento", etc. Esta representación no es más que la expresión de tales conceptos mediante la utilización de lenguajes específicamente diseñados para tal fin. En este sentido, es conveniente señalar que no deben confundirse tales conceptos con la representación que se hace de ellos en ámbitos como las Ciencias de la Computación [Fernández Fernández, G. (2004)], aunque, en la mayoría de los casos, utilicemos los propios términos ("conocimiento", "información", "razonamiento", etc.) para referirnos a sus representaciones.

Por otra parte, al definir el conocimiento como información que es utilizada por los procesos de razonamiento del sistema informático, establecemos un vínculo entre el conocimiento y el razonamiento que es difícil de disociar en la práctica. Es decir, el conocimiento sin ningún sistema de razonamiento que lo maneje no resulta de gran utilidad. Por el contrario, un mecanismo de razonamiento que no sea alimentado con conocimiento no sirve de nada. Por regla general, buena parte de las propuestas de representación del razonamiento llevan asociado un tipo de representación del conocimiento. Este es el motivo por el que dedicamos el apartado 2 y el apartado 3 del presente capítulo de representación del conocimiento a la cuestión de la representación del razonamiento.

1.2.- Tipos de conocimiento

La necesidad de modelizar los procesos de la inteligencia humana, ha originado el nacimiento de diferentes teorías sobre su funcionamiento. En este sentido, la Inteligencia Artificial ha tratado de encontrar en la "inteligencia" del ordenador una emulación de la mente humana. Hoy en día, la Inteligencia Artificial se subdivide en dos vertientes que se complementan [Newell, A (1981)]. Una corriente tradicional o también denominada *simbólica* [Begeer, S.; Chrisley, R. (2000); Partridge D.; Wilks, Y. (1990)], basada en el estudio del comportamiento cognitivo del ser humano ante

diferentes problemas para la creación de sistemas inteligentes de propósito específico que emulan dichos comportamientos; y la *subsimbólica* [Begeer, S.; Chrisley, R. (2000); Partridge D.; Wilks, Y. (1990)], cuyo máximo exponente es la Inteligencia Artificial Conexionista[3], que trata de simular los elementos de más bajo nivel que componen o intervienen en los procesos inteligentes, con la confianza que de su combinación surja de forma espontánea el comportamiento inteligente. Concretamente, tratan de simular la fisiología del sistema nervioso humano independientemente de su aplicación a la resolución de problemas concretos. Otro modo de explicar la diferencia entre ambas vertientes, es la fuente de inspiración: la psicología en el caso de la Inteligencia Artificial *Simbólica*[4] y la biología en el caso de la *Subsimbólica*[5].

El enfoque que trata de seguir el presente trabajo se centra en la vertiente simbólica, cuyas técnicas hacen uso de una representación del conocimiento basada en símbolos. Esta representación se inspira en la lógica matemática, haciendo uso de los símbolos matemáticos para establecer una representación de objetos del mundo. Un problema que usa este tipo de representación del conocimiento, es expresado como una colección de símbolos sobre los que se aplican los algoritmos de procesamiento e inferencia, de tal forma que esto lleve a una solución eficaz [Newell, A (1981)]. En síntesis, una máquina capaz de trabajar con un lenguaje simbólico puede procesar conocimiento representado de manera igualmente simbólica.

Un *símbolo* es una cadena de caracteres que representa una idea o un objeto y es la unidad mínima de representación matemática del conocimiento. Estas representaciones están bastante alejadas de los lenguajes naturales, que son, podríamos decir, el tipo de lenguaje de representación utilizado por los seres humanos. Es más, una representación simbólica del conocimiento que pretenda ser acertada, debe ser fácilmente manejable, bien por acción de un ser humano o de un programa automático. En efecto, esta cualidad permite la depuración del conocimiento almacenado en una base de datos simbólica, bien mediante la incorporación de nuevo conocimiento o la depuración del ya existente.

Históricamente han surgido varios formalismos y en consecuencia diferentes tipos de lenguajes para la representación del conocimiento, cuyo estudio permitirá comprender la estructuración y organización de la información. Ryle[6], filósofo británico representante de la escuela de Oxford, criticó la "leyenda intelectualista" que sostenía que la inteligencia es una facultad que se realiza mediante actos internos de pensamientos y que consiste en el examen de proposiciones. Esta doctrina reconoce que "saber qué" no significa "saber cómo" [Ryle, G. (1946)].

Ya en 1949, en los distintos ámbitos de la investigación, fue tomándose en consideración la distinción entre dos formas de conocimiento humano [Ryle, G.

---

[3] El *conexionismo* es un modelo de pensamiento basado en la Inteligencia Artificial cuyo origen se remonta a las primeras investigaciones iniciadas en las décadas de los sesenta y setenta sobre redes neuronales y algoritmos genéticos. Alejándose del determinismo reduccionista de considerar la cognición como simples manipulaciones simbólicas, mantenían que la inteligencia humana, no es sólo un conjunto de reglas lógicas.

[4] Como ejemplo representativo tenemos el proyecto CyC de Douglas B. Lenat, con un sistema que posee en su memoria millones de hechos interconectados.

[5] El primer modelo de red neuronal fue propuesto en 1943 por McCulloch y Pitts.

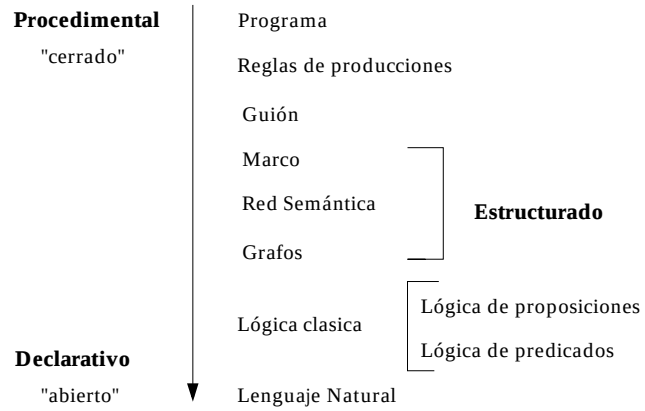
[6] (Brighton, 19/08/1900 – Oxford, 06/10/1976). Es considerado como uno de los miembros destacados de la llamada escuela analítica, cuya parte central considera que la mente humana no puede alcanzar ningún esquema que explique al universo, por lo que la metafísica es imposible. Debe limitarse a la consideración de problemas lógicos y metodológicos.

(1949)]. Efectivamente, a partir de la evolución de la psicología cognitiva[7] y sobre los numerosos estudios realizados acerca de la memoria humana, se muestra que existen dos tipos simultáneos de conocimientos, pero que a la vez son distintos, en la estructura cognitiva de las personas: el conocimiento *procedimental* y el *declarativo*.

Este debate, se retomó parcialmente en Inteligencia Artificial, donde existe cierta controversia. Para los seguidores de una concepción *procedimental* del conocimiento, la inteligencia descansa sobre un único conjunto ya que el conocimiento no se puede separar de los procedimientos, es decir el conocimiento coexiste con el conjunto de programas y procedimientos que dispone para actuar. Esta concepción invoca la simplicidad y la facilidad de comprensión del razonamiento representado por algoritmos que simulan el comportamiento real. En cambio, para aquellos que son seguidores de la concepción *declarativa* del conocimiento [Michel F. (1990)], la esencia del conocimiento reside en la posesión de dos conjuntos separados: un conjunto específico de hechos que describen dominios concretos de conocimiento y un conjunto general de procedimientos para manipular estos hechos.

Por otra parte, las representaciones procedimentales son más eficaces en ejecución, por su proximidad formal a un algoritmo computacional, pero su mantenimiento es más difícil. La ejecución de un procedimiento es también más fácil de seguir, puesto que uno puede examinar simplemente el flujo de instrucciones y se puede traducir inmediatamente a un programa escrito en un lenguaje de programación.

Tomando como base la clasificación realizada por Laurière [Laurière, J. L. (1982)], la siguiente figura presenta los distintos formalismos que van de lo procedimental (más estructurado) a lo más declarativo (más abierto y libre).



Concretamente, basándonos en esta clasificación, las representaciones pueden agruparse en tres grandes clases: *procedimentales*, *estructuradas* y *declarativas*. A continuación, se verá con más detalle cada una de ellas.

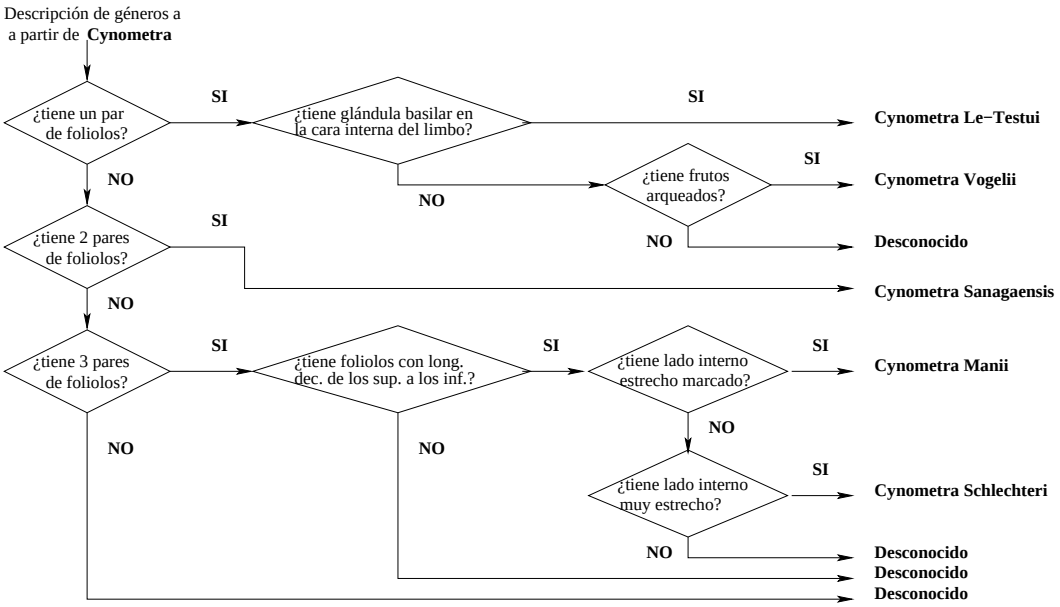
1.2.1.- Representación procedimental del conocimiento

La representación procedimental es la forma menos abstracta y más cercana a la representación del conocimiento que asociamos a un ordenador. El conocimiento se ocupa de cómo utilizamos la información, cómo operamos con ella y, en general, de los procedimientos que empleamos para alcanzar los conocimientos y recuperarlos cuando

[7] En la que la cognición se convierte en la manipulación y transformación de un conjunto de símbolos a partir de ciertas reglas [Newell, A (1981)].

han de ser utilizados. Permite tratar problemas de tipo algorítmico, es decir, completamente analizados e íntegramente conocidos. Utiliza información perfectamente estructurada y traduce "cómo" transita el conocimiento, lo que supone una simulación del comportamiento real.

Usando un ejemplo del ámbito de la botánica, se va a considerar el conocimiento que se tiene a la hora de describir los géneros que proceden de la especie *Cynometra*. Un procedimiento podría ser el indicado en la siguiente figura:



Siguiendo con esta idea, surgen los sistemas de *reglas de producción* que veremos más adelante, formalismo que incluye reglas de comportamiento de la forma: "si *condición* entonces *acción*". La acción de una regla se ejecuta cuando se cumple su condición. Representar el conocimiento aprendido significa modificar la condición o acción de una regla, o compactar o disociar éstas.

Por último, los *esquemas de acción* o *guiones* (*scripts*) permiten describir de forma genérica el conocimiento procedimental. Un esquema de acción es una representación estructurada y estereotipada de una secuencia de acciones complejas, compuesta por una sucesión de escenas o eventos que describen el devenir esperado de los acontecimientos en una determinada situación.

Por su carácter dinámico y difícilmente verbalizable, la representación procedimental es más difícil de modelar, debido a la imposibilidad de separarla formalmente de su procesado. Aunque, suele ser más eficiente computacionalmente, deja de serlo en el momento en el que el dominio que se desea representar posee *incertidumbre* relativa a los métodos de procesamiento del conocimiento. Se dice que existe incertidumbre cuando dichos métodos no están matemáticamente respaldados o cuando la constante evolución del conocimiento del dominio exige una evolución paralela de sus métodos de procesado.

1.2.2.- Representación estructurada del conocimiento

La representación estructurada de conocimiento se define como aquella representación en la que se describe la manera en que las ideas son integradas o

interrelacionadas [Diekhoff, G. M.; Diekhoff, K. B. (1982)]. Consiste, pues, en esquemas que a su vez poseen patrones que relacionan los conceptos. Concretamente, Jonassen justificó el estudio del conocimiento estructurado porque considera que es la forma fundamental de recordar, comprender y aprender [Jonassen, D. H. (1985)]. En este sentido, una representación estructurada muestra como interactúan los conceptos dentro de un dominio y se podría decir que se encuentra a medio camino entre las representaciones procedimentales y declarativas.

Dentro de este tipo de representación y siguiendo con la clasificación realizada por Laurière [Laurière, J. L. (1982)], un escalón por encima de la representación declarativa en términos de abstracción se encuentra la *representación relacional*, que hace uso de conceptos y relaciones entre ellos, para representar el conocimiento. Este tipo de acercamiento es fuertemente dependiente del tipo de contenido que se desea representar. Los conceptos se estructuran en torno a tuplas que contienen atributos específicos para almacenar información. Dependiendo del ámbito con el que se esté tratando, los atributos y conceptos de la representación relacional varían. Concretamente, este tipo de modelo de representación tuvo un precedente con las *redes semánticas* [Quillian, M. R. (1967)] que veremos más adelante, paradigma de representación basado en un modelo de la memoria humana, donde el significado de un concepto reside en las relaciones con otros conceptos de la red.

Posteriormente, la *representación jerárquica* surge como una evolución de la relacional, aumentando su grado de abstracción y añadiendo un tipo específico de relación entre concepto, la *herencia*. Esta relación permite agruparlos conforme a la similitud de sus atributos, los cuales se transmiten de forma descendente a través de una estructura jerárquica. Los conceptos más elevados dentro de la jerarquía son los más abstractos, siendo los inferiores especificaciones de aquellos conceptos de los que heredan. Esta amplitud del espectro de abstracción permite a la representación jerárquica procesar información a distintos niveles de profundidad o granularidad. Posiblemente, el modelo que mejor refleja una estructura jerárquica es la que introdujo Minsky, el denominado *marco* (del que daremos cuenta en los próximos apartados) como elemento de alto nivel que organiza y estructura la representación del conocimiento [Minsky, M. (1974)]. Ambas ideas fueron incorporadas inmediatamente a la corriente principal de investigación en la Inteligencia Artificial, aunque algunos autores consideraron que estos paradigmas eran poco más que variaciones de la lógica de primer orden [Hayes, P. J. (1979), Nilsson, N. J. (1982)].

1.2.3.- Representación declarativa del conocimiento

La representación declarativa es una extensión de la representación procedimental que resuelve algunos de los inconvenientes de ésta, permitiendo la representación por separado del conocimiento y de las técnicas para su procesado. De esta forma, en un dominio con incertidumbre, se pueden ensayar distintas representaciones de conocimiento para resolver uno o varios problemas relacionados, de tal forma que, en función del rendimiento y resultados de cada una de ellas, la representación del conocimiento se va refinando hasta alcanzar un alto grado de eficiencia.

La representación declarativa del conocimiento pretende tratar el conocimiento de tal forma que su resultado representacional responda a la pregunta "¿qué?" y no a la pregunta "¿cómo?" como sucedería en el caso de la representación procedimental del

conocimiento. En los sistemas automáticos que modelizan las propuestas de representación declarativa del conocimiento suele distinguirse entre la representación del conocimiento en sí misma y el modo en cómo, a partir de éste, se puede generar más conocimiento. La representación del conocimiento se lleva a cabo mediante la expresión de *hechos* de carácter concreto (por ejemplo, siendo *a* y *b* plantas concretas que se pasan a considerar dentro del sistema, serían hechos "*a* es un Cynometra", "*a* tiene dos pares de foliolos", "*b* es un Cynometra", "*b* tiene un par de foliolos", "*b* tiene una glándula basilar en la cara interna del limbo" etc.) y la expresión de *reglas* y *definiciones* de carácter más general (por ejemplo, "si X es un Cynometra y X tiene dos pares de foliolos entonces X es un Cynometra Sanagaensis", "si X es un Cynometra y X tiene un par de foliolos y X tiene una glándula basilar en la cara interna del limbo, entonces X es un Cynometra Le-Testui"). Para generar más conocimiento a partir del explicitado en los hechos y las reglas suelen utilizarse *motores de inferencia*, los cuales, son capaces de manejar las reglas y definiciones de tal forma que, junto con los hechos representados y mediante procedimientos de razonamiento automático, se obtenga un conocimiento que no había sido explicitado por los hechos, reglas y definiciones originales (por ejemplo, un motor de inferencia podría aplicar la regla del ejemplo anterior y obtener que "*a* es un Cynometra Sanagaensis" o que "*b* es un Cynometra Le-Testui").

Además de todo lo dicho, estos sistemas automáticos de representación declarativa del conocimiento suelen llevar incorporado un *sistema de consulta* de tal manera que se puede interpelar al sistema para averiguar si determinado hecho es verdadero o no, ya sea este un hecho de los incorporados originalmente en el conjunto de hechos o bien un hecho inferido mediante el motor de inferencia. Frecuentemente, estos sistemas de consulta suelen también ofrecer la posibilidad de realizar consultas más generales mediante el uso de variables, las cuales, serán instanciadas por el sistema con el uso del motor de inferencia (por ejemplo, "¿cuáles son Cynometra?" a lo cual nuestro sistema respondería con *a* y *b*, "¿cuáles tienen dos pares de foliolos?" cuya respuesta sería *a*, etc.).

2.- Representación del conocimiento y razonamiento

Inicialmente, habíamos definido "conocimiento" como la información que es utilizada por los procesos de razonamiento de un sistema informático y que se usa con un determinado objetivo o propósito. En buena medida, tales procesos de razonamiento, que también son representacionales, suelen llevar asociadas determinadas formas de representación del conocimiento y su mecanismo de representación por excelencia es la Lógica. Ésta surge de los esfuerzos de filósofos y matemáticos por caracterizar los principios básicos de la argumentación o razonamiento. La idea principal es definir un catálogo de estructuras que no nos lleve a error a la hora de obtener una conclusión a partir de un conjunto de premisas.

Ahora bien, debemos tener en cuenta que existen múltiples modos de razonamiento que corresponden a múltiples versiones de la Lógica. La versión más estándar de la Lógica es aquella que pretende proporcionar un modelo para el razonamiento deductivo. Razonamientos como "los Cynometras que tienen dos pares de foliolos son Cynometras Sanagaensis. *a* es Cynometra y tiene dos pares de foliolos. Por tanto, *a* es un Cynometra Sanagaensis" son del tipo deductivo, y en ellos el que los profiere, pretende que la conclusión (el enunciado que va después de la expresión "Por tanto") se siga necesariamente de las premisas (los enunciados anteriores). En el

ejemplo, "*a* es un Cynometra. Por tanto, *a* tiene las hojas verdes" no es tan fácil establecer la pretensión de que la conclusión se siga necesariamente de las premisas, pero sí que se siga probablemente, ya que, aunque la mayoría de los Cynometras tienen las hojas verdes podrían existir especies de Cynometras que no las tengan de ese color. Este tipo de razonamientos se denominan razonamientos inductivos. Un ejemplo más: "Estas jugando al fútbol. Por tanto, se te ha curado la gripe". Aquí el razonamiento ni es deductivo ni inductivo, sino abductivo. Se pretende que la conclusión sea una conjetura establecida a partir de lo que se dice en la premisa. Según el tipo de conexión que se pretenda establecer entre las premisas y la conclusión de un razonamiento tendremos un tipo u otro de ellos, y en consecuencia podremos representarlo con un tipo de Lógica distinto. Cada una de estas Lógicas tratará de proporcionar métodos y reglas que nos permitan discernir qué ejemplos de razonamiento son correctos y cuáles no lo son dentro de cada tipo.

En definitiva, el objetivo de la Lógica es diseñar un método general de prueba de razonamientos que sea efectivo y que responda de un modo *categórico* a la pregunta de si determinado razonamiento es correcto o no. A este modelo de razonamiento se le denomina *cálculo formal*. El hecho de que se califique al cálculo como *formal*, significa que debe poderse definir sin hacer referencia a cuestiones de índole semántica. Para caracterizar a un cálculo formal es necesario dar cuenta de:

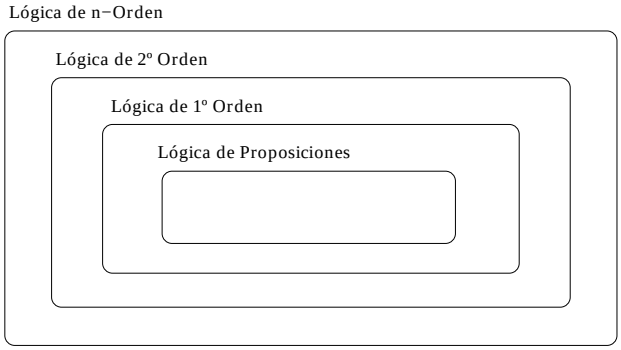
- *Un lenguaje formal*, que nos permite generar o reconocer *formulas bien formadas* y distinguirlas de las que no lo son. Para especificar un lenguaje formal se utiliza:
    - *Un vocabulario*, donde los elementos básicos vendrán dados por una *definición exhaustiva* que determine todos y cada uno de los mismos.
    - *Las reglas de formación*. Una vez conocido el vocabulario que se va a utilizar, éstas permitirán ordenar los elementos, y establecer cuáles son las combinaciones posibles entre éstos. En este sentido, han de constituir una fórmula bien formada, de modo que, ante cualquier combinación de elementos del vocabulario, se pueda determinar si la expresión resultante es o no de este tipo.
- *Un mecanismo deductivo*, que establece los criterios que permiten el paso de una fórmula bien formada a otras que también estén bien formadas. Si el cálculo formal es axiomático[8], para especificar su mecanismo deductivo se utilizan:
    - *Axiomas*, que son fórmulas bien formadas que se toman como representaciones de enunciados de partida indiscutibles.
    - *Las reglas de transformación*, que son las que nos indican de que manera podemos pasar de una fórmula bien formada a otra. Tendrán que asegurar una secuencia de pasos finitos, que permitan establecer si una transformación se ha realizado de forma correcta.

Históricamente, el tipo de razonamiento más estudiado es el razonamiento deductivo. Entenderemos por Lógica Clásica, la Lógica Deductiva a la que se le incorpora una semántica bivalente. Es decir, una semántica que considera a cada fórmula bien formada del cálculo como susceptible de ser verdadera o falsa sin que pueda existir ningún otro valor de verdad. El cálculo lógico no deja de ser una simple estructura sintáctica. La potencia y expresividad de la representación del razonamiento que nos proporcione determinada lógica dependerá de dos factores:

[8] Es conveniente señalar que existen presentaciones de cálculos lógicos que no son axiomáticas, en las que la definición del mecanismo deductivo se realiza de forma distinta a como aquí está siendo presentado. Por ejemplo, en los cálculos formales de deducción natural, el mecanismo deductivo se define indicando solamente reglas de transformación.

- *Sintáctico*. El número de expresiones consideradas como lógicas y tratadas como tales en el cálculo. Esto vendrá determinado por la sintaxis del cálculo y se revelará en el vocabulario primitivo del lenguaje formal del sistema.
- *Semántico*. El número posible de valores de verdad que se pueden atribuir a una fórmula bien formada del sistema. Esto se determinará en la semántica que, con frecuencia, se asocia al cálculo lógico.

Desde el punto de vista sintáctico, una posible clasificación de las distintas lógicas formales podría ser la que representa el siguiente diagrama de Venn en dónde, sobre un cálculo, se incorpora otro que contiene más recursos expresivos y que necesita de nuevos elementos, o incluso que evita restricciones del uso de estos recursos. Esta figura debe de interpretarse de una forma monótona ascendente, es decir, la expresividad de los sistemas crece en los sistemas de los niveles superiores, pero no en sentido contrario. Teniendo esto presente, la lógica se estructura de la siguiente manera:



- *Lógica de Proposiciones*. Es el cálculo básico de la lógica formal. En él, las fórmulas representan proposiciones. En este cálculo la deducción se pone de manifiesto en la relación de implicación que se da entre las premisas y la conclusión de un argumento (si ocurren las premisas, entonces necesariamente ocurrirá la conclusión, esto significa que es imposible que siendo las premisas verdaderas la conclusión sea falsa). Las variables son únicamente booleanas, y resulta difícil generalizar los razonamientos a no ser que se haga por enumeración de la totalidad de los casos individuales, lo que imposibilita su aplicabilidad a dominios de definición infinitos.
- *Lógica de Primer Orden*. Caracterizado por la introducción de los conceptos de variable y constante de individuo y porque permiten usar cuantificadores con elementos individuales. Además, permite expresar la pertenencia o posesión de propiedades por parte de los distintos individuos y también las relaciones entre ellos.
- *Lógicas de 2º (3º, 4º...n) Orden*. Usando como base la lógica de primer orden, podremos cuantificar sobre los predicados (propiedades o relaciones) obteniendo una lógica de segundo orden, o sobre los predicados de predicados obteniendo una lógica de tercer orden, y así sucesivamente.

A parte de esta clasificación que afecta fundamentalmente a la cuantificación, existen lógicas en cuyos lenguajes formales se consideran nuevas expresiones como poseedoras de carácter lógico. Algunos ejemplos de estas lógicas son:

- La *Lógica Modal*, que incorpora como operadores los modificadores relativos a lo que es necesario y lo que es posible. En este tipo de lógica se podrían expresar formalmente cosas como "Posiblemente los Cynometras que tienen dos pares de foliolos son Cynometras Sanagaensis".

- La *Lógica Temporal*, que incorpora parámetros temporales. Para muchas sentencias su verdad depende del momento en que se produce. Un ejemplo sería si se hablase del color de los pétalos de una flor en función de la época del año que se estuviera describiendo.

Desde el punto de vista semántico, se podrían clasificar las lógicas según el número de valores de verdad posibles para la interpretación de una fórmula bien formada del sistema. De esta manera tenemos:

- *Lógica Bivalente*. Considera que una fórmula bien formada del sistema es susceptible de ser verdadera o falsa sin que exista otra posibilidad.
- *Lógicas Multivaloradas*
    - *Lógicas finitamente valoradas.* Contemplan tres o más valores de verdad; lo verdadero, lo falso y otros valores intermedios considerados desconocidos o inciertos. Por ejemplo, el enunciado "la Cynometra Manii crece en entornos húmedos" puede ser verdadero, falso o incierto si la Cynometra Manii se da en entornos de humedad intermedia.
    - *Lógicas infinitamente valoradas.* En las que se consideran infinitos valores de verdad generalmente establecidos el intervalo [0, 1]. Si tomamos como ejemplo *"*un árbol x, cuya altura es de 5 metros*"* podría poseer un grado de pertenencia 0.6 para el valor "alto" y un grado de 0.4 para el conjunto "bajo", aunque también tendría un grado de pertenencia de 1 para el valor "mediano".
- *Lógica Borrosa*. Considera valores de verdad lingüísticos como "muy verdadero", "bastante verdadero", "poco verdadero", "poco falso", "bastante falso", "muy falso" etc. que se representan mediante el uso de números borrosos y a los que subyace toda una aritmética con este tipo de números.

2.1.- Lógica clásica

La lógica clásica aporta un buen número de ventajas para la representación del conocimiento y su manejo, partiendo de una sintaxis y una semántica bien definidas que detallan perfectamente la forma de construir sentencias y razonamientos sobre ellas. En este sentido, una de esas ventajas es que el conocimiento que se proporciona será siempre preciso, o lo que es lo mismo, susceptible de ser exclusivamente verdadero o falso. Existen varias notaciones, pero nos centraremos en las dos más populares. Se mostrarán la *Lógica Proposicional* y la *Lógica de Primer Orden*.

2.1.1.- Lógica Proposicional

La *Lógica Proposicional* (LP) es el sistema lógico mejor conocido, pero también la más simple de las formas de lógica. Toma una representación primitiva del lenguaje mediante un vocabulario básico que incluye variables sólo de tipo booleano, y que permite representar y manipular aserciones sobre el mundo que nos rodea. Estas aserciones son llamadas *proposiciones* y corresponden a una oración enunciativa que puede ser verdadera o falsa. Permite el razonamiento a través de un mecanismo que primero evalúa sentencias simples y luego complejas, formadas mediante el uso de *símbolos lógicos* o *conectivas*, como son NO ($\neg$), Y ($\wedge$), O ($\vee$), ENTONCES ($\rightarrow$). Pero además de estos operadores lógicos, también se incluyen dos signos de puntuación: los

paréntesis izquierdo y derecho, que sirven para variar las precedencias de las conectivas anteriores, tal como se muestra en la siguiente tabla:

$\neg$	Negación	Función lógica NO	$\neg\, p$
$\wedge$	Conjunción	Función lógica Y	$p \wedge q$
$\vee$	Disyunción	Función lógica O	$p \vee q$
$\rightarrow$	Condicional	Función condicional	$(p \vee q) \rightarrow r$
()	Asociación	Jerarquía de operadores	$((p \vee q) \wedge r)$

Las conectivas combinan los símbolos lógicos siguiendo una sintaxis que pasamos a definir.

Dado un conjunto P de variables proposicionales, y siendo φ y ψ fórmulas bien formadas cualesquiera, se define la sintaxis de la Lógica Proposicional sobre P, denotado por L(P), como el menor conjunto que satisface las siguientes reglas:

- *$p \in P$, entonces $p \in L(P)$.*
- *Si $\varphi \in L(P)$, entonces $(\neg\varphi) \in L(P)$.*
- *Si φ, $\psi \in L(P)$, entonces:$(\varphi \vee \psi) \in L(P)$, $(\varphi \wedge \psi) \in L(P)$ y $(\varphi \rightarrow \psi) \in L(P)$.*

Una vez que conocemos el vocabulario básico, las variables proposicionales y las conectivas; estas reglas permitirán determinar si una proposición cualquiera pertenece o no a nuestro cálculo, es decir, si es una fórmula bien formada. El lenguaje obtenido a partir de *L*(*P*) podría interpretarse como un intento de formalizar un pequeño fragmento del lenguaje natural, a saber, sus estructuras conectivas. A continuación introducimos desde un punto de vista intuitivo estos conectores:

- *Negación*. Corresponde a las expresiones "no", "no es cierto que", etc. del lenguaje natural. En un primer análisis elemental no hay dificultades entre el sentido lógico formal y el que conlleva su uso en un lenguaje natural. Su dificultad estriba en su implementación computacional a menudo sujeta a criterios de eficacia computacional, lo que a menudo lleva a interpretaciones como la negación por fallos, difícilmente adaptables a un uso idiomático.
- *Conjunción*. Corresponde a las expresiones "y", "pero", etc. del lenguaje natural. En la conjunción de oraciones no existen grandes diferencias en relación a los lenguajes naturales a la hora de abordar su formalización, pero si deben de notarse un par de problemas que es necesario tener en cuenta. Consideremos los casos siguientes:
  - El problema de la reflexividad: El concepto "*Luisa y Daniel tienen los ojos azules*" puede formalizarse como:

    ("*Luisa tiene los ojos azules*" $\wedge$ "*Daniel tiene los ojos azules*")

    pero la frase "*Luisa y Daniel se miran en el espejo*" no está tan claro que se deba formalizar como:

    ("*Luisa se mira en el espejo*" $\wedge$ "*Daniel se mira en el espejo*")

    ya que la primera podría indicar que ambos se están mirando el uno al otro en el espejo, en cambio la segunda no lo implica. El problema radica en la ambigüedad de la oración.
  - El problema de la temporalidad: En los lenguajes naturales, la conjunción implica secuencia temporal. Es el caso de:

    "*Luisa abre el grifo y se moja*" no es lo mismo que "*Luisa se moja y abre el grifo*"

sin embargo, este sentido se pierde en la formalización. En la descripción de la especie *Cynometra* correspondiente ejemplo descrito con anterioridad, aparentemente no existen problemas ni de temporalidad ni de reflexividad en los textos.

- *Disyunción*. Corresponde a las expresiones "o", "o bien", etc. del lenguaje natural. Por regla general, en los lenguajes naturales se suele entender en sentido exclusivo. En cambio en LP, será verdadero cuando una o ambas lo sean. Consideremos la frase "*las hojas son paripinnadas o imparipinnadas*" que podríamos formalizar mediante:

  "*Las hojas son paripinnadas"* $\vee$ "*Las hojas son imparipinnadas*"

  según esta formalización ambas podrían ser verdaderas, cuando lo que en realidad se quiere expresar es o una u otra pero no ambas.
- *Condicional*. Podríamos identificarlo con el conectivo idiomático *"si..., entonces..."*. En el caso de la LP la verdad o falsedad de $p \rightarrow q$ depende exclusivamente de la verdad o falsedad de cada proposición por separado.

Brevemente descrita la sintaxis, queda todavía un aspecto de la LP por analizar: su semántica. La interpretación o significado de una proposición es lo que se llama su *valor de verdad*. Las proposiciones que interesan a la LP son siempre proposiciones asertivas, y como es una lógica bivalente, sus proposiciones serán siempre *verdaderas* o *falsas*. Es decir, una proposición formalizada por la variable *p* podrá tener uno de esos dos valores. En cambio, el valor de verdad de una fórmula compuesta vendrá determinado por el valor de verdad de cada una de las variables proposicionales que en ella intervienen en función de las conectivas que contenga.

En general, dado un número *n* de variables proposicionales, el número de combinaciones posibles de sus valores de verdad sería 2^n. Así, para n=2, sus combinaciones posibles serán de 4, para n=3, 8, etc. La forma más simple de determinar los valores de verdad para fórmulas complejas es la de las llamadas *tablas de verdad*. El procedimiento se muestra a continuación:

p	q	$\neg$ p	(p $\wedge$ q)	(p $\vee$ q)	(p $\rightarrow$ q)
Verdadero	Verdadero	Falso	Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso	Falso	Verdadero	Falso
Falso	Verdadero	Verdadero	Falso	Verdadero	Verdadero
Falso	Falso	Verdadero	Falso	Falso	Verdadero

En este sentido, el resultado de la evaluación de una fórmula depende del valor de verdad de las variables proposicionales que la forman. Sin embargo, existen ciertas expresiones que siempre producirán el mismo resultado independientemente del valor de verdad de las variables que la componen. Cuando este resultado es siempre "verdadero" para cualquier combinación de variables proposicionales, a la fórmula en cuestión se le denomina *tautología.*

Una tautología *es una fórmula bien formada de un sistema lógico que resulta verdadera para cualquier interpretación; es decir, para cualquier asignación de valores de verdad que se haga a sus fórmulas atómicas.*

Para mostrarlo, vamos a evaluar la fórmula $(p \wedge q) \rightarrow p$ para todos los posibles valores de sus variables proposicionales. Para ello basta con construir su tabla de verdad, tal y como se muestra a continuación:

p	q	$(p \wedge q)$	$(p \wedge q) \rightarrow p$
Verdadero	Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso	Verdadero
Falso	Verdadero	Falso	Verdadero
Falso	Falso	Falso	Verdadero

A partir de ésta tabla, es fácilmente verificable la verdad de la proposición cualquiera que sea el estado en el que sea evaluada, siempre y cuando nuestra proposición esté bien definida.

El estudio de la LP es relativamente sencillo, sin embargo, existe una multitud de estructuras deductivas que la LP no puede formalizar de forma adecuada. Es el caso por ejemplo de la frase:

"Todos los Cynometras son árboles, el Cynometra Sanagaensis es un Cynometra, luego el Cynometra Sanagaensis es un árbol"

Algo que justifica en sí la consideración de un formalismo más potente, la lógica de primer orden, que pasamos a comentar con más detalle.

2.1.2.- Lógica de Primer Orden

Las sentencias complejas pierden mucho de su significado cuando son representadas en Lógica Proposicional. Tomando el ejemplo:

"Todos los Cynometras Sanagaensis son Cynometras, Todos los Cynometras son árboles, luego todos los Cynometras Sanagaensis son árboles",

Para captar la forma lógica de estos argumentos es necesario un análisis formal más exhaustivo que el proporcionado por la LP. Así, en el caso anterior, una de las claves está en la ocurrencia de la palabra "todo". Este término, junto con la palabra "alguno", dejan atrás el ámbito de la LP, introduce los cuantificadores y variables en nuestros cálculos y sirve de base a la *Lógica de Primer Orden* (LPO). Más formalmente, introducimos ahora los conceptos fundamentales de LPO.

El *alfabeto* de la Lógica de Primer Orden está formado por los siguientes conjuntos de símbolos:

- *Términos de individuo:*
  - *Constantes de individuo*: Representan objetos concretos. Las constantes son individuos o elementos distinguidos del universo del discurso.
  - *Variables de individuo*: Sirven para representar objetos, cuyo dominio hay que especificar, es decir, se pueden tener objetos desconocidos.
- *Predicados:* Son objetos estructurados, definidos por un *funtor* que le da nombre, y una *aridad* que designa el número de argumentos asociados al funtor. El par funtor/aridad se denomina *firma* y refleja la estructura arborescente del predicado. Un predicado asocia una interpretación lógica que se expresa en relación a otros predicados. Así, por ejemplo, P^3 designa un funtor P con tres argumentos.
- *Conectivas:* Idéntico al de la Lógica Proposicional, es decir, la negación, la conjunción, la disyunción y el condicional.
- *Cuantificadores:* El cuantificador universal, $\forall$ ("para todo"), y el existencial, $\exists$ ("existe un").

Los símbolos anteriores conforman el alfabeto del lenguaje que, combinados de forma adecuada, permitirán crear las fórmulas que constituyen la base expresiva de la LPO. Se introducen nuevos elementos del lenguaje para permitir expresar propiedades,

y atribuir éstas a individuos. Esta nueva lógica permite una descripción más fina de la realidad, pudiendo distinguir los conceptos de sus propiedades. Tomando como base el ejemplo anterior, un objeto sería un árbol concreto y una propiedad sería la de *ser un Cynometra*. La LP, cuyos elementos básicos son las proposiciones, no permite realizar esta distinción.

> *Dado un alfabeto de Lógica de Primer Orden, se define una* fórmula bien formada *como una expresión construida recursivamente mediante la aplicación de alguna de estas reglas:*
>
> - *Todo predicado P n-ario seguido de un conjunto $t_1, ..., t_n$ de términos de individuo es una fórmula bien formada, y se denomina fórmula atómica.*
> - *Si φ es una fórmula bien formada, entonces $\neg\varphi$ también lo es.*
> - *Si φ y ψ son fórmulas bien formada, entonces $(\varphi\vee\psi)$, $(\varphi\wedge\psi)$ y $(\varphi\rightarrow\psi)$ también lo son.*
> - *Si φ es una fórmula bien formada y x es una variable, entonces $\forall x(\varphi)$ y $\exists x(\varphi)$ también son fórmulas.*

Un ejemplo usando un cuantificador universal se aplicaría en la frase "Todos los Cynometras son árboles" pudiendo formalizarse empleando los predicados:

Cynometra(x)="x es Cynometra" y Árbol(x)="x es árbol"

como:

$$\forall x(Cynometra(x)\rightarrow Árbol(x))$$

donde *x* es el término, *Cynometra(x)* y *Árbol(x)* son fórmulas atómicas y (*Cynometra(x)→Árbol(x)*) también es una fórmula.

Un ejemplo usando un cuantificador existencial se aplicaría en la frase "Algunos Caesalpinioideaes son pinnados" pudiendo formalizarse empleando los predicados:

Caesalpinioideae(x)="x es Caesalpinioideae" y Pinnado(x)= "x es pinnado"

como:

$$\exists x(Caesalpinioideae(x)\wedge Pinnado(x))$$

donde *x* es el término, *Caesalpinioideae(x)* y *Pinnado(x)* son fórmulas atómicas y (*Caesalpinioideae(x) ∧ Pinnado(x)*) también es una fórmula.

La representación mediante LPO presenta, sin embargo, limitaciones importantes. Así, la limitación de la cuantificación de las variables impide, por ejemplo, plantear cuestiones como el *principio de identidad de indiscernibles*, que expresa que $\forall P\,\forall x\,\forall y((Px\leftrightarrow Py)\leftrightarrow x = y)$ y exigen la cuantificación sobre predicados. En el ejemplo anterior aparece un cuantificador aplicado a un funtor de un predicado. Ya hemos hecho referencia anteriormente a la lógica que permite ese tipo de cuantificaciones, la habíamos denominado Lógica de Predicados Segundo Orden.

2.2.- Sistemas Basados en Reglas

La secuencialidad, concepto muy ligado a al paradigma de programación imperativo que asumió el protagonismo absoluto en los orígenes de la informática, se comenzó a cuestionar tan pronto como se dieron los primeros pasos en el campo de la Inteligencia Artificial. Este procedimiento, que consiste en la ejecución paso a paso y siempre en el mismo orden de las instrucciones de un programa, no resulta adecuado en situaciones en las que el entorno es cambiante. En este sentido, puede ser útil dividir el programa en una serie de módulos, cada uno de los cuales ha de encargarse de tratar específicamente alguno de estos cambios. El objetivo es utilizar los datos como la parte del programa que dirige las operaciones, una idea que fructifica en la concepción de los

Sistemas Basados en Reglas de Producción [Davis, R.; King, J. J. (1984)], uno de los modelos de representación del conocimiento más ampliamente utilizados. Su simplicidad y aparente similitud con el razonamiento humano, han contribuido a su popularidad en diferentes dominios. Formalmente, las reglas de producción tienen la forma:

Si <premisas> **entonces** <conclusiones, acciones>

siguiendo un acercamiento inferencial; donde las premisas no son más que precondiciones y antecedentes que son distintivos, es decir, una cadena de hechos unidos por conectivas (Y, O, NO), de la siguiente forma:

Si <hecho1> Y/O <hecho2> Y/O … <hechoN> **entonces** <hechoB1> Y/O <hechoB2> Y/O … <hechoBN>

En esencia, una *regla* es una combinación de hechos que permite representar conocimientos y derivar conclusiones. La representación de los hechos debe de corresponderse con el conocimiento del dominio. Al conjunto de hechos que describen el problema se le denomina *base de hechos o memoria de trabajo*, y al conjunto de reglas que describen el problema se le conoce como *base de reglas*.

Definida la sintaxis del lenguaje, y ya desde un punto de vista semántico, para la interpretación de las conclusiones/acciones, existen dos aproximaciones principales [Vianu, V. (1997)]. Por un lado, la conclusión representa una declaración de un hecho a incorporar a la memoria de trabajo, siguiendo el paradigma que proporciona la semántica declarativa. Por otro, una acción representa cualquier actuación procedimental, es decir, una aplicación que, de hecho, se ha de ejecutar, usando el mismo principio del paradigma que proporciona la semántica procedimental, empleada en las *bases de datos activas*[9]. Generalmente, la acción puede llevar a la activación de otra condición. Vamos a ilustrar esto con los siguientes ejemplos:

Siguiendo la descripción de los géneros que proceden de la especie Cynometra, se van a definir una serie de hechos:

- El Cynometra tiene un par de foliolos.
- El Cynometra tiene una glándula basilar en la cara interna del limbo.
- El Cynometra tiene dos pares de foliolos.
- El Cynometra tiene frutos arqueados.

y las reglas siguientes, en una aproximación declarativa:

- *R1*: **SI** (el Cynometra tiene un par de foliolos) **Y** (el Cynometra tiene una glándula basilar en la cara interna del limbo) **ENTONCES** (el género es Cynometra Le-Testui).
- *R2:* **SI** (el Cynometra tiene frutos arqueados) **Y NO**(el Cynometra tiene dos pares de foliolos) **ENTONCES** (el género es Cynometra Vogelii).

Se observa como partiendo de hechos conocidos que describen algún conocimiento se pueden inferir otros, así como nuevos conocimientos, usando un lenguaje declarativo para su especificación. Por otra parte la regla 2, no tiene porque ser

[9] Una base de datos activa es un sistema de bases de datos que maneja la vigilancia de condiciones (con disparadores y alertas). En este sentido, vigila condiciones disparadas por sucesos que representan acciones, es decir, la evaluación de la condición resulta verdadera, se ejecuta la acción, ofreciendo modularidad y respuesta oportuna en la acción [Elmasri, R.; Navathe, S. B. (1997)].

totalmente cierta, existe la posibilidad de que "el Cynometra tenga 3 pares de foliolos", con lo cual no se puede hacer esta afirmación con toda certeza.

A partir de los conceptos expuestos en esta sección, se puede introducir ya formalmente el concepto de Sistema Basado en Reglas de Producción. Un sistema basado en reglas de producción se define mediante una tripla $R=(B_h, B_r, M_i)$, donde*:*

- B_h *es la base de hechos.*
- B_r *es la base de reglas.*
- M_i *es un motor de inferencia que emplea una estrategia de control.*

Durante la ejecución de un Sistema Basado en Reglas de Producción, como resultado del proceso de resolución del modelado ante una instancia del problema, se añaden o eliminan hechos de la *memoria de trabajo* (frecuentemente, se distingue la base de hechos de la memoria de trabajo diciendo que ésta representa el conocimiento temporal que se tiene del problema hasta el momento). La semántica de los Sistemas Basados en Reglas de Producción implica que las reglas que forman el programa se ejecutan y disparan en paralelo hasta que se alcanza su punto fijo. El proceso de inferencia es la operación por la que se obtendrá conocimiento nuevo a partir del existente y para ello se utiliza uno de estos métodos de razonamiento:

- *Deducción*. Usando reglas de inferencia de la lógica, como la resolución.
- *Inducción*. Mediante generalización de observaciones para sintetizar conocimiento de más alto nivel.
- *Abducción*. Método de razonamiento por explicación posible.

Por regla general, cuando se utiliza el método de razonamiento deductivo se hace uso del *modus ponens*[10] para manipular los hechos y las reglas. Mediante técnicas de búsqueda y procesos de unificación, los Sistemas Basados en Reglas de Producción automatizan sus métodos de razonamiento. Esta progresión hace que se vayan conociendo nuevos hechos, a medida que nos acercamos a la solución del problema. Los Sistemas Basados en Reglas de Producción difieren de la representación basada en lógica en aspectos fundamentales:

- Son en general no-monotónicos, es decir, hay hechos derivados que pueden ser retractados en el momento en que dejen de ser verdaderos.
- Pueden aceptar incertidumbre en el proceso de razonamiento, tal como se vio en el ejemplo anterior.

Definimos la *base de conocimiento* como formada por la base de hechos y la base de reglas. Pero es necesario un sistema capaz de hacerlos funcionar. Ésta es la principal misión de la *estrategia de control*. Esencialmente existen dos tipos de procedimientos para inferir nuevo conocimiento a partir de un conjunto de reglas de producción decidiendo el orden en que se seleccionan y disparan esas reglas. Así tenemos:

- *Razonamiento hacia adelante*. Una regla es activada si su antecedente empareja con alguno de los hechos del sistema. En terminología anglosajona, *forward chaining*. En el caso de razonamiento hacia delante, se disparan todas las reglas posibles, en base a que las premisas sean verdaderas hasta llegar a la conclusión. Este modo de razonamiento deriva un conocimiento máximo, ya que a partir de datos iniciales encontrará todas las posibles conclusiones a las que se pueda llegar. Este procedimiento puede ser implementado de muchas formas. Una de ellas comienza con las reglas cuyas premisas tienen valores conocidos. Estas reglas deben concluir y sus conclusiones dan lugar a nuevos hechos. Estos

[10] En lógica, el *modus ponens* es una regla de inferencia que tiene la siguiente forma: *Si A*, *entonces B y A; por lo tanto, B*.

nuevos hechos se añaden al conjunto de hechos conocidos, y el proceso continúa hasta que no pueden obtenerse nuevos hechos. El algoritmo sería el siguiente:

```
Algoritmo:
Procedure Encadenamiento_hacia_adelante;
 begin
  //BH es la base de hechos,
  //CC el conjunto conflicto,
  //MT es la memoria de trabajo,
  //Meta es el objetivo que se busca
  //BC es la base de conocimiento
  MT → CargarHechosIniciales(BH);
  CC → ExtraeCualquierRegla(BC);
  while NoContenida(Meta, MT) and NoVacio(CC) do begin
        CC → Equiparar(Antecedentes(BC), MT)
        if NoVacio(CC) then begin
              R = Resolver(CC);
              NuevosHechos = Aplicar(R, MT);
              Actualizar(MT, NuevosHechos);
        end if
  end while
  if Contenida(Meta, MT) then begin
        return "exito";
  end if
end
```

Ilustraremos este proceso con el siguiente ejemplo. Supongamos que nuestra base de hechos inicial consta de los siguientes elementos: Diámetro = 5 cm., Forma = oval, tipo semilla = múltiple, y Color = amarillo. Usando la siguiente base de conocimiento, vamos a buscar, partiendo de estos hechos, cuál es la fruta que se ajusta a estas condiciones:

Regla 1 IF nº semillas=1 THEN tipo semilla=hueso	Regla 2 IF nº semillas>1 THEN tipo semilla=multiple	Regla 3 IF Forma=oval AND Color=marron AND tipo frutal=arbol THEN Fruta=kiwi
Regla 4 IF Forma=oval AND color=violeta AND tipo frutal=arbol AND tipo semilla=hueso THEN Fruto=ciruela	Regla 5 IF Forma=oval AND color=amarillo AND tipo frutal=arbol THEN Fruto=limon	Regla 6 IF Forma=redonda AND color=naranja AND tipo frutal=arbol AND tipo semilla=hueso THEN Fruto=melocoton
Regla 7 IF color=amarillo AND tipo frutal=emparrado AND THEN Fruta=melon	Regla 8 IF Forma=oval o redonda AND diametro > 10cm THEN tipo frutal=emparrado	Regla 9 IF Forma=redonda u oval AND AND diametro < 10cm THEN tipo frutal=arbol

Aplicando el algoritmo, y usando una estrategia de ejecutar la primera regla de la BC que se ajusta a los requisitos, se puede ver como:

Ciclo	Conjunto conflicto	Regla seleccionada	Hecho derivado
1	2, 9	2	tipo semilla=multiple
2	2, 9	9	tipo frutal=arbol
3	5	5	fruta=limon

Es necesario tener en cuenta que determinados consecuentes de reglas pueden llevar consigo la ejecución de acciones que pueden modificar algún elemento de la memoria de trabajo, lo que provoca que se vuelvan a comprobar los antecedentes de las reglas para así activarlas. Si no se modifica ningún elemento de la memoria de trabajo, se selecciona otra de las instancias de las reglas presentes en el conjunto conflicto. La ejecución del Sistema Basado en Reglas de Producción finaliza cuando no queda ninguna instancia de regla almacenada. En este sentido, se dice que un Sistema Basado en Reglas de Producción alcanza su punto fijo cuando ninguna de las reglas está activa, o únicamente hay activas reglas cuya conclusión no modifica la memoria de trabajo.

- *Razonamiento hacia atrás*. Una regla es activada si su consecuente empareja con alguno de los hechos del sistema. En terminología anglosajona, *backward chaining*. Cuando se habla de razonamiento hacia atrás se hace referencia sólo al proceso de búsqueda y a la selección de las reglas, partiendo de las conclusiones. Este tipo de razonamiento es muy útil cuando los datos iniciales son muy numerosos y sólo una pequeña parte de ellos es relevante. El algoritmo de encadenamiento hacia atrás requiere que el usuario seleccione, en primer lugar, un objetivo; entonces el algoritmo navega a través de las reglas en búsqueda de una conclusión para él. Si no se obtiene ninguna conclusión con la información existente, entonces el algoritmo fuerza a preguntar al usuario en busca de nueva información sobre los elementos que son relevantes para obtener información sobre el objetivo. Realmente lo que se realiza es lo siguiente:
    1. Se forma una pila inicial compuesta por todos los consecuentes que satisfacen el objetivo inicial
    2. Se considera el primer elemento de la pila y se localizan todas las reglas que lo satisfagan.
    3. Se examinan las premisas de dichas reglas, en orden:
        - Si todas las premisas se satisfacen, entonces se ejecutan las reglas y se derivan sus conclusiones. Si se derivó un valor para el objetivo actual entonces se elimina de la pila y se vuelve al paso 2.
        - Si la premisa de una regla no se satisface, porque tiene un valor desconocido en la base de conocimiento, se mira si existen reglas que concluyan un valor para ella. Si existen se inserta en el tope de la pila de objetivos y se vuelve al paso 2.
        - Si en el paso anterior no se encontró ninguna regla que concluya un valor para la premisa actual, entonces se pregunta al usuario por dicho valor y se añade a la base de conocimiento.
            - Si el valor satisface la premisa actual se continúa examinado el resto del antecedente.
            - Sino se considera la siguiente regla que concluya un valor para el objetivo actual
    4. Si se han examinado todas las reglas que concluyen un valor para el objetivo actual y todas fallaron entonces se marca el objetivo como

indeterminado, se extrae de la pila y se vuelve al paso dos. Si la pila esta vacía el proceso finaliza.

Una propuesta en pseudocódigo podría ser la siguiente:

```
Procedure Encadenamiento_hacia_atrás
Begin
 //BH es la base de hechos,
 //MT es la memoria de trabajo,
 //Meta es el objetivo que se busca
 //BC es la base de conocimiento
 //BR es la base de reglas
 pila → Construir_pila_inicial(Equiparar(Consecuentes(BC), Meta));
 MT → CargarHechosIniciales(BH);
 while NoVacio(pila) do begin
      BR → pop(pila);
      AntecedentesReglas [] → ExtraerAntecedentes(BR);
      Verificado → 0;
      while NoVacio(AntecedentesReglas) and (Verificado!=1) do begin
            //puede tener 3 valores: 0 =true 1=false 2=no sabe
            Verificado → VerificarPremisa(AntecedentessReglas[i],MT);
            if (Verificado!=1) then do begin
                  NuevasReglas → Derivar(AntecedentessReglas[i],BC);
                  if(NoVacio(NuevasReglas)) then
                                          push(pila,NuevasReglas);
            else do begin
              Bool:conocido → Preguntar_Usuario();
              if(conocido) then Añadir_MT();
           end else
        end if
    end while
      if( esVacio(AntecedentesReglas) and Verificado==0) then do begin
            Devolver(exito);
      end if
 end while
 Devolver (falso);
end procedure
```

Para ejemplificar esto supongamos que nuestro objetivo es “fruta” y que nuestra base de hechos inicial consta de los siguientes elementos: Forma = oval, tipo semilla= múltiple y Color = amarillo. Usando la misma base de conocimiento que la utilizada para el ejemplo anterior, vamos a buscar, partiendo de estos hechos, cual es la fruta que se ajusta a estas condiciones:

- Las reglas que se van a tener en cuenta son inicialmente la 3, 4, 5, 6, 7.
- Por lo que partiendo de la regla 3, se deriva que no cumple la premisa de color por lo que se descarta.
- Tomando la regla 4, no se cumple la segunda premisa.
- Tomando la regla 5, las dos primeras premisas se cumplen. Pero tipo frutal=árbol, por lo que debe añadir a la lista de reglas la 9. Queda entonces 9, 5, 6, 7.

- Tomando la regla 9, la primera premisa se cumple pero no existe en la BH ninguna regla para la dimensión por lo tanto es necesario preguntar al usuario cual es la dimensión del fruto.
- La respuesta es de 5cm por lo que se añade a la BC, se dispara la regla 9 y se añade a la base de hechos tipo frutal = árbol.
- Ahora con tipo frutal = árbol, Dimensión=5 cm., Forma = oval, tipo semilla = múltiple, y Color = amarillo se dispara la regla 5 concluyendo que la fruta es un limón.

Ciclo	Conjunto conflicto	Regla seleccionada	Hecho derivado
1	3,4,5,6,7	3	–
2	4,5,6,7	4	–
3	5,6,7	5	–
4	9,5,6,7	9	Pregunta? Dimension=5cm, tipo frutal=arbol
5	5,6,7	5	fruta=limon

Algunas de las desventajas existentes en los Sistemas Basados en Reglas de Producción son la ineficiencia en tiempo de desarrollo, que conlleva la necesidad de modularizar o de introducir metarreglas, es decir, reglas que incorporan conocimiento acerca de las reglas de producción. Las metarreglas facilitan y aceleran la búsqueda de soluciones, ya que dan una indicación sobre cómo razonar en una situación dada y, por tanto, proporciona un consejo útil a la estrategia de selección de reglas. Otra desventaja es la dificultad de establecer las relaciones entre hechos que explicitan las reglas. Finalmente, otro problema es la adaptación al dominio que tiene como consecuencia un rápido crecimiento del número de reglas.

Sin embargo, a pesar de las desventajas anotadas, los sistemas basados en reglas han permanecido como los esquemas más comúnmente utilizados para la representación del conocimiento. Como ventajas significativas se pueden mencionar las siguientes: modularidad, uniformidad y naturalidad para expresar el conocimiento. Concretamente, las reglas resultan adecuadas para la representación formal de recomendaciones, órdenes o estrategias.

2.3.- Representaciones estructuradas

La lógica, aunque constituye una buena representación del conocimiento, no aporta mucho cuando tenemos que describir la estructura compleja del mundo y tenemos que escoger un diseño de implantación. Lo mismo ocurre con respecto a las limitaciones de las representaciones basadas en reglas de producción. En general, la complejidad del problema puede hacer que el enfoque mediante reglas resulte difícilmente abordable, fundamentalmente por falta de legibilidad y por el tipo de relaciones a modelizar. En concreto, resulta a menudo útil representar aspectos como estructuras y relaciones que permiten agrupar las propiedades de los objetos del mundo en unidades de descripción. Esto permite al sistema focalizar su atención en un objeto completo, sin considerar el resto de hechos que conoce, lo cual es importante para evitar la explosión combinatoria que supone explorar la totalidad del espacio de cálculo. En este sentido, las representaciones estructuradas de conocimiento tienen una gran potencia expresiva y permiten una fácil interpretación del mismo. Entre las

representaciones estructuradas más populares podemos considerar las *redes semánticas* y los *marcos*. Ambas se basan en el uso de grafos, estableciendo relaciones entre conceptos en el primer caso, y estructuras de jerarquía en el segundo. En cualquier caso, el conocimiento expresado mediante una de estas representaciones estructuradas puede ser traducido a la Lógica de Primer Orden.

2.3.1.- Redes semánticas

En su sentido más amplio, una red se compone de un conjunto de nodos unidos entre sí por cierto tipo de enlace. En este sentido, existen numerosos tipos de redes, de naturaleza muy diferente. En el caso que nos ocupa, se trata de un modelo teórico llamado *redes asociativas* donde cada nodo representa un concepto, o incluso una proposición, y los enlaces se corresponden a las relaciones que se establecen entre estos conceptos. Estas relaciones pueden referirse a causalidad, pertenencia, inclusión pero también a categorías gramaticales, como son un sujeto o un objeto. Concretamente, las redes asociativas destinadas a comprender el lenguaje natural, se conocen como *redes semánticas*. Actualmente, las redes asociativas sirven para representar, además de las reglas semánticas, asociaciones físicas o causales entre objetos.

Una red semántica no es más que una notación gráfica para representar el conocimiento existente, diferenciando los conceptos y las relaciones y mostrando patrones de interconexiones entre ellos. En este sentido, el concepto de red se podría considerar como sinónimo de grafo, aunque éste último conlleva una interpretación matemática precisa, mientras que el sentido de red es más amplio. Asumiendo esta sinonimia entre ambos términos, los *conceptos* se representarían en los grafos mediante *nodos* y a las *relaciones* mediante *arcos*. Por lo tanto, si los nodos son la representación intensional de los conceptos que pueden existir en un mundo real, los arcos se corresponden con las funciones y las diferentes relaciones de las realizaciones en lenguaje natural. Concretamente, las implementaciones que se han desarrollado de este tipo de redes en informática, fueron desarrolladas para la Inteligencia Artificial y la traducción automática [Shapiro, S. C.; Eckroth, D. (1987)], inicialmente con el propósito de representar, en términos de objetos y relaciones, el significado de frases en inglés.

Usando los conceptos expuestos, se puede introducir formalmente el concepto de *red semántica* de la siguiente manera:

> *Dado un conjunto de términos* $(t_1, t_2, \ldots, t_m)$ *ligados por cierta relación semántica, una* red semántica *es un grafo* $G = (N, A)$ *que cumple las condiciones siguientes:*
>
> - *El conjunto N es el de nodos del grafo. Estará formado por m elementos, es decir, tantos nodos como términos relacionables existan, y se llamarán* $t_1, t_2, \ldots, t_m$.
> - *El conjunto A es el de arcos del grafo. Dados dos nodos o términos del grafo* t_i *y* t_j, *existirá un arco* $a_{i,j}$ *que une el nodo* t_i *y* t_j *si y sólo si los términos* t_i *y* t_j *están relacionados.*

Una *red semántica* es una estructura de representación del conocimiento lingüístico, donde a las relaciones entre los diversos elementos semánticos se les da un aspecto de grafos cuyos nodos pueden representar objetos, entidades, atributos, eventos o estados; y los arcos representan las relaciones existentes entre ellos. Concretamente, las redes semánticas pueden agruparse en dos tipos:

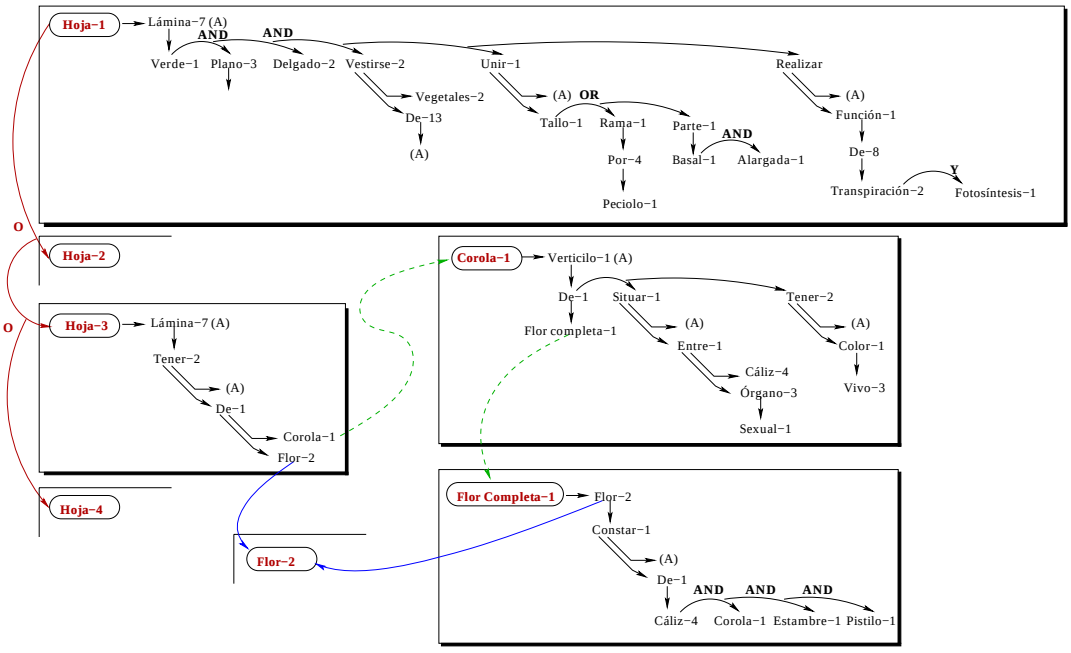
- *Sistemas asertivos*. Consiste en realizar afirmaciones particulares. En ellas no existen definiciones de conceptos ni clasificaciones jerárquicas, sino solamente afirmaciones concretas. Son sistemas que no excluyen la posibilidad lógica de una contradicción. Para ello se requiere formalizar las relaciones mediante etiquetas para poder representar conocimiento declarativo. En este tipo de sistemas se pueden incluir, entre otros, los denominados *modelos de memoria semántica* o *grafos relacionales* [Quillian, M. R. (1967)], y los *grafos de dependencia conceptual de Schank* [Schank, R. C. (1975), Schank, R. C.; Kolodner, J. L.; DeJong, G. (1980)].
- *Sistemas taxonómicos*. Son sistemas que permiten relacionar los conceptos mediante jerarquías. Los tipos de relaciones que incluyen serán relaciones de instancias y entre conjuntos y subconjuntos, es decir, de subclase, incluyendo relaciones de pertenencia y de propiedades, la denominada *jerarquía de conceptos* [Brachman, R. J. (1983)].

Para explicar con más claridad lo expuesto, nos centraremos en los siguientes tres casos que acabamos de relatar: los modelos de memoria semántica de Quillian, los grafos de dependencia conceptual de Schank y la jerarquía de conceptos.

2.3.1.1.- Modelos de memoria semántica o grafos relacionales de Quillian

El primer modelo de representación formalizado en el ámbito de las redes semánticas fue desarrollado por Quillian [Quillian, M. R. (1967)], que basándose en los trabajos de Selz [Simon, H. A. (1981)], trató de construir "un modelo computacional cuya fundamentación se basaba en la mente humana" con el fin de llegar a comprender el lenguaje natural. El modelo desarrollado, consistía en representar el significado de los términos de modo parecido a como lo hacen los diccionarios. En este sentido, esta representación consta de un conjunto de enlaces que unen entre sí el conjunto de términos, de ahí que se le conozca como *grafo relacional*.

Tomando como ejemplo la siguiente figura:



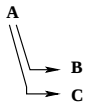
Vamos a ilustrar este tipo de red semántica, en el que se muestran dos de los diferentes sentidos de la palabra *hoja*, así como el de la palabra *corola*. Para ello, vamos a suponer las siguientes definiciones extraídas del diccionario de la Real Academia Española:

- *Hoja-1*: Cada una de las láminas verdes, planas y delgadas, de que se visten los vegetales, unidas al tallo o a las ramas por el peciolo o, por una parte basal alargada, en las que se realizan las funciones de transpiración y fotosíntesis.
- *Hoja-3*: Cada una de las láminas que tiene la corola de una flor.
- *Corola-1*: Verticilo de las flores completas, situado entre el cáliz y los órganos sexuales, y que tiene vivos colores

El conocimiento se organiza en *planos*, donde cada uno representa el grafo asociado a la acepción de una palabra. Además, se observa que los nodos encerrados en óvalos corresponden a los encabezamientos de las definiciones, es decir, *hoja* seguido de la acepción que ocupa en el diccionario, que puede ser *1, 2, ...* A estos nodos se les denomina *nodos-tipo*. Por ejemplo *Hoja-1* hace referencia a la primera definición. De este modo se evitan ambigüedades en las definiciones, pues, por ejemplo *Hoja-1* hace referencia a la hoja de los vegetales y *Hoja-5* a la de los libros y cuadernos. En este sentido, las palabras que aparecen en la propia definición, se les denomina *nodos-réplica* y estos a su vez serán nodos-tipo de su propia definición. Si observamos la definición de *Hoja-3*, éste posee un nodo-réplica *Corola-1* que a su vez es un nodo-tipo.

Una vez definidos los nodos, es necesario indicar cuales son los tipos de relaciones que aparecen:

- *Subclase*. Une un nodo-tipo con la clase a la que pertenece. Por ejemplo, *Hoja-3* está unido con la clase *Lámina-7* y *Corola-1* con la clase *Verticilo-1*.
- *Disyunción*. Se usa mediante la etiqueta "OR", uniendo nodos entre sí. Por ejemplo, el enlace que une *Hoja-1* con *Hoja-2* y con *Hoja-3*, uniendo con las posibles interpretaciones de la palabra *Hoja*.
- *Conjunción*. Se usa mediante la etiqueta "AND", y también une nodos entre sí. Por ejemplo, el enlace que une *Verde-1* con *Plano-3* y con *Delgado-2* une los dos nodo-réplicas con la subclase.
- *Propiedad*. Se usa para unir tres nodos, tal como se ve en la siguiente figura, donde A es la relación que se establece entre el sujeto, es decir, B, y el objeto, es decir, C. Por ejemplo, en la definición de *Hoja-1*, se unen el nodo-réplica *Realizar*, con el sujeto *A* y el objeto *Función-1*. En este caso, la variable *A* indica el concepto que aparece en el mismo plano de la definición, es decir, hace referencia a *Lámina-7*:



- *Referencia al tipo*. Estas referencias van siempre desde el nodo-réplica hasta el nodo-tipo, dándose siempre en planos diferentes. Por ejemplo, en la definición de *Hoja-3* aparece el término *Corola-1* que a través de su enlace lleva a su definición, en el plano adecuado, es decir, en la acepción correcta. En la Figura se observa a través de la flecha punteada que apunta a *Corola-1*.

El programa creado por Quillian [Quillian, M. R. (1967)] usaba esta base de conocimiento con el fin de localizar relaciones entre pares de palabras. Dadas dos palabras, busca los grafos asociados a cada una de ellas. Puede ocurrir que exista en ambos grafos un nodo de concepto común, denominado *nodo intersección*. El camino a esos nodos intersección representa la relación entre los conceptos de esas palabras. Por

ejemplo, en la figura, el nodo intersección de los grafos asociados a las palabras *Hoja-3* y *Corola-1* es el nodo-réplica *Flor-2*, por lo que el camino que los une entre sí corresponde a la relación entre los significados de ambos conceptos.

Debido a la existencia de numerosos términos polisémicos, Quillian señaló la conveniencia de pasar de una representación de palabras a la representación de conceptos, sin depender de ningún idioma en particular. Esta idea dio lugar a una solución propuesta por Schank [Schank, R. C. (1975), Schank, R. C.; Kolodner, J. L.; DeJong, G. (1980)] y denominada *grafos de dependencia conceptual*.

2.3.1.2.- Grafos de dependencia conceptual de Schank

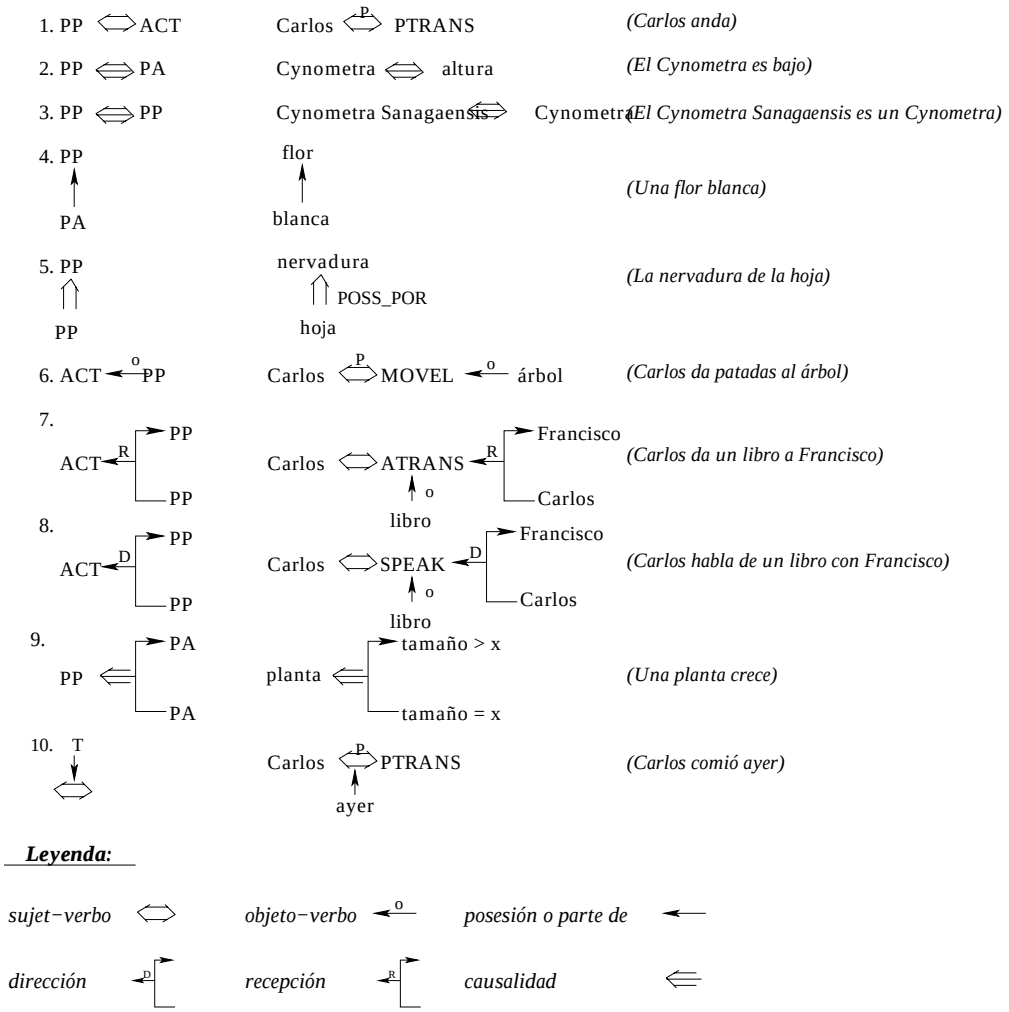
A diferencia de Quillian, Schank estaba interesado en la "comprensión del lenguaje natural" de ahí que sus perspectivas fueran diferentes. A Schank, lo que le interesaba era representar los "conceptos" que se encuentran debajo de las palabras. Además, otra diferencia con Quillian era que las representaciones que creaba Schank trataban de ser independientes del idioma que se estuviera usando, lo que, en ese momento, no ocurría con Quillian.

Concretamente, este método consiste en representar cualquier frase mediante primitivas, que pueden ser de distintos tipos:

- *Categorías conceptuales*, Seis en total, indicando si es un objeto físico (*PP*), una acción (*ACT*), el atributo de un objeto (*PA*), el atributo de una acción (*AA*), tiempo (*T*) y localización (*L*).
- *Reglas sintácticas*, dieciséis en total, determinando los diferentes tipos de relación que pueden existir entre los elementos de una frase. Entre otros se encuentran las relaciones sujeto-verbo, objeto-verbo, posesión o parte-de, dirección, recepción, causalidad.
- *Acciones primitivas*, indicando el conjunto de acciones básicas que componen otras complejas. Entre otros se encuentran:
    - PTRANS: Para transferir físicamente un objeto, es decir, cambiarlo de lugar, por ejemplo, "ir".
    - ATRANS: Para transferir una relación abstracta, como posesión o control, por ejemplo, "dar".
    - MTRANS: Para transferir información mentalmente, por ejemplo, "decir, contar, comunicar".
    - PROPEL: Es la aplicación de una fuerza física a un objeto, por ejemplo, "empujar".
    - MOVEL: El movimiento de una parte del cuerpo por su propietario, por ejemplo, "dar patadas".
    - GRASP: El acto por el que un actor coge un objeto, por ejemplo, "coger".
    - INGEST: Ingestión de un objeto por un ser animado, por ejemplo, "comer, ingerir".
    - CONC: La conceptualización o pensamiento de una idea por un actor.
    - EXPEL: Es la expulsión desde un cuerpo animado al exterior, por ejemplo, "llorar".
    - MBUILD: Es la construcción de una información a partir de una que existía, por ejemplo, "decidir".
    - ATTEND: Es la acción de dirigir un órgano de los sentidos hacia un objeto o estímulo, por ejemplo, "escuchar, mirar".

- SPEAK: Es la acción de producir sonidos, por ejemplo, "hablar".

Estas primitivas se usan para definir *relaciones de dependencia conceptual* que describen el sentido de las estructuras semánticas. Estas relaciones de dependencia conceptual son las reglas de sintaxis y constituyen una auténtica guía para el establecimiento de las relaciones semánticas significativas. De este modo cada frase se descompone en elementos simples que pretenden ser independientes del idioma, utilizando estas relaciones luego para construir la representación interna de una frase. Para ilustrar estos conceptos, la figura, inspirada en [Luger, G.F (2005)], muestra estas relaciones como un primer nivel de la construcción de la teoría, pero a partir a ellas se pueden obtener otras más complejas:



Leyenda:

Esta teoría ofrece un número importante de beneficios. Al proporcionar una interpretación de la semántica del lenguaje natural, reduce problemas de ambigüedad, limitándose a no proporcionar una forma canónica para el significado de las frases. Esto quiere decir que sólo las frases con el mismo sentido se representarán sintácticamente de un mismo modo. Otras ventajas tienen relación con la utilización de un conjunto limitado de primitivas. Éstas determinan unívocamente la representación del conocimiento, evitando una explosión combinatoria en el número de representaciones asociadas a cada frase. Al tiempo, al ser un método determinista y finito, se puede

construir un intérprete capaz de realizar inferencias; sin limitar el número de elementos y relaciones esto sería extremadamente difícil.

Sin embargo, el hecho de que este tipo de representación requiera una descripción demasiado detallada de las acciones representa una dificultad añadida, hasta el punto de que la descomposición puede resultar en extremo laboriosa. En este sentido, algunos autores, como podría ser Sowa [Sowa, J.F. (1983)], afirman que es más útil trabajar con distintos niveles de detalle y no con un conjunto cerrado de primitivas, de tal manera que se pueda explicitar los elementos cuando sea necesario. Por ejemplo, Schank sólo distingue entre seis tipos de categorías conceptuales. Concretamente, si nos centramos en la de objeto físico, Schank nos dirá que es de tipo *PP*, pero no podríamos hacer una distinción entre objetos móviles y objetos inmóviles, o incluso entre un objeto y un ser vivo, etc. Por este motivo surgen otros tipos de representaciones tales como las que vamos a ver a continuación.

2.3.1.3.- Jerarquía de conceptos

Sin duda el tipo de red semántica por excelencia es el de *redes ES-UN*, de hecho, muchas veces se mencionan como sinónimo de red semántica. Esta red es una jerarquía taxonómica, es decir, es un sistema de clasificación compuesto por una jerarquía de clases anidadas, cuya espina dorsal está constituida por un sistema de enlaces de herencia entre los objetos o conceptos de representación, conocidos como nodos. Concretamente, este tipo de redes son el resultado de la observación de que gran parte del conocimiento humano está basado en la adscripción de un subconjunto de elementos como parte de otro más general. Las taxonomías naturales (aquellas que agrupan los seres vivos en base a determinadas características comunes) son un buen ejemplo: "un vitacola es un Afzelia Africana", "un Afzelia Africana es un Caesalpinioideae", "un Caesalpinioideae es una Fabaceae", "una Fabaceae es un vegetal".

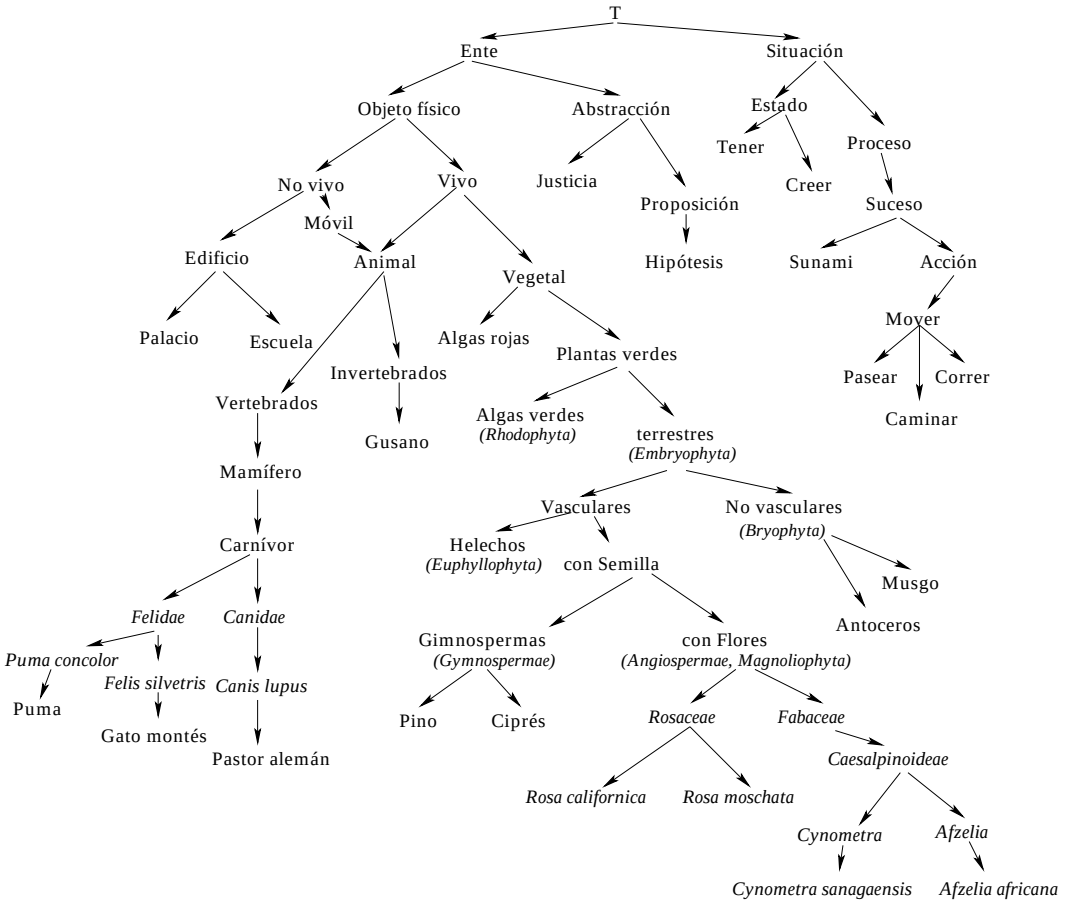
$$\forall x(vitacola(x) \rightarrow Afzelia\ Africana(x))$$
$$\forall x(Afzelia\ Africana(x) \rightarrow Caesalpinioideae(x))$$
$$\forall x(Caesalpinioideae(x) \rightarrow Fabaceae(x))$$
$$\forall x(Fabaceae(x) \rightarrow Vegetal(x))$$

Los nodos de las estructuras taxonómicas se han usado en multitud de representaciones [Brachman, R. J. (1983)], pero un hecho fundamental es la interpretación genérica o específica de los nodos, es decir, si éstos representan un único individuo o varios. Los nodos situados en lo más bajo de la jerarquía denotan individuos concretos o instancias, mientras que los nodos superiores denotan clases de individuos. En este sentido, un arco trazado desde un nodo *A* hacia un nodo *B* especifica que *A* es más general. Se trata de un grafo dirigido acíclico, en el que nos podemos encontrar bucles, pero no ciclos[11]. Un nodo puede tener varios ascendientes y descendientes, pero el descendiente de un nodo no se puede convertir en su ascendiente mediante un ciclo.

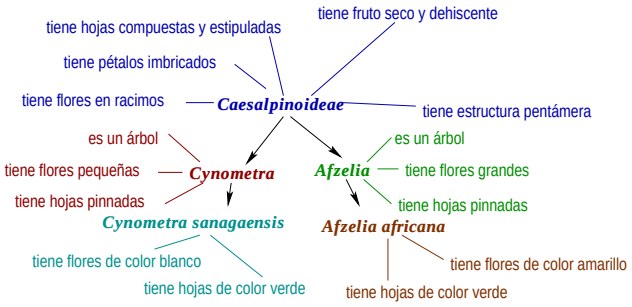
Si observamos la siguiente figura, se muestra como el concepto superior, el que engloba a todos los demás, se representa por "T". Los arcos representan relaciones de orden parcial. Es el caso del arco que va de *Vivo* a *Vegetal* donde *Vegetal* $\subseteq$ *Vivo*, es decir, que *Vivo* es más general que *Vegetal*. Como se ha dicho, en este grafo se pueden

[11] Un ciclo es un grafo que se asemeja a un polígono de *n* lados, es decir, posee un camino cerrado en el que no se repite ningún nodo a excepción del primero que aparece dos veces como principio y fin del camino.

obtener bucles y no ciclos, como ocurre con *Objeto físico*, *No vivo*, *Vivo*, *Móvil* y *Animal*. El interés de agrupar los conceptos en una red jerárquica tiene como finalidad poder realizar un tipo de inferencia que permita que un concepto herede las propiedades de sus antepasados. Concretamente, la inferencia mediante herencia de propiedades consiste en aplicar una cadena de silogismos extraídos de la lógica clásica: "Si *X* es un vegetal, los vegetales son seres vivos, y los seres vivos son objetos físicos, entonces *X* es objeto físico":



Además de esto, este tipo de jerarquía permite a una categoría heredar las propiedades de las categorías superiores. Lo que especifica y concreta una categoría son las nuevas propiedades que se añaden a las mismas distinguiéndolas de sus ascendientes. Por tanto, cuantas más propiedades tenga una categoría más bajo estaremos en la jerarquía y más específica será. Si nos centrásemos única y exclusivamente en la clase *Caesalpinioideae*, las propiedades podrían ser las que se observan en la figura siguiente, que permitiría la realización de herencia de propiedades por parte de los antepasados:



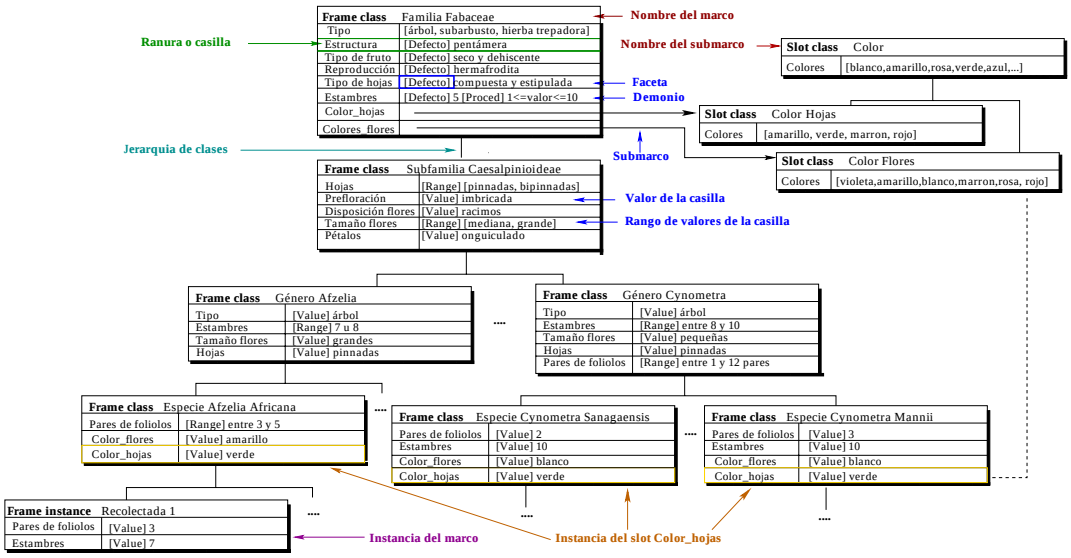
Concretamente, existen dos tipos de herencia en redes jerárquicas: la *herencia estricta*, en la que todos los conceptos descendientes de una clase poseen sus mismas propiedades; y la *herencia por defecto* que supone que los descendientes de una clase poseen sus mismas propiedades mientras no se indique lo contrario. De hecho, la posibilidad de trabajar con dos tipos de herencia plantea un problema al trabajar con grafos dirigidos acíclicos, ya que el hecho de que un nodo pueda tener distintos padres hace que puedan surgir contradicciones entre los diferentes valores por defecto heredados. De ahí surge la necesidad de establecer mecanismos para resolver estos conflictos [Touretzky, D. S. (1986), Thomason, R. H.; Touretzky, D. S. (1991)].

2.3.2.- Marcos

En el campo de la Inteligencia Artificial, el término *marco* se refiere a una forma especial de representación de conceptos, llamados *clases*, y situaciones estereotipadas, usado en una amplia variedad de dominios, como pueden ser los contextos visuales, de resolución de problemas y semánticos. Los marcos fueron propuestos inicialmente por Minsky [Minsky, M. (1974)], considerando la resolución de los problemas humanos como el proceso de rellenar huecos de descripciones de la mente y usándolos para representar dicho conocimiento mediante el rellenado de esos espacios vacíos [Simon, H. A. (1981)]. En este sentido, fueron propuestos para superar las limitaciones de la lógica a la hora de abordar problemas como la visión artificial, la comprensión del lenguaje natural o el razonamiento basado en el sentido común; precisamente por que todos estos problemas requieren identificar estructuras complejas de conocimiento que representan una situación conocida. Por lo tanto, los marcos no dejan de ser una evolución de las redes semánticas, donde el nodo es sustituido por una estructura de datos que representa una situación estereotipada a partir de sus elementos más significativos.

Se podría imaginar cada marco como una *colección de ranuras o casillas* donde se almacena la información respecto a su uso y a lo que se espera que ocurra a continuación. Cada casilla contiene la información sobre un atributo particular del objeto que se modela o una operación del marco. En muchos aspectos, un marco se podría identificar con los objetos estructurados de los lenguajes imperativos.

En este sentido, las casillas asocian información, que puede ser de tipos diferentes, y que denominamos *facetas*. Las facetas son un modo de proporcionar conocimiento extendido acerca de un atributo. Cada una puede contener un valor por defecto o un puntero a otro marco, llamado *submarco* del propio marco; un conjunto de reglas o un procedimiento con el que se obtendrá el valor de la misma, tal y como podemos ilustrar a partir de la figura. A continuación, haremos referencia a cada uno de los componentes de los marcos, refiriéndonos a ellos mediante ejemplos:



- *Nombre del marco*. Un ejemplo de nombre de clase de la figura sería *Género Cynometra*.
- *Relaciones de un marco con otro*: En la figura, el marco *Recolectada 1* es un ejemplar de la clase *Especie Afzelia Africana*, el cual a su vez pertenece a la clase *Género Afzelia*.
- *Valor de la casilla*. El valor de una casilla puede ser simbólico, numérico o booleano. Por ejemplo, en el marco de la figura, la casilla *Prefloración* de la clase *Subfamilia Caesalpinioideae* tiene un valor simbólico *imbricada* y la casilla *Pares de foliolos* de la instancia *Recolectada 1* un valor numérico *3*. Estos valores se pueden asignar cuando se crea el marco o más tarde.
- *Valor por defecto de la casilla*. El valor por defecto se toma cuando no hay evidencias de lo contrario. Por ejemplo, un marco *Familia Fabaceae* tiene una *estructura pentámera* como valor por defecto en su correspondiente casilla. Las clases que heredan de ésta, si no se indica lo contrario, también tendrán una *estructura pentámera*.
- *Rango de los valores de la casilla*. El rango va a determinar si un objeto en particular encaja con los requerimientos estereotipados definidos por el marco. Por ejemplo, las *hojas* de la *subfamilia Caesalpinioideae* podría ser considerado entre el rango de valores *pinnada y bipinnada*.
- *Información procedimental*. Una casilla puede tener asignado un procedimiento, el cuál se ejecutará si el valor de la casilla cambió o, si en cambio, se necesita para comprobar algún otro valor de otra casilla. A estos procedimientos anexados a la casilla se les denominan *demonios*. Por ejemplo, la casilla *Estambres* del marco *familia Fabaceae* tiene por defecto un valor *5*, pero también posee un demonio que se activa cuando, en los marcos heredados, ese valor cambia. El demonio será el encargado de ejecutar un procedimiento que en este caso verifique que el nuevo valor que se le asigne se encuentre entre *1 y 10*.

La colección de marcos interconectados entre sí forma un *sistema de marcos*. Se puede pensar en un Sistema de Marcos como una red de estructuras de datos y relaciones [Negnevitsky, M. (2005)], donde los marcos de los “niveles superiores” (Por ejemplo, en la figura, el marco *Familia Fabaceae*) representan lo más general de lo que se supone de la situación. Los marcos de los “niveles inferiores” poseen muchas casillas que deben rellenarse mediante instancias específicas o datos [Burkert, C. (1995)]. Por

ejemplo, en la figura anterior, el marco *Especie Cynometra Mannii* es el nivel inferior antes de definir una instancia concreta, y posee, además de las casillas de este marco (es decir, *pares de foliolos* y *estambres*), todas aquellas cuyo valor se haya definido en niveles superiores. Es el caso de la casilla *hojas* del marco *Género Cynometra* cuyo valor es *pinnada*. El marco *Especie Cynometra Mannii* heredará este valor. A diferencia de las redes semánticas, se pueden definir atributos sin valor en las clases, como ocurre en el marco *Familia Fabaceae* de la figura donde el atributo *tipo* no tiene un valor concreto. Este valor se rellena en las subclases o incluso en las instancias.

Además, cada campo puede especificar varias opciones, y sus asignaciones deben satisfacerse. Estas asignaciones pueden delimitar submarcos. Las condiciones simples se especifican por indicadores que pueden obligar a que a un campo se le asigne un objeto de valor suficiente, es el caso en el marco *Familia Fabaceae* de la figura, para el atributo *estructura*, o un puntero a un submarco de cierto tipo, como ocurre en el mismo marco para el atributo *color hojas* y *color flores*. Concretamente, este último caso indica que tanto el submarco *color hojas* y *color flores* heredan del submarco *colores*. Otras condiciones más complejas pueden especificar relaciones entre los objetos asignados a diferentes campos.

Una vez que se ha establecido la colección de marcos y se han interconectado entre sí, estamos en disposición de crear los objetos concretos que hacen referencia a esas situaciones estereotipadas. Concretamente, existen instancias de marcos que asignan ejemplares a las clases y marcos de clase que describen clases completas. La relación "es_un", abreviatura de "es miembro de la clase", asigna instancias a las clases de las que son miembros. Por ejemplo, la instancia *Recolectada 1* "es_una" *Especie Afzelia Africana*, es decir, *Recolectada 1* es un miembro de la clase. Otra relación es "es_un_tipo_de", que vincula clases entre sí. Esto implica que si una superclase tiene una relación, entonces el ejemplar la hereda. Es el caso del marco *Género Cynometra*. Éste tiene una relación "es_un_tipo_de" con *Subfamilia Caesalpinioideae*, por lo que hereda los atributos de éste que no se hayan redefinido en *Género Cynometra*. Lo mismo ocurre entre *Género Cynometra* y *Subfamilia Caesalpinioideae*.

Una vez explicado el sintaxis de los marcos, y siguiendo la figura anterior, se puede interpretar cuáles son las características asociadas a cada concepto y las relaciones que se establecen entre ellos. Por ejemplo, sabemos que la *Familia Fabaceae* es un tipo de *árbol, subarbusto* o *hierba trepadora* y que generalmente su tipo de *reproducción* es *hermafrodita*; que el *Género Cynometra* es una *Caesalpinioideae* cuya cantidad de *estambres* se sitúa *entre 8 y 10*; que la especie *Cynometra Sanagaensis* es de tipo género *Cynometra*, que generalmente tienen *flores de tamaño pequeño*. La planta *Recolectada 1* es una *especie Afzelia Africana* con *7 estambres*, *3 pares de foliolos* y cuyas flores son de color *amarillo*.

De todo lo anterior podemos colegir que una base de conocimiento basada en marcos es una colección de marcos organizados jerárquicamente, según un número de criterios estrictos y otros principios más o menos imprecisos tales como el de similitud entre estos marcos. A nivel práctico, los marcos poseen mayores posibilidades que las redes semánticas, en particular, en lo referente a:

- *Precisión*. Se precisan los objetos, las relaciones entre objetos y sus propiedades; en ausencia de evidencia contraria se usan valores por omisión. Es decir todas las propiedades especificadas en categorías superiores tienen especificado un valor. Y esos valores serán los que se tomen, si no se especifica lo contrario, en las categorías inferiores.
- *Sobrecontrol*. Para cada nodo hijo, el enlace con el nodo padre es un enlace de herencia. El nodo hijo hereda todas las casillas de su padre a menos que se

especifique lo contrario. Por ejemplo, la *Subfamilia Caesalpinioideae* hereda de la *Familia Fabaceae* el *Tipo de hojas* que tiene, es decir, *Compuesta y estipulada*. Pero a su vez, el *Género Cynometra* lo hereda de la *Subfamilia Caesalpinioideae*.

- *La herencia por defecto es no monotónica.* Debido al sobrecontrol, no hay posibilidad de negar la herencia por defecto de propiedades en un contexto o situación determinada. Esta es una gran diferencia con las redes semánticas, donde la herencia es siempre monotónica. Por ejemplo, la *Especie Afzelia Africana* al ser un marco que hereda del *Género Afzelia*, y no tener definido un valor para la propiedad *Estambres*, por herencia de propiedades por defecto esta propiedad toma el valor especificado en el marco *Familia Fabaceae*, es decir la cantidad de estambres será de *5*. Por el contrario, la instancia *Recolectada 1*, a pesar de ser también una instancia de la *Especie Afzelia Afriacana*, no hereda esta propiedad por defecto, pues tiene defino que su cantidad de estambres es de *7*.
- *Activación dinámica de procesos*. Se pueden adjuntar procedimientos a un marco o alguno de sus componentes y ser llamados y ejecutados automáticamente tras la comprobación de cambio de alguna propiedad o valor. Es el caso, por ejemplo, de la *Familia Fabaceae* donde se activa dinámicamente un proceso para comprobar que la cantidad de estambres de sus categorías inferiores se encuentran entre 1 y 10.
- *Modularidad*. La base de conocimiento está organizada en componentes claramente diferenciados. Los nodos pueden ser de dos tipos: *nodos de clase*, como por ejemplo, la *Especie Cynometra Sanagaensis*, y *nodos de instancia*, como por ejemplo, la instancia *Recolectada 1*. Todos los nodos internos (no terminales) han de ser nodos de clase.

El potencial de estas características puede verse, por ejemplo, en los mecanismos de razonamiento que son capaces de llevar a cabo: el razonamiento en un sistema de marcos se realiza mediante dos mecanismos básicos: el reconocimiento de patrones y la herencia. En el entorno de los marcos el proceso de reconocimiento de patrones se centra en encontrar el lugar más apropiado para un nuevo marco dentro de la jerarquía de marcos. Esto requiere que el mecanismo de reconocimiento sea capaz de recibir información sobre la situación existente (en forma de marco) y lleve a cabo una búsqueda de aquel más adecuado de entre todos contenidos en la base de conocimiento. En este sentido, al contrario que las reglas o las representaciones lógicas, los marcos son unidades de almacenamiento suficientemente grandes como para imponer una estructura en el análisis de una situación.

Además aporta un tipo de razonamiento que no se consigue a través de la lógica. Éste es el razonamiento por defecto y hace referencia a hacer cierto tipo de deducciones usando valores heredados por defecto. Posiblemente, estas deducciones se deban eliminar cuando se tenga más información. Esto ocurriría por ejemplo si se quisieran crear instancias directamente del marco *Familia Fabaceae*, donde por defecto el número de *Estambres* es de *5*. Cuando se hable de la *Especie Afzelia Africana* el razonamiento deductivo se habrá obtenido usando el valor por defecto, puesto que en este caso, el número de *Estambres* oscilará entre *7* y *8*.

2.4.- Manejo de Incertidumbre

Las propuestas de representación del conocimiento que se han presentado en el apartado anterior tienen en común que son más adecuadas para el manejo de lo que se suele denominar *conocimiento categórico*, es decir, conocimiento susceptible de ser verdadero o falso y asociado a un modo de razonar que podríamos calificar de preciso, en el cual se manejan reglas, hechos y conclusiones no ambiguos.

Sin embargo, al enfrentarse a situaciones del mundo real, no es frecuente encontrar un conocimiento y modo de razonar de tal índole. Por el contrario, el conocimiento que se debe representar y manejar suele ser dudoso y/o incompleto. El sistema inteligente puede no tener acceso a toda la información necesaria, el razonamiento que debe efectuar suele ser inexacto y los hechos y reglas que se manejan pueden ser inciertas.

Por una parte, la *incertidumbre* o falta de información adecuada para tomar una decisión o realizar un razonamiento puede impedir llegar a una conclusión correcta. En el ejemplo:

$$\forall x\ tieneFiebre(x) \rightarrow tieneGripe(x)$$

La expresión lógica presentada no es necesariamente cierta siempre, dado que un paciente con fiebre puede tener catarro, bronquitis, etc. En este caso, una forma "correcta" de representar este conocimiento desde el punto de vista de la lógica clásica, aunque poco útil, sería:

$$\forall x\ tieneFiebre(x) \rightarrow tieneGripe(x) \vee tieneCatarro(x) \vee tieneBronquitis(x) \vee tieneEscarlatina(x) \vee tieneEbola(x) \vee \ldots$$

Las situaciones de incertidumbre suelen darse también cuando tenemos varias reglas donde elegir y no sabemos cuál de ellas sería más correcto utilizar, por ejemplo:

$$\forall x\ fiebreAlta(x) \wedge dolorMuscular(x) \rightarrow tieneGripe(x)$$
$$\forall x\ fiebreAlta(x) \wedge dolorMuscular(x) \rightarrow tieneEbola(x)$$

Por otra parte, la imprecisión de algunos términos hace que no tengamos claro si determinado enunciado es verdadero o falso, tal es el caso del predicado "muy caliente" del siguiente ejemplo:

$$\forall x\ frenteMuyCaliente(x) \rightarrow fiebreAlta(x)$$

Los motivos por los cuales se produce la incertidumbre o la imprecisión pueden ser muy dispares. Uno de ellos es la falta de conocimiento en situaciones en las que o bien no es posible identificar o manejar todo el conocimiento relevante para razonar en un determinado dominio, o bien no es práctico hacerlo. Otro motivo es la propia ignorancia por inexistencia de un conocimiento completo de un dominio determinado o, aun habiéndolo, la imposibilidad de acceder a todos los datos que serían relevantes. Un tercer motivo podría ser la inexactitud del propio conocimiento proporcionado por un manejo de datos o informaciones erróneas (errores de lectura de un sensor, error humano, creencias falsas, etc), por la imprecisión en la formulación del conocimiento (ambigüedad en el significado de las reglas y hechos), por las existencia de inferencias previas incorrectas (que provocan incertidumbre en cascada) o por la presencia de conocimiento contradictorio.

Para tratar este tipo de situaciones se han ideado formalismos de representación del conocimiento más adecuados que los expuestos en el apartado anterior. La idea general de este tipo de esquemas de representación de la incertidumbre consiste en asociar a los elementos de un formalismo de representación categórico, como los revisados en apartados anteriores, información adicional (normalmente valores numéricos) que cuantifique su grado de certeza y manejar esa información en los procesos inferenciales. En este apartado veremos tres alternativas de representación de

conocimiento impreciso o incierto: la teoría de probabilidades, el modelo de los factores de certidumbre y las lógicas que manejan grados de verdad englobadas bajo el rótulo *Lógica Difusa*.

2.4.1.- Razonamiento probabilístico[12]

Dicho de una forma intuitiva, la Teoría de las Probabilidades proporciona un modelo teórico sólido que "resuelve", mediante el uso de un valor número denominado probabilidad, la incertidumbre respecto a un hecho, definiendo un lenguaje formal para representar conocimiento incierto, junto con una serie de mecanismos para razonar con él.

El concepto de que partiremos para presentar la teoría de la probabilidad es el de *variable aleatoria,* que se utilizan para representar las características de carácter incierto presentes en un dominio concreto. Estas variables aleatorias podrán corresponderse con proposiciones susceptibles de ser verdaderas o falsas, con medidas físicas a las que les corresponde un valor numérico o con categorías discretas como colores, letras, etc.

En la teoría de la probabilidad se denomina *probabilidad a priori* o *probabilidad incondicional*, denotada[13] $P(V_1=v_1,V_2=v_2,...,V_n=v_n)$ ó $P(V_1=v_1 \wedge V_2=v_2 \wedge ... \wedge V_n=v_n)$, a la probabilidad conjunta de que las variables aleatorias $\{V_1,V_2,...,V_n\}$ tomen el conjunto de valores $\{v_1,v_2,...,v_n\}$ sin ningún conocimiento previo de otras situaciones. Las probabilidades a priori pueden obtenerse mediante técnicas estadísticas como muestreos, distribuciones de probabilidades y estudios de frecuencias, basándose en reglas generales, o bien estimarse de forma subjetiva mediante asignación por parte de expertos. Dichas probabilidades toman valores en el intervalo [0, 1] sujetas a las siguientes restricciones:

- El valor 0 corresponde a la creencia inequívoca de que las variables no tomarán los valores indicados.
- El valor 1 corresponde a la creencia inequívoca de que las variables tomarán los valores indicados.
- El resto de posibles valores intermedios representan los niveles intermedios de certeza.
- Se verifica que $\Sigma\, v_1, v_2, ..., v_n \;\; P(V_1=v_1,V_2=v_2,...,V_n=v_n)=1$.

Se llama *probabilidad a posteriori* o *probabilidad condicional* a la probabilidad de que determinadas variables aleatorias tomen un conjunto determinado de valores en función del conocimiento sobre los valores tomados por otro conjunto de variables en situaciones previas. La utilidad de la probabilidad condicional radica en el hecho de utilizar la información sobre el valor de unas variables para obtener la probabilidad de otras. A este procedimiento se le conoce como *inferencia probabilística* y se suele definir en términos de conjuntos de *hipótesis* ($H = \{H_1=h_1,...,H_n=h_n\}$) y *evidencias* ($E = \{E_1=e_1,...,E_m=e_m\}$) del siguiente modo:

$$P(H \mid E) = \frac{P(H,E)}{P(E)}$$

Donde $P(H|E)$ se lee como la probabilidad del conjunto de hipótesis H condicionada a que conjunto de evidencias E sea cierto. En el caso de un sistema basado en reglas, la

[12] Para una exposición general de este tema véase [Russell, S.; Norvig, P. (2002)].

[13] Para simplificar la notación se considerarán únicamente variables aleatorias booleanas, que representarán proposiciones ciertas o falsas, empleando la notación abreviada $P(V_i)$ para denotar $P(V_i$=verdadero) y $P(\neg V_i)$ para denotar $P(V_i$=falso).

utilidad de estas probabilidades condicionadas es directa ya que se pueden utilizar como mecanismo para cuantificar la certeza de determinadas reglas, como en el siguiente ejemplo:

"SI *fiebre* ENTONCES *gripe* CON *P(gripe|fiebre)* = 0,75"

Es decir, la probabilidad de que el paciente tenga gripe cuando tiene fiebre es de 0,75. Si disponemos del dato de la probabilidad de que el paciente tenga fiebre podríamos calcular la probabilidad conjunta de gripe y fiebre mediante la expresión *P(gripe, fiebre)=P(gripe|fiebre)P(fiebre)*, denominada *Regla del Producto*, que se generaliza del siguiente modo:

$$P(V_n, V_{n-1}, ..., V_1) = \prod_{i=n}^{1} P(V_i \mid V_{i-1} ... V_1)$$

Si bien esta aproximación es válida, en la práctica nos encontramos con que es difícil calcular o estimar directamente todos los valores de *P(H|E)* necesarios, sobre todo en situaciones donde debemos manejar múltiples evidencias. Generalmente será más factible poder disponer de las probabilidades *P(E|H)*, *P(H)* y *P(E)*, especialmente si existe una relación causal entre *H* y *E*, es decir, si la hipótesis *H* causa la evidencia *E*.
En esos casos es posible emplear el *Teorema de Bayes* para obtener *P(H|E):*

$$P(H \mid E) = \frac{P(E \mid H)P(H)}{P(E)}$$

Por ejemplo, supongamos que tenemos un paciente con la evidencia de que tiene fiebre y pretendemos determinar cuál de los tres diagnósticos posibles (nuestras hipótesis) es más plausible: gripe, bronquitis o ébola. Suponemos también que se dispone de la probabilidad *a priori* de cada posible enfermedad (tasas de incidencia) y de las probabilidades condicionadas de que padecer dichas enfermedades provoque fiebre (sensibilidad del síntoma), que podemos interpretar como un conjunto de reglas causales cuantificadas:

SI *gripe* ENTONCES *fiebre* CON *P(fiebre|gripe) = 0,85*,
SI *bronquitis* ENTONCES *fiebre* CON *P(fiebre|bronquitis) = 0,80*
SI *ébola* ENTONCES *fiebre* CON *P(fiebre|ébola) = 0,95*.

Que son, respectivamente las probabilidades de que la gripe provoque fiebre, la bronquitis provoque fiebre y el ébola provoque fiebre. Suponiendo que la probabilidad a priori de la evidencia fuera *P(fiebre)=0,45* y las probabilidades a priori de las hipótesis fueran *P(gripe)*=0,35, *P(bronquitis)*=0,10 y *P(ébola)*=0,0001, aplicando el Teorema de Bayes tendríamos una indicación de cuál de las tres conclusiones es la más probable, en este caso sería el padecer gripe:

$$P(gripe \mid fiebre) = \frac{P(fiebre \mid gripe)P(gripe)}{P(fiebre)} = \frac{0{,}85 \cdot 0{,}35}{0{,}45} = 0{,}66111$$

$$P(bronquitis \mid fiebre) = \frac{P(fiebre \mid bronquitis)P(bronquitis)}{P(fiebre)} = \frac{0{,}80 \cdot 0{,}35}{0{,}45} = 0{,}17778$$

$$P(ébola \mid fiebre) = \frac{P(fiebre \mid ébola)P(ébola)}{P(fiebre)} = \frac{0{,}95 \cdot 0{,}0001}{0{,}45} = 0{,}00002$$

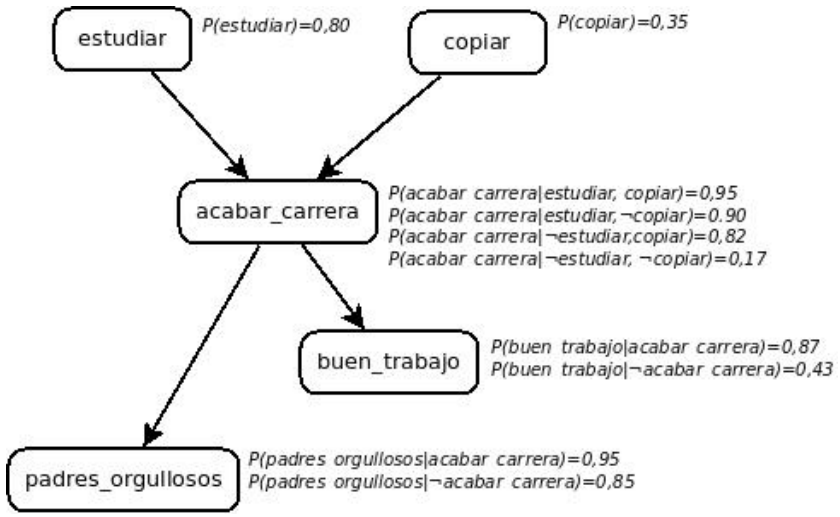
Esta aproximación basada en la aplicación directa del Teorema de Bayes presenta el problema del incremento exponencial en el número de combinaciones de probabilidades condicionadas del tipo *P(E|H)* que es necesario definir o estimar a medida que crece el número de evidencias a considerar en las inferencias. Por ejemplo,

en un diagnóstico de gripe en base a tres síntomas, fiebre, tos y dolor muscular, para evaluar *P(gripe|fiebre,tos,dolor_muscular)* utilizaríamos en el numerador de la expresión del Teorema de Bayes la probabilidad *P(fiebre,tos,dolor_muscular| gripe)*, que aplicando la regla del producto se descompondría en tres factores, *P(fiebre|tos,dolor_muscular, gripe)*, *P(tos|dolor_muscular,gripe)* y *P(dolor_muscular| gripe)*. En la práctica, no será posible disponer de datos para estimar probabilidades condicionadas con combinaciones de variables aleatorias tan complejas, ni de expertos capaces de proporcionar valoraciones subjetivas a probabilidades de ese tipo.

Para garantizar un uso efectivo de estas aproximaciones basadas en el Teorema de Bayes es necesario imponer restricciones adicionales sobre las variables aleatorias para simplificar el cálculo de este tipo de probabilidades complejas. Se dirá que dos variables aleatorias V_i y V_j son la *condicionalmente independientes* entre sí dado un conjunto de variables $\boldsymbol{X}$ si $P(V_i|V_j,\boldsymbol{X})=P(V_i|\boldsymbol{X})$ y viceversa. Es decir, dos variables son condicionalmente independientes si la presencia de un conjunto de variables **X**, hace que el conocimiento sobre los valores que toma una de ellas no afecte a la probabilidad condicionada de la otra. La existencia de relaciones de independencia condicional permite simplificar la expresión de la Reglas del Producto, para los casos en los que todas las variables aleatorias, $V_1, V_2, \ldots V_n$, son condicionalmente independientes entre sí dado un conjunto de variables $\boldsymbol{X}$, resultando la siguiente expresión:

$$P(V_n, V_{n-1}, \ldots, V_1 \mid X) = \prod_{i=n}^{1} P(V_i \mid X)$$

Para capturar las relaciones de independencia condicional entre las variables de un dominio y simplificar el cálculo de probabilidades condicionadas implicadas en las inferencias probabilísticas se utiliza un formalismo de representación conocido como *Redes Bayesianas*. Las Redes Bayesianas, también denominadas redes causales, son grafos dirigidos acíclicos en los que cada nodo representa una variable aleatoria y cada arco dirigido representa una dependencia probabilística (generalmente una relación causal) entre dos variables. Además, como se muestra en la siguiente figura, cada nodo tiene asociada una tabla de probabilidades condicionales respecto a sus padres que especifica cuantitativamente el efecto de sus padres sobre el nodo *Vi*. A los nodos sin padres, que son las evidencias o percepciones iniciales, se les asigna su probabilidad a priori.



A partir de la topología que definen las conexiones de una red bayesiana se pueden determinar una serie de relaciones de independencia condicional entre las variables o nodos de la misma. En una red bayesiana se verifica que un nodo V_i es condicionalmente independiente de todos los demás nodos que no sean descencientes suyos dados sus padres, *padres*(V_i). Utilizando esta propiedad se puede simplificar la expresión de la Regla del Producto para el cálculo de probabilidades en redes bayesianas, asumiendo una ordenación adecuada en el conjunto de variables aleatorias considerado, V_1,V_2,...V_n. La Regla del Producto para redes bayesianas queda enunciada del siguiente modo, donde los valores de todas la probabilidades condicionadas necesarias estarán presentes en las tablas de probabilidades asociadas los nodos de la red:

$$P(V_n, V_{n-1}, ..., V_1) = \prod_{i=n}^{1} P(V_i \mid padres(V_i))$$

Existen algoritmos que, dada una red bayesiana, permiten calcular cualquier probabilidad condicionada donde estén involucradas las variables aleatorias de la red, asegurando resultados exactos para el caso de redes simples (árboles o poliárboles) o resultados aproximados para el caso de redes complejas. Haciendo uso de este tipo de mecanismos de cálculo se podrán obtener las probabilidades implicadas en las inferencias realizadas a partir únicamente de la topología de la red y de las tablas de probabilidades condicionadas parciales. De este modo será posible hacer inferencias probabilísticas y cuantificar la incertidumbre de las conclusiones obtenidas.

2.4.2.- Modelo de los factores de certidumbre[14]

El modelo de los factores de certidumbre se engloba dentro de un conjunto de técnicas heurísticas menos formales, como el caso de la Teoría Evidencial de Dempster y Shafer [Shafer, G. (1976)], que en su momento fueron propuestas como solución para abordar el problema de la incertidumbre en el conocimiento. Se trata de un modelo "ad hoc" desarrollado originalmente para sistemas de diagnóstico médico basados en reglas. En concreto para el Sistema Experto MYCIN [Shortliffe, E. H.; Buchanan, B. G. (1975)] diseñado para el diagnóstico de enfermedades infecciosas. Este modelo carece de una base teórica sólida, pero resulta ser un sistema muy intuitivo y poco costoso computacionalmente que facilita la adquisición de conocimiento de expertos. Se trata de un formalismo orientado a dotar de capacidades de representación y razonamiento incierto a los sistemas basados en reglas con encadenamiento hacia adelante. Consiste en asociar a los hechos y reglas *factores de certidumbre (CF, certain factor)*, cuantificando el grado de confianza y desconfianza en tales hechos y reglas. Cuando se realizan inferencias, los factores de certidumbre de los elementos involucrados, reglas y hechos, se combinan y se propagan para obtener el valor de los factores de certidumbre de los nuevos hechos inferidos.

Desde un punto de vista práctica, los factores de certidumbre, denotados *CF(h,e)* (factor de certidumbre de una hipótesis *h* conocida una evidencia *e*), se asocian las reglas de forma subjetiva por parte de los expertos del dominio durante la fase de adquisición del conocimiento. El valor del CF asociado a una regla o a un hecho toma valores entre -1 y 1, son la siguiente interpetación intuitiva:

- Un valor +1 representa una creencia total en que el hecho o regla sea ciertos.
- Un valor -1 representa una creencia total en que el hecho o regla sea falso.

[14] Para una exposición general de este tema véase [Shortliffe, E. H.; Buchanan, B. G. (1975)].

- Un valor positivo (*CF(h,e) > 0*) cuantifica un grado de apoyo a favor de que el hecho o regla sean ciertos.
- Un valor positivo (*CF(h,e) > 0*) cuantifica el grado en que se desestima que el hecho o regla sean ciertos
- Un valor 0 representan la ignorancia total respecto al hecho o regla.

Además, los factores de certidumbre satisfacen la propiedad C*F(h,e) = –CF(¬h,e)*

Como mencionábamos, en la práctica, los valores de los CF suelen establecerlos inicialmente los expertos de forma subjetiva. A medida que se va encadenando la ejecución de las distintas reglas activadas, se van combinando los valores de los CF de los hecho iniciales con los CF de estas reglas para obtener los CF de los hechos intermedios insertados en la memoria activa del sistema basado en reglas, así como los CF de las conclusiones finales. Este es el motivo por el que este tipo de metodologías suelen recurrir a técnicas de *poda*. Por ejemplo, en MYCIN se desechaban los hechos intermedios que tuviesen factores de certidumbre cercanos a cero, en concreto aquellos con valores entre –0,2 y +0,2, dado que se consideran hechos cuyo grado de ignorancia respecto a su credibilidad es excesivo.

Para integrar el cálculo de factores de certidumbre en las inferencias realizadas por los sistemas basados en reglas mediante encadenamiento hacia adelante se define un conjunto de mecanismo de combinación y propagación de CF, que se aplicarán en el momento en que las reglas activadas sean ejecutadas.

1. Partiendo del CF de los hechos de la memoria activa que emparejaron con los antecedentes de la regla a ejecutar se calcula en CF del antecedente atendiendo a si este se corresponde con una conjunción o una disyunción de hechos:
   - $CF(e_1 \wedge e_2, e') = min\{CF(e_1, e'), CF(e_2, e')\}$, para el caso de reglas con antecedentes del tipo "*IF (e1 AND e2)*".
   - $CF(e_1 \vee e_2, e') = max\{CF(e_1, e'), CF(e_2, e')\}$, para el caso de reglas con antecedentes del tipo "*IF (e1 OR e2)*".
2. Una vez cuantificada la credibilidad del antecedente, este CF se combina con el CF de la regla a ejecutar para determinar el CF de la consecuencia inferida, empleando la fórmula para propagación de CF:

   $CF(h, e') = CF(antecedente, e') \times max\{0, CF(regla)\}$, para el caso de reglas con el esquema "*IF antecendente THEN h WITH CF(regla)*"
3. Finalmente, en caso de que un mismo hecho esté apoyado o desestimado por varias reglas, se utiliza la siguiente fórmula para combinar los efectos de reglas en paralelo:

$$CF(h, h_1 \,||\, h_2) = \begin{cases} CF(h_1, e') + CF(h_2, e') - CF(h_1, e')CF(h_2, e') \\ si \quad CF(h_1, e') > 0 \quad y \quad CF(h_2, e') > 0 \\ CF(h_1, e') + CF(h_2, e') + CF(h_1, e')CF(h_2, e') \\ si \quad CF(h_1, e') < 0 \quad y \quad CF(h_2, e') < 0 \\ \dfrac{CF(h_1, e') + CF(h_2, e')}{1 - min\{|CF(h_1, e')|, |CF(h_2, e')|\}} \quad \text{en otro caso} \end{cases}$$

En el primer caso, cuando ambos CF son positivos, se acumula creencia, en el segundo, con CF negativos, se acumula desconfianza y en el tercer caso, con evidencias en ambos sentidos, estas se contrarrestan mutuamente.

El modelo de los factores de certidumbre presenta como gran ventaja el proporcionar un procedimiento intuitivo, simple y muy eficiente que permite manejar

múltiples fuentes de evidencia y expresar la creencia y la incertidumbre en los elementos del esquema de representación del conocimiento, en este caso reglas y hechos. Los principales problemas de este modelo radican en su escasa capacidad de generalización y adaptación, limitando su uso a dominios similares a los que fueron su origen, el diagnóstico médico mediante sistemas basados en reglas. Además, pese a que el modelo de MYCIN pretendía alejarse de los modelos probabilísticos puros, alegando que eran excesivamente rígidos e imponían demasiadas restricciones, se demostró su equivalencia con un subconjunto de la teoría de probabilidades, resultando que, en realidad, las restricciones de independencia condicional que exigía el modelo probabilístico subyacente al concepto de los factores de certidumbre eran mayores que las de la teoría de la probabilidad clásica.

2.4.3.- Lógica Difusa

La *Lógica Difusa*, también llamada *Lógica Borrosa* o *Lógica Fuzzy* se debe a Lotfi Zadeh, quién en 1965 aplicó la Lógica Multivalorada a la Teoría de Conjuntos creando lo que se denomina teoría de conjuntos borrosos [Zadeh, L. (1965)] tras darse cuenta de los problemas de las propuestas basadas en Lógica Clásica a la hora de tratar conceptos imprecisos como "alto", "joven", etc. cuyo uso dentro de enunciados hace que se realicen afirmaciones que no son absolutamente verdaderas o absolutamente falsas y que son provocados por el hecho de que los objetos que manejamos en el mundo real no suelen tener criterios de pertenencia a conjuntos definidos de forma precisa. Es decir, en determinados contextos, puede ser difícil establecer de forma precisa si un individuo que mide 1,70 pertenece o no al conjunto de las personas altas.

Podría considerarse la Lógica Difusa como una ampliación de la Lógica Clásica orientada a dar soporte a razonamiento con imprecisión. Esta lógica se caracteriza porque en su semántica utiliza grados de verdad de tal forma que el valor semántico de los elementos considerados por la lógica no es solamente Verdadero o Falso sino que se establece una gradación. En las reglas de inferencia de esta lógica (por ejemplo, el *modus ponens difuso*) se combinan esos grados de verdad.

La Lógica Difusa resulta útil para representar conceptos imprecisos y operar con ellos, para manejar procesos complejos o no lineales y para modelizar partes desconocidas del un sistema o partes que no pueden medirse de manera fiable. Las principales aplicaciones de la Lógica Difusa han surgido en el ámbito de los Sistemas Expertos Difusos y en el ámbito del control de sistemas donde a día de hoy se han presentado en el mercado o como prototipos, desde sofisticados sistemas de control de tráfico, sistemas de frenado, vehículos no tripulados controlados por voz o electrodomésticos como lavadoras, cámaras de fotografía o de vídeo.

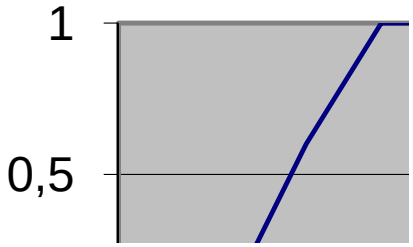
La *Teoría de Conjuntos Difusos* es, por tanto, una generalización de la Teoría Clásica de Conjuntos que proporciona un formalismo para la representación de la imprecisión y la incertidumbre en el que los elementos de un dominio pueden pertenecer parcialmente con diferentes grados de pertenencia a uno o a varios conjuntos. Además, en esta propuesta, existe la posibilidad de tratar lingüísticamente dicha pertenencia.

Un *conjunto difuso A* en un dominio *D*, viene caracterizado por una *función de pertenencia* $f_A(x)$ que asocia a cada elemento x del dominio un valor en el intervalo real [0, 1] que determina su grado de pertenencia a ese conjunto. De esta manera se relaja la pertenencia a un conjunto de tal forma que $\forall x \in D\ f_A(x) \in [0, 1]$. La teoría clásica de conjuntos es un caso particular de la teoría de conjuntos difusos cuando:

$$f_A(x) = \begin{cases} 1, & \text{si } x \in A \text{ (pertenencia total a } A) \\ 0, & \text{si } x \notin A \text{ (no pertenencia a } A) \end{cases}$$

La propuesta de una función de pertenencia para determinado conjunto difuso va a depender de múltiples factores como el concepto a definir, el contexto, la aplicación que se está diseñando etc. Son preferibles las funciones de pertenencia simples ya que simplifican los cálculos, eliminan complejidad computacional y no pierden exactitud. Cuando contamos con universos de discurso discretos, la definición de una función de pertenencia se hace extensionalmente mediante conjuntos de pares tal que el primer elemento del par es un elemento del dominio y el segundo elemento del par es un grado de pertenencia al conjunto en cuestión. Por ejemplo, para el concepto difuso "cerca de 4" la función de pertenencia podría ser:

{0/0, 1/0.1, 2/0.6, 3/1, 4/1, 5/1, 6/0.6, 7/0.1, 8/0, … }



Las funciones de pertenencia más usuales suelen tener forma trapezoidal o triangular aunque también se utilizan formas escalonadas, sigmoidales o gausianas.

Las siguientes son algunas definiciones que caracterizan a los conjuntos difusos:

- *Altura*: Es el mayor valor de la función de pertenencia

$$Altura(A) = argmax_{x \in D}\{A(x)\}$$

- *Soporte*: Son los elementos de D que pertenecen al conjunto A con grado mayor que 0

$$Soporte(A) = \{x \in D | A(x) > 0\}$$

- *Núcleo*: Son los elementos de D que pertenecen al conjunto A con grado 1

$$Nucleo(A) = \{x \in D | A(x) = 1\}$$

- *α-corte*: Son los elementos de D con grado de pertenencia de como máximo α

$$A_\alpha = \{x \in D | A(x) \leq \alpha\}$$

- *Teorema de Representación*: Todo conjunto difuso puede descomponerse en una familia de conjuntos difusos. Además cualquier conjunto difuso podrá reconstruirse a partir de una familia de sus α-cortes.
- *Conjunto vacío*: un conjunto difuso es vacío si su función de pertenencia es siempre 0

$$\forall x \in D \; f_A(x) = 0$$

- *Igualdad*: dos conjuntos difusos, definidos sobre el mismo dominio D, son iguales si tienen la misma función de pertenencia

$$A = B \text{ si y sólo si } \forall x \in D \; f_A(x) = f_B(x)$$

- *Inclusión*: Un conjunto difuso, A, está incluido en otro, B, si su función de pertenencia toma valores iguales o más pequeños para todo elemento del dominio

$$A \subseteq B \text{ si y sólo si } \forall x \in D \; f_A(x) \leq f_B(x)$$

- *Unión*: $A \cup B$: $f_{A \cup B}(x) = max\{f_A(x), f_B(x)\}$
- *Intersección*: $A \cap B$: $f_{A \cap B}(x) = min\{f_A(x), f_B(x)\}$
- *Conjunto complementario*: $\neg A$: $f_{\neg A}(x) = 1 - f_A(x)$

El máximo y el mínimo son dos de las operaciones más utilizadas para modelizar la unión y la intersección respectivamente pero también se puede representar la conjunción con otras funciones como la suma acotada ($f_{A \cup B}(x)=min\{1, f_A(x)+f_B(x)\}$) y la intersección con funciones como el producto ($f_{A \cap B}(x) = f_A(x) \cdot f_B(x)$). En la Teoría de Conjuntos Difusos se han propuesto un buen número de funciones de este estilo con el fin de proporcionar modelos para la unión y la intersección y se han analizado sus propiedades. La selección de unas funciones u otras hará que el sistema completo tenga unas propiedades u otras.

Tomando como base la Teoría de Conjuntos Difusos se ha propuesto una Lógica Difusa que es una generalización de una Lógica Multivalorada que hace corresponder semánticamente un grado de verdad a cada una de las fórmulas lógicas del sistema en lugar de una verdad estricta. Esta lógica permite representar y utilizar conceptos imprecisos y realizar razonamientos aproximados. Los grados de verdad utilizados pueden ser valores numéricos del intervalo real [0, 1] o etiquetas lingüísticas como "muy verdadero", "bastante verdadero", "poco verdadero", "muy falso", "bastante falso" etc. Estos grados de verdad están asociados a conjuntos difusos de tal forma que, al igual que sucede con la Lógica Clásica y la Teoría Clásica de Conjuntos también existe un homomorfismo entre la Lógica Difusa y la Teoría de Conjuntos Difusos.

Las siguientes son algunas definiciones que caracterizan a la Lógica Difusa:

- *Fórmulas atómicas*: Son proposiciones susceptibles de tener un valor de verdad en el intervalo [0, 1] o bien expresiones predicativas de la forma $A(x)$ ("x es A") donde A es un conjunto difuso. En este sentido, el valor de verdad de una fórmula $A(x)$ es el grado de pertenencia del conjunto difuso asociado ($f_A(x)$).
- *Conectivas lógicas*: Las fórmulas bien formadas se construyen de igual modo que en la Lógica Clásica, lo que cambia es su interpretación semántica. Las conectivas se corresponden con las operaciones sobre conjuntos difusos[15]:
    - *Conjunción ("y")*: Dados dos conjuntos difusos P y Q pertenecientes a dos dominios. La conjunción $P \wedge Q$ indica en que medida un elemento x del primer dominio pertenece al conjunto difuso P y un elemento y del segundo dominio pertenece al conjunto difuso Q.

      $$f_{P \wedge Q}(x, y) = min\{f_P(x), f_Q(y)\}$$

      Por ejemplo, "x es alto e y es muy guapo"

      $$F_{alto \wedge muy\ guapo}(x, y) = min\{f_{alto}(x), f_{muy\ guapo}(y)\}$$

      Esto hace corresponder el grado de verdad de las conjunciones con el grado de pertenencia a la intersección de dos conjuntos difusos.
    - *Disyunción ("o")*: Dados dos conjuntos difusos P y Q pertenecientes a dos dominios. La disyunción $P \vee Q$ indica en qué medida un elemento x del primer dominio pertenece al conjunto difuso P o un elemento y del segundo dominio pertenece al conjunto difuso Q.

      $$f_{P \vee Q}(x, y) = max\{f_P(x), f_Q(y)\}$$

      Por ejemplo, "la temperatura es alta o la humedad muy baja"

      $$f_{temp.\ alta \vee hum.muy\ baja}(x, y) = max\{f_{temp.\ alta}(x), f_{hum.muy\ baja}(y)\}$$

      Esto hace corresponder el grado de verdad de las disyunciones con el grado de pertenencia a la unión de dos conjuntos difusos.

[15] De igual modo a como sucedía en la Teoría de Conjuntos Borrosos donde existen varias funciones alternativas que modelizan la intersección y la unión, en la Lógica Difusa existen las mismas alternativas para modelizar la conjunción y la disyunción respectivamente.

- *Negación ("no")*: Dado un conjunto difuso P que es un subconjunto de un dominio. La negación $\neg P$ indica en qué medida un elemento x del dominio pertenece al conjunto difuso complementario de P.

$$f_{\neg P}(x) = 1 - f_P(x)$$

- *Condicional ("Si ... entonces ...")*: Define una relación difusa entre dos conjuntos P y Q que tiene gran relevancia porque da soporte a reglas difusas. Igual que sucede con la conjunción (intersección) y con la disyunción (unión) existen varias funciones que modelizan el condicional. $f_{P \to Q}(x, y)$ representa el grado de verdad del condicional "si x pertenece a P entonces y pertenece a Q". Una de las funciones utilizadas para definir el condicional es la conocida como condicional de Kleene que se obtiene a partir de la regla de interdefinición condicional-disyunción de la Lógica Clásica ($P \to Q = \neg P \vee Q$):

$$f_{P \to Q}(x, y) = max\{1 - f_P(x), f_Q(y)\}$$

Por ejemplo, "si la temperatura es alta, entonces subida normal de la calefacción"

$$f_{temp.alta \to subida\_normal.}(x, y) = max\{1 - f_{temp.alta}(x), f_{subida\_normal.}(y)\}$$

Una de las principales características de la Lógica Difusa es que facilita el uso de etiquetas lingüísticas a la hora de establecer valores de verdad de una fórmula. Para predicados correspondientes a los conceptos "altura" o a la "belleza" de un individuo podríamos tener por ejemplo:

Variable	Expresiones Ling.	Dominio
altura	"bajo", "muy alto", "aprox 1,80"	numérico (cm.)
belleza	"muy guapo", "feo", "un callo",...	personas (discreto)

Las etiquetas lingüísticas son utilizadas principalmente en la definición de reglas difusas y tienen el cometido de comprimir información ya que una etiqueta aúna un grupo de valores. También caracterizan fenómenos mal definidos o complejos y convierten términos lingüísticos en descripciones numéricas dado que traducen un proceso simbólico a un proceso numérico. Existen dos tipos de etiquetas lingüísticas:

- *Los términos primarios*: suelen ser adjetivos calificativos asociados a un conjunto difuso P sobre el dominio de una variable. Por ejemplo, para variables como la temperatura, la estatura, etc. tendríamos términos primarios como "alto", "bajo", "normal", "pequeño", etc.
- *Los términos compuestos*: su conjunto difuso se define a partir de los términos primarios usando modificadores lingüísticos y cuantificadores como por ejemplo, "muy alto", "moderadamente alto", "poco bajo", "extremadamente pequeño", etc.
  - *Modificador lingüístico*: es un operador que transforma el conjunto difuso asociado a un término primario P en otro conjunto difuso. Por ejemplo existen modificadores lingüísticos de concentración como "muy", "más", etc. de dilatación como "más o menos", "menos", "poco", etc. de intensificación del contraste como "especialmente", "bastante cerca de", etc. de difuminación como "cerca de", "casi", etc.
  - *Cuantificadores difusos*: así como en la Lógica Clásica, los dos únicos cuantificadores son "para todo" y "existe un", en Lógica Difusa tenemos cuantificadores absolutos referidos a una única cantidad como "muchos", "pocos", "muchísimos", "aproximadamente entre 7 y 9", etc. y cuantificadores relativos referidos a una proporción de elementos respecto del total como "la mayoría", "la minoría", "casi todos", "casi ninguno", "aproximadamente la mitad", etc.

Después de aportar todas estas técnicas de representación del conocimiento, en el ámbito de la Lógica Difusa se proponen procedimientos para formular reglas de razonamiento aproximado. Así como en la Lógica Clásica sólo es posible realizar una inferencia si los datos coinciden exactamente con las premisas, en los procesos de inferencia difusa se puede obtener como conclusión un conjunto difuso a partir de unas premisas también difusas, de tal forma que a partir de las premisas *"si la curva es cerrada entonces reducir la velocidad"* y *"la curva es ligeramente cerrada"*, empleando la regla de inferencia del *modus ponens difuso*, se podría establecer la conclusión *"reducir moderadamente la velocidad"*.

Las principales ventajas de la Lógica Difusa son que combina razonamiento simbólico y numérico, que está muy próxima a la forma de expresión humana, que es simple y eficiente desde el punto de vista computacional y, para contextos restringidos y concretos, ha demostrado que es capaz de proporcionar aplicaciones exitosas.

Entre los inconvenientes de la Lógica Difusa se podría destacar que, en ocasiones, la interpretación de los valores difusos resulta difícil debido al uso de una semántica poco clara. Por otra parte, a veces resulta complicado justificar los razonamientos efectuados mediante técnicas de razonamiento aproximado. Finalmente, la existencia de múltiples operadores para las conectivas lógicas y para las reglas de inferencia hace que los diseñadores de sistemas tengan que dedicar tiempo a la delicada y compleja labor de seleccionar cuáles son los operadores más adecuados para tratar determinado problema, lo cual redunda, la mayoría de las veces, en la necesidad de realizar complicados ajustes en los sistemas.

3.- Ontologías

3.1.- Introducción

Hasta el momento se han mostrado algunos tipos de representación del conocimiento vinculados a determinados modos de representación del razonamiento como pueden ser la Lógica Clásica, los Sistemas Basados en Reglas, el Razonamiento Probabilístico o la Lógica Difusa entre otros. Desde el advenimiento de Internet y, más concretamente, con la evolución de la World Wide Web que ha traído consigo tal advenimiento, las ontologías se han mostrado como una nueva forma de representar el conocimiento altamente adecuada en este contexto. Aunque existe cierta controversia con respecto a la definición de lo que es una ontología, algo de lo que daremos cuenta en el próximo apartado, baste para esta breve introducción concebir las ontologías como especificaciones formales y explícitas de términos en un determinado dominio y sus relaciones entre ellos [Gruber, T. R. (1993-1)]. Así definida, una ontología viene a ser un listado más o menos exhaustivo de términos que, mediante las relaciones establecidas entre ellos, se disponen, de manera generalmente arbórea, formando una taxonomía más o menos compleja. La exhaustividad de la ontología viene dada por el volumen del listado de términos mientras que la complejidad de la misma viene dada por la cantidad y el tipo de relaciones definidas entre tales términos.

Uno de los principales objetivos de la creación y uso de las ontologías es unificar el conocimiento de un dominio concreto que figura en distintas localizaciones. Ello supone una ventaja tanto para agentes humanos como para agentes automáticos. Suele suceder que sitios Web distintos dedicados al mismo tema no coincidan al completo en la terminología ni en la caracterización de los conceptos que manejan. Esto supone una mayor dificultad a la hora de realizar búsquedas por parte de los usuarios de

la Web y cierta pérdida de precisión a la hora de ser clasificados mediante los procedimientos automáticos de los buscadores. Si estos sitios poseyesen una misma ontología subyacente, estas dificultades se verían mermadas considerablemente, y a su vez, se facilitaría la incorporación de nueva información a los mismos.

Otra de las ventajas de las ontologías es su carácter reutilizable. En ocasiones, la creación de una ontología requiere la incorporación de información que ya figura en otra ontología, esto significa que los creadores de la primera pueden utilizar la información que figura en la segunda. Por otra parte, pueden existir ontologías complementarias de tal forma que uniéndolas se obtiene una ontología mayor y más completa.

Además las ontologías permiten hacer explícitos los supuestos de un dominio. Esto es de gran utilidad cuando un nuevo usuario necesita aprender los significados de los términos de un determinado dominio. Por otra parte, si el conocimiento del dominio cambia, la explicitación de los supuestos hace que éstos puedan ser modificados sin demasiada dificultad.

Otra de las ventajas de las ontologías es que separan el conocimiento de un dominio concreto del conocimiento operacional que se puede aplicar en ese dominio o en otros. Por ejemplo, imaginemos una ontología sobre la instalación y configuración de un ordenador. Se distinguirá entre el conocimiento proporcionado para cada uno de los componentes del ordenador (conocimiento del dominio) y el procedimiento mediante el cual se configura o instala (conocimiento operacional). Este procedimiento podría utilizarse en un dominio de instalación y configuración de una impresora, cambiando los términos del dominio de instalación y configuración de un ordenador por el de instalación y configuración de una impresora.

Finalmente, otra ventaja es que la existencia de una ontología para determinado dominio hace que el conocimiento de tal dominio sea analizable. Dado que tal análisis puede ser formal o no, resulta de gran utilidad no sólo para el agente humano interesado por el contenido del dominio en cuestión sino también para el agente automático que pretende utilizar, reutilizar o extender ontologías.

3.1.1.- Definiciones de ontología

El concepto de ontología es utilizado en Filosofía para referirse al tipo de estudio realizado por Aristóteles de lo que él llamaba “Filosofía Primera” y que los estudiosos de su obra posteriormente denominaron “Metafísica”. Aunque el término como tal no fue acuñado hasta el siglo XVII, la ontología podría definirse, sin abusar en exceso de la terminología filosófica, como el estudio del ser en cuanto ser. Es decir, la ontología trata de definir qué es el existir y trata de establecer categorías con respecto a lo que existe.

Aunque está muy relacionada con esta definición, el concepto de ontología utilizado en Informática establece una restricción fundamental que ha sido puesta de manifiesto por Gruber en una de las definiciones de ontología más citadas en este ámbito: “Una ontología es una especificación explícita de una conceptuación. El término se ha obtenido de la filosofía, en la que una ontología es una explicación sistemática de la existencia. En un sistema de inteligencia artificial, lo que existe es todo aquello que puede ser representado” [Gruber, T. R. (1993-2)].

Ya se ha indicado que la definición de ontología resulta controvertida. Se han realizado diversas propuestas algunas de las cuales han sido clasificadas por Guarino y Giaretta en tres grandes apartados [Guarino N.; Giaretta P. (1995)]:

- *Ontología como entidad semántico-conceptual*: Se han propuesto dos concepciones de ontología a este respecto. Una de ellas la considera como un sistema conceptual informal que subyace a una determinada base de conocimiento. La otra la considera como una explicación semántica que subyace a determinada base de conocimiento y que es expresada en términos de ciertas estructuras formales apropiadas a nivel semántico.
- *Ontología como objeto sintáctico específico*: A este respecto se han propuesto tres caracterizaciones ontología. La primera de ellas la concibe como la representación de un sistema conceptual mediante el uso de una teoría lógica, ya sea ésta caracterizada por sus propiedades formales o sólo por sus propósitos. En este sentido una ontología no sería otra cosa que una teoría lógica a la que se podría obligar o no a satisfacer una serie de propiedades formales para que pueda ser considerada una ontología. La segunda de las caracterizaciones concibe a la ontología no como una teoría lógica sino como un vocabulario utilizado por una teoría lógica. La tercera concibe a la ontología como una especificación de segundo nivel (meta-especificación) de una teoría lógica que da cuenta de los componentes estructurales o primitivos utilizados dentro de un particular dominio teórico.
- *Ontología como especificación de una conceptuación*: Es la propuesta por Gruber y, por regla general, la más utilizada en el campo de la Inteligencia Artificial para definir el término. Uno de los problemas que posee tal definición es que no deja claro en qué consiste el término "conceptuación". Guarino y Giaretta proponen que una conceptuación es una estructura semántica intensional que codifica las reglas implícitas que afectan a la estructura de una parte de la realidad.

3.1.2.- Componentes de una ontología

Para representar el conocimiento de un dominio mediante una ontología suele ser necesario definir componentes de la misma como pueden ser:

- *Conceptos*: son las entidades fundamentales a formalizar, para ello se utilizan por regla general las clases de objetos pero los conceptos pueden ser también métodos, procesos de razonamiento, estrategias, etc. Los conceptos vienen definidos por el establecimiento de:
    - *Roles, atributos o propiedades*: se denominan también *slots* y son el modo de definir un concepto. Son las características o rasgos que definen al concepto y que son relevantes para la representación del mismo dentro del sistema.
    - *Facetas*: son restricciones de roles y proporcionan información como los tipos de datos de los roles (string, real, entero, booleano, etc.), los valores posibles para un rol (dominio y rango de un rol), el número de valores (cardinalidad o número de valores que puede tomar un rol) y cualquier descripción necesaria para la definición de un rol.

  Los conceptos cristalizan en lo que se denominan *instancias*. Las instancias son objetos de una clase que representa un concepto en la que todos o parte de los valores correspondientes a los roles están rellenos. Cuando este proceso de relleno se realiza a partir de texto libre se denomina *anotación* y puede ser realizado o bien mediante la incorporación de etiquetas semánticas dentro del

texto cuando el texto que se está procesando es editable o bien directamente sobre el modelo cuando no existe la posibilidad de editar o procesar el texto.
Existe la posibilidad de definir clases de las que no se permite crear instancias. Estas clases se denominan *clases abstractas* y se utilizan para representar conceptos generales (abstractos) que suelen agrupar distintos conceptos más concretos.

- *Relaciones*: representan el modo en como los conceptos de un determinado dominio interactúan entre sí. Son las que unen los conceptos formando la taxonomía de la ontología. Algunos ejemplos de relaciones son "… es subclase de …", "… es parte de …", etc. pero la relación más relevante es la relación del tipo "… es un …" también denominada *herencia* debido a que permite que las clases relacionadas (hijas) hereden las propiedades o atributos de la clase con la cual se relacionan (padre). Existe también la posibilidad de que una clase herede propiedades de dos o más clases padre; a este fenómeno se le conoce como *herencia múltiple*. El procedimiento de la herencia es recursivo en el sentido de que puede darse sucesivamente generándose un árbol jerárquico con las dimensiones y profundidad que la propia ontología requiera; a este hecho se le denomina *derivación*. Debido a que las clases están dispuestas de esta forma jerárquica, cuando se genera la instancia de una clase concreta también se genera una instancia de las clases padre de ésta. Este tipo de instanciación se denomina *indirecta*. Por ejemplo, si tenemos una clase "Felino" y una clase "Gato" de tal forma que están relacionadas mediante la relación "es_un" (un gato *es un* felino), cuando se genera una instancia de "Gato", resultará ser una instancia directa de "Gato" y una instancia indirecta de "Felino".
- *Funciones*: se pueden considerar un tipo especial de relación en la que se obtiene un elemento mediante la realización de una serie de cálculos a partir de varios elementos de la ontología. Por ejemplo, el precio de un objeto en una ontología de artículos de venta puede calcularse sumando el coste del producto más la ganancia que se pretende obtener más los impuestos que es necesario añadirle.
- *Axiomas*: son enunciados o reglas fundamentales sobre las propiedades o condiciones que deben satisfacer las relaciones entre los conceptos de la ontología. Por ejemplo, "Si *A* y *B* son de la clase *C*, entonces *A* no puede ser subclase de *B*".

3.1.3.- Tipos de ontologías

Dependiendo del criterio que sigamos a la hora de clasificar las ontologías tendremos una tipología u otra de las mismas. Una clasificación que se suele proporcionar es aquella que se centra en los distintos tipos de conceptualizaciones de los que puede dar cuenta una ontología. De esta forma tenemos:

- *Ontologías Terminológicas*: suelen consistir en un listado de términos que son utilizados para representar el conocimiento de determinado contexto. Resultan útiles para uniformizar el vocabulario de un dominio que es una de las ventajas de las ontologías señaladas anteriormente.
- *Ontologías de información*: proporcionan información estructural del modo en como los datos de determinado dominio se almacenarían en una base de datos. Su utilidad es proporcionar un método para estandarizar el almacenamiento de información.

- *Ontologías de modelado del conocimiento*: son una conceptualización del conocimiento relativo a determinado contexto.

Otra clasificación interesante es aquella que atiende al alcance de aplicación de las ontologías, es decir, al carácter más general o particular que puede tener una ontología. En este sentido tenemos:

- *Ontologías de dominio*: son ontologías diseñadas e implementadas para dar cuenta de un dominio concreto.
- *Ontologías de tareas*: son ontologías que definen y clasifican las tareas que se pueden realizar en un dominio concreto.
- *Ontologías generales*: son ontologías que no se ciñen a un dominio específico sino que representan conceptos generales como puede ser el tiempo, el espacio, la conducta, etc.

3.2.- Creación de ontologías

3.2.1.- Metodología y diseño de ontologías

Aunque no existe una única metodología adecuada para la creación de ontologías pueden establecerse ciertas pautas generales para este cometido. Conviene señalar desde el principio que para elaborar una ontología siempre habrá distintas alternativas correctas, lo cual explica la existencia de varias ontologías adecuadas y correctas para el mismo dominio. Por este motivo, la creación de una ontología no tiene por que ser un procedimiento definitivo sino que, por regla general, suelen seguirse estrategias iterativas que retroalimentan constantemente la elaboración.

Genéricamente, se podría indicar que una ontología debe ser construida de forma clara y objetiva ya que es la mejor forma de proporcionar a sus futuros usuarios información útil y fiable. Las definiciones que contenga una ontología deben ser completas en el sentido de que deben hacerse explícitas las condiciones necesarias y suficientes de los conceptos definidos. La ontología debe garantizar que las inferencias realizadas sobre ella no den lugar a contradicciones, por este motivo en su elaboración debe insistirse en la consistencia y la coherencia de la misma. Las especializaciones y generalizaciones de una ontología deben ser autocontenidas de tal forma que no sea necesario recurrir a la revisión de definiciones ya existentes. Los conceptos de una ontología deben ser una versión lo más próxima posible a los objetos que pretenden representar, mientras que las relaciones deben aproximarse al máximo a las que, en realidad se dan entre esos objetos. Las clases representantes de conceptos dentro de una ontología deben ser disjuntas; a este hecho se le conoce como *principio de distinción ontológica*. Además las ontologías deben ser preferiblemente modulares y los nombres utilizados en ellas deben estar estandarizados en la medida de lo posible.

Por regla general, las fases por las que habitualmente pasa el proceso de elaboración de una ontología son las siguientes:

- *Establecer el dominio, propósito y alcance*: en esta fase se debe dejar claro el dominio de la ontología, los objetivos de la misma, las posibles preguntas a las que la ontología deberá responder, así como los perfiles de los futuros usuarios y del personal encargado de su mantenimiento. Aunque a lo largo del diseño de la ontología las decisiones tomadas en esta fase pueden cambiar, éstas suponen un punto de partida adecuado que permite limitar el alcance del modelo en cuestión.
- *Considerar la reutilización de ontologías*: es altamente recomendable indagar si existe alguna ontología que represente el conocimiento del dominio con el cual

estamos trabajando. No para evitarnos el trabajo de generar una nueva ontología sino para que la ontología que tratamos de crear pueda interactuar con otras del mismo dominio, para extender o mejorar una ontología existente en el mismo ámbito o simplemente para estandarizar vocabularios entre ontologías.

- *Enumerar términos importantes para la ontología*: es importante en este punto hacer un listado de aquellos términos relevantes dentro de la ontología con el fin de proporcionar una explicación para ellos que será de gran utilidad para los futuros usuarios.
- *Construir la ontología*: en este paso se realizan labores tanto de codificación, es decir, explicitar la ontología en un lenguaje formal, como de integración de ontologías existentes en el caso de reutilización de ontologías. Las labores de codificación conllevan:
    - *Definir las clases*: esta fase consiste en obtener un listado de conceptos que se expresarán en forma de clases mediante el uso de un lenguaje formal. Existes varias estrategias a seguir:
        - *De arriba a abajo*: consiste en comenzar con la definición de los conceptos más generales y seguir progresivamente con los de mayor especificidad.
        - *De abajo a arriba*: consiste en comenzar con los conceptos más específicos para seguir con los más generales y abstractos.
        - *Combinada*: consiste en comenzar con los conceptos más relevantes del dominio, generalizándolos y especializándolos convenientemente después.
    - *Crear la jerarquía taxonómica de las clases*: esta fase puede realizarse de forma simultánea a la definición de clases y consiste en el establecimiento de las relaciones que unirán las clases (generalmente relaciones del tipo "es un" o "tipo de") y configurarán la estructura jerárquica de la ontología.
    - *Definir los roles o propiedades (slots)*: no basta solamente con la definición de las clases para que una ontología esté completa y resulte útil como representación del conocimiento en un determinado dominio. Es necesario definir el esqueleto interno de los conceptos mediante la incorporación de los roles o propiedades en las clases que los representan.
    - *Definir las facetas*: posteriormente deben describirse las características de cada uno de los roles o propiedades de los conceptos como la cardinalidad del rol (número de valores posibles que puede tener), el tipo de dato con el que se puede rellenar (string, número, booleano, etc.), el dominio y el rango, etc.
    - *Crear las instancias de las clases*: se hace rellenando los slots de las clases con los valores reales de los objetos que se pretenden modelar.
- *Evaluación*: se trata de comprobar la integridad y coherencia de la ontología así como su capacidad para ser reutilizada.
- *Documentación y reutilización*: la documentación debe llevarse a cabo al tiempo que se realizan las fases anteriores. Debe hacerse de una forma organizada haciendo uso preferiblemente de técnicas de indexación, con el fin de que la ontología creada pueda reutilizarse.

3.2.2.- Implementación de ontologías

Existen varios lenguajes alternativos para implementar una ontología. El más básico de ellos es RDF (*Resource Description Framework*) que posee una sintaxis basada en XML y se utiliza para la representación de conocimiento sobre recursos en la Web. Con RDF se puede representar información como el nombre del creador, la fecha de creación o el lenguaje en el que está escrito un recurso Web, sin embargo, es un lenguaje tan simple que resulta insuficiente para especificar características comunes de las ontologías como el rango o la cardinalidad de un rol de un objeto, la existencia de propiedad transitiva en determinada relación, etc. Por este motivo, posteriormente se crearon lenguajes basados en RDF con el fin de tratar estos problemas.

Alguno de estos lenguajes es DAML+OIL o DAML-S. El primero de ellos es una extensión de RDF creada con el fin de definir una serie de primitivas que fuesen interesantes para el modelado y que surge de la combinación del lenguaje de ontologías DAML-ONT con el lenguaje OIL. DAML-S fue creado para completar la información que proporciona el WSDL de un Web Service, el cual indica cómo el Web Service realiza determinada tarea, es decir, qué parámetros hay que pasarle, de que tipo son y qué parámetros devuelve. El DAML-S completa esta información indicando qué significa cada parámetro y qué es lo que hace el Web Service en sí.

Pero sin duda, la extensión de RDF más específica para ontologías es el OWL (*Web Ontology Language*). El OWL es un lenguaje de marcas semántico orientado a la definición de ontologías cuyo propósito es ser utilizadas en la Web. Es una mejora del DAML+OIL que fue creado con la intención de convertirse en un estándar para lo que se conoce como *Web Semántica*[16] y que pasaremos a describir a continuación.

3.2.2.1.- OWL

OWL esta formado por tres lenguajes de complejidad creciente que se incluyen unos a otros: OWL-Lite, OWL-DL (Lógica Descriptiva) y OWL-Full. La especificación más simple es OWL-Lite y está orientada a implementaciones de ontologías con poca complejidad. OWL-Lite constituye un sub-lenguaje de OWL-DL que incluye un subconjunto de constructores de éste. OWL-DL está orientado al desarrollo de ontologías de complejidad intermedia y aunque el número de constructores es el mismo que el de OWL-Full, se le han incorporado una serie de restricciones en el uso de éstos con el fin de garantizar, después de haber incorporado motores de inferencia sobre la ontología, ciertas características deseables en el sistema como puede ser el carácter computable del mismo. OWL-Full es la especificación completa de OWL, y dado que no tiene tales restricciones sobre el uso de los constructores, ofrece la posibilidad de representar conocimiento que no sea computable o sobre el que los motores de inferencia puedan realizar inferencias incorrectas.

Algunas de las etiquetas de OWL para la definición de las clases y relaciones de una ontología son las siguientes[17]:

---

[16] Se conoce como *Web Semántica* al proyecto consistente en incorporar metadatos semánticos y ontológicos a la *World Wide Web* con el fin de mejorar la interoperatividad entre los sistemas informáticos involucrados en la Web mediante el uso de *agentes inteligentes*, es decir, mediante el uso de programas informáticos que buscan información sin la intervención de operadores humanos.

[17] Para una visión más completa del lenguaje OWL consúltense las siguientes páginas Web:
http://www.w3.org/2003/08/owl-pressrelease.html.en
http://www.w3.org/2004/OWL/
http://www.w3.org/2001/sw/WebOnt/

- Una clase se define con las etiquetas:

```
<owl:Class rdf:ID="Nombre_de_la_clase">
</owl:Class>
```

- Para definir las propiedades o roles de la clase, dentro de las dos etiquetas anteriores se añade:

```
<owl:intersectionOf rdf:parseType="Collection">
   <owl:Restriction>
      <owl:onProperty rdf:resource="#nombre_de_la_propiedad" />
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
         1
      </owl:cardinality>
   </owl:Restriction>

   … (las <owl:Restriction> se repiten tantas veces como propiedades del
   objeto tengamos)

</owl:intersectionOf>
```

- Para definir una sub-clase de una clase se utliza:

```
<owl:Class rdf:ID=" Nombre_de_la_sub-clase">
   <rdfs:subClassOf rdf:resource="#Nombre_de_la_clase_padre" />
   <owl:disjointWith rdf:resource="#Nombre_de_una_clase_hermana" />

   … (las <owl:disjointWith …> se repiten tantas veces como clases
   hermanas tengamos)

</owl:Class>
```

3.2.3.- Herramientas de desarrollo de ontologías

Al igual que sucede con cualquier otro lenguaje de marcas o de programación, manejar OWL directamente resulta una labor demasiado tediosa. Por eso existen una serie de herramientas que nos facilitan la implementación de ontologías mediante la eliminación de trabajos rutinarios de codificación, la visualización gráfica de lo implementado o el uso de interfaces gráficas de usuario.

3.2.3.1.- Protégé

Protégé es un proyecto OpenSource que permite, de una forma sencilla, editar, desarrollar y mantener ontologías y bases de conocimiento así como constituye una librería que otras aplicaciones pueden utilizar para acceder y utilizar ontologías. Soporta lenguajes como OWL o DAML, entre otros.[18]

3.2.3.2.- Jena

Jena es otro proyecto OpenSource que proporciona un entorno de desarrollo para la elaboración de aplicaciones específicas de la Web Semántica. Esta implementado en Java y soporta lenguajes como RDF o OWL. Además de una API específica para cada uno de estos lenguajes, incluye un motor de inferencia basado en reglas.[19]

---

http://www.w3.org/TR/owl-guide/

[18] Para obtener una información más completa sobre Protégé consúltese la siguiente Web: http://protege.stanford.edu/

[19] Para obtener una información más completa sobre Jena consúltese la siguiente Web: http://jena.sourceforge.net/index.html

3.2.3.3.- DAMLdotnetAPI

DAMLdotnetAPI es una interfaz, basada en el proyecto Jena, para la creación de ontologías en el entorno dotNet. Soporta DAML y RDF proporcionando un parser y un modelo para leer y escribir código en estos lenguajes.[20]

3.3.- Ejemplos de ontologías

3.3.1.- Un ejemplo sencillo

En esta sección pondremos en práctica lo explicado en apartados anteriores respecto a las ontologías. Especificaremos una simple clasificación de vehículos y trataremos de realizar el análisis y desarrollo pertinentes para que tal clasificación pueda ser considerada una ontología en el sentido aquí manejado. Con ello mostraremos las dificultades y complejidad de la creación de ontologías. Veremos cómo, a pesar de la sencillez del ejemplo, surgen problemas y ambigüedades que hay que tratar, buscando la mejor solución posible o la que menos afecte al objetivo final de la ontología.

La clase más abstracta del ejemplo representa al concepto de *vehículo*. Definido de una manera general podríamos indicar que un vehículo es una entidad cuyo objetivo es el desplazamiento de personas, animales o cosas de un lugar a otro. Un vehículo puede ser de dos tipos dependiendo de su tracción:

- *Vehículo de tracción animal*: se caracterizan por el hecho de que el desplazamiento es proporcionado por la fuerza imprimida por personas u otro tipo de animales.
- *Vehículo de tracción mecánica*: se caracterizan por poseer un dispositivo denominado motor que es el que causa el desplazamiento del vehículo.

Para nuestro sistema sólo es necesario especificar dos tipos de vehículos de tracción animal:

- *Carro*: es un vehículo de tracción animal con cuatro ruedas o menos cuya característica principal para nuestro propósito es que carece de pedales.
- *Bicicleta*: es un vehículo de tracción animal con dos ruedas cuyo desplazamiento es provocado mediante la acción de pedales.

Los vehículos de tracción mecánica tienen una tipología más compleja que los vehículos de tracción animal que acabamos de describir. Se dividen de la siguiente forma:

- *Ciclomotor*: es un vehículo de tracción mecánica de dos ruedas con una cilindrada de hasta 50 cc. y una capacidad de un pasajero cuyo fin es el transporte de personas.
- *Motocicleta*: es un vehículo de tracción mecánica de dos ruedas con una cilindrada de más de 50 cc. y una capacidad de dos pasajeros cuyo fin es el transporte de personas.
- *Sidecar*: es un vehículo de tracción mecánica de tres ruedas con una cilindrada de más de 50 cc. y una capacidad de tres pasajeros cuyo fin es el transporte de personas.
- *Coche*: es un vehículo de tracción mecánica de cuatro ruedas cuyo fin es el transporte de personas.

[20] Para obtener una información más completa sobre DAMLdotnetAPI consúltese la siguiente Web: http://www.daml.org/2002/10/dotnetAPI/

- *Furgoneta*: es un vehículo de tracción mecánica de cuatro ruedas cuyo fin es el transporte de mercancías.
- *Camión*: es un vehículo de tracción mecánica de seis o más ruedas cuyo fin es el transporte de mercancías.

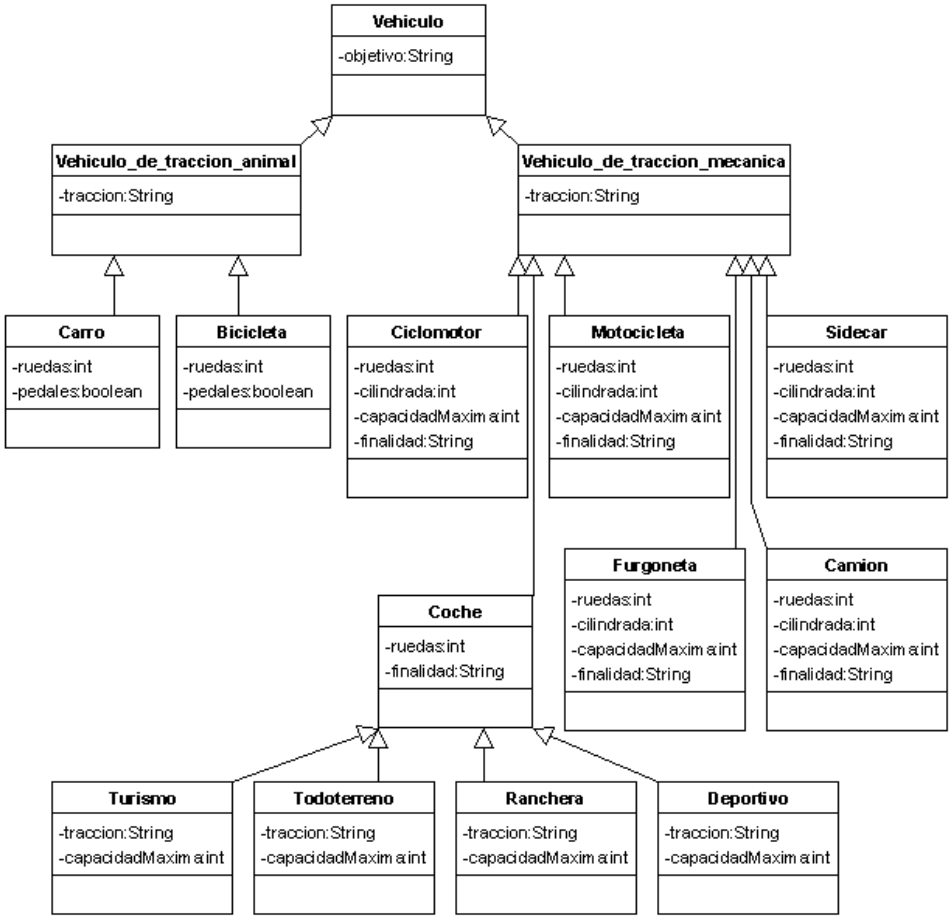
Para nuestro ejemplo, no es necesario desglosar ninguno de los vehículos de tracción mecánica excepto el coche. Se considerarán cuatro tipos de coche:

- *Turismo*: es un coche con tracción fija a dos ruedas cuya capacidad máxima es de cinco personas.
- *Todoterreno*: es un coche con tracción variable a dos o cuatro ruedas cuya capacidad máxima es de ocho personas.
- *Ranchera*: es un coche con tracción fija a dos ruedas cuya capacidad máxima es de ocho personas.
- *Deportivo*: es un coche con tracción fija a dos o cuatro ruedas cuya capacidad máxima es de cuatro personas.

Descrito así nuestro dominio, tenemos un listado de los conceptos que serán utilizados en la ontología, tenemos el árbol que indica cómo se relacionan dichas clases mediante la relación de herencia y podemos extraer cuales son los roles, propiedades o atributos de cada uno de estos conceptos. De esta manera se puede describir el árbol de conceptos con las propiedades correspondientes a cada uno de ellos entre llaves:

- *Vehículo* {objetivo:desplazamiento}
    - *Vehículo de tracción animal* {tracción:animal}
        - *Carro* {ruedas:=<4, pedales:no}
        - *Bicicleta* {ruedas:2, pedales:si}
    - *Vehículo de tracción mecánica* {tracción:mecanica}
        - *Ciclomotor* {ruedas:2, cilindrada:=<50 cc, capacidadMaxima:1, finalidad:transporte de personas}
        - *Motocicleta* {ruedas:2, cilindrada:>50 cc, capacidadMaxima:2, finalidad:transporte de personas}
        - *Sidecar* {ruedas:3, cilindrada:>50 cc, capacidadMaxima:3, finalidad:transporte de personas}
        - *Coche* {ruedas:4, finalidad:transporte de personas}
            - *Turismo* {tracción:fija a dos ruedas, capacidadMaxima:5}
            - *Todoterreno* {tracción:variable a dos o cuatro ruedas, capacidadMaxima:8}
            - *Ranchera* {tracción:fija a dos ruedas, capacidadMaxima:8}
            - *Deportivo*: {tracción:fija a dos o cuatro ruedas, capacidadMaxima:4}
        - *Furgoneta* {ruedas:4, finalidad:transporte de mercancías}
        - *Camión* {ruedas:>=6, finalidad:transporte de mercancías}

    La relación de herencia podría representarse con el siguiente diagrama de clases:



Utilicemos ahora el lenguaje OWL para expresar este pequeño fragmento de conocimiento. El concepto de vehículo se especificaría de la siguiente forma:

```
<owl:Class rdf:ID="Vehiculo">
   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#objetivo" />
         <owl:cardinality rdf:datatype="String">
            1
         </owl:cardinality>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>
```

Los tipos de vehículo, vehículo de tracción mecánica y vehículo de tracción animal se codificarían e OWL de la siguiente manera, teniendo en cuenta que heredan las propiedades de su clase padre vehículo:

```
<owl:Class rdf:ID="Vehiculo_de_traccion_animal">

   <rdfs:subClassOf rdf:resource="#Vehiculo" />
   <owl:disjointWith rdf:resource="#Vehiculo_de_traccion_mecanica" />

   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#traccion" />
         <owl:cardinality rdf:datatype="String">
            1
         </owl:cardinality>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>
```

```
<owl:Class rdf:ID="Vehiculo_de_traccion_mecanica">

   <rdfs:subClassOf rdf:resource="#Vehiculo" />
   <owl:disjointWith rdf:resource="#Vehiculo_de_traccion_animal" />

   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#traccion" />
         <owl:cardinality rdf:datatype="String">
            1
         </owl:cardinality>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>
```

El desglose de los vehículos de tracción animal en carro y bicicleta quedaría de la siguiente forma:

```
<owl:Class rdf:ID="Carro">

   <rdfs:subClassOf rdf:resource="#Vehiculo_de_traccion_animal" />
   <owl:disjointWith rdf:resource="#Bicicleta" />

   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#ruedas" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#pedales" />
         <owl:cardinality rdf:datatype="Boolean">
            1
         </owl:cardinality>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>


<owl:Class rdf:ID="Bicicleta">

   <rdfs:subClassOf rdf:resource="#Vehiculo_de_traccion_animal" />
   <owl:disjointWith rdf:resource="#Carro" />

   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#ruedas" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#pedales" />
         <owl:cardinality rdf:datatype="Boolean">
            1
         </owl:cardinality>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>
```

Los tipos de vehículos de tracción mecánica: ciclomotor, motocicleta, sidecar, coche, furgoneta y camión, se implementarían de la siguiente forma:

```
<owl:Class rdf:ID="Ciclomotor">

   <rdfs:subClassOf rdf:resource="#Vehiculo_de_traccion_mecanica" />
   <owl:disjointWith rdf:resource="#Motocicleta" />
   <owl:disjointWith rdf:resource="#Sidecar" />
   <owl:disjointWith rdf:resource="#Coche" />
   <owl:disjointWith rdf:resource="#Furgoneta" />
   <owl:disjointWith rdf:resource="#Camion" />

   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#ruedas" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#cilindrada" />
```

```
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#capacidadMaxima" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#finalidad" />
         <owl:cardinality rdf:datatype="String">
            1
         </owl:cardinality>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>


<owl:Class rdf:ID="Motocicleta">

   <rdfs:subClassOf rdf:resource="#Vehiculo_de_traccion_mecanica" />
   <owl:disjointWith rdf:resource="#Ciclomotor" />
   <owl:disjointWith rdf:resource="#Sidecar" />
   <owl:disjointWith rdf:resource="#Coche" />
   <owl:disjointWith rdf:resource="#Furgoneta" />
   <owl:disjointWith rdf:resource="#Camion" />

   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#ruedas" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#cilindrada" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#capacidadMaxima" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#finalidad" />
         <owl:cardinality rdf:datatype="String">
            1
         </owl:cardinality>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>


<owl:Class rdf:ID="Sidecar">

   <rdfs:subClassOf rdf:resource="#Vehiculo_de_traccion_mecanica" />
   <owl:disjointWith rdf:resource="#Ciclomotor" />
   <owl:disjointWith rdf:resource="#Motocicleta" />
   <owl:disjointWith rdf:resource="#Coche" />
   <owl:disjointWith rdf:resource="#Furgoneta" />
   <owl:disjointWith rdf:resource="#Camion" />

   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#ruedas" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#cilindrada" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#capacidadMaxima" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#finalidad" />
```

```
         <owl:cardinality rdf:datatype="String">
            1
         </owl:cardinality>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>


<owl:Class rdf:ID="Coche">

   <rdfs:subClassOf rdf:resource="#Vehiculo_de_traccion_mecanica" />
   <owl:disjointWith rdf:resource="#Ciclomotor" />
   <owl:disjointWith rdf:resource="#Motocicleta" />
   <owl:disjointWith rdf:resource="#Sidecar" />
   <owl:disjointWith rdf:resource="#Furgoneta" />
   <owl:disjointWith rdf:resource="#Camion" />

   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#ruedas" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#finalidad" />
         <owl:cardinality rdf:datatype="String">
            1
         </owl:cardinality>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>


<owl:Class rdf:ID="Furgoneta">

   <rdfs:subClassOf rdf:resource="#Vehiculo_de_traccion_mecanica" />
   <owl:disjointWith rdf:resource="#Ciclomotor" />
   <owl:disjointWith rdf:resource="#Motocicleta" />
   <owl:disjointWith rdf:resource="#Sidecar" />
   <owl:disjointWith rdf:resource="#Coche" />
   <owl:disjointWith rdf:resource="#Camion" />

   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#ruedas" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#finalidad" />
         <owl:cardinality rdf:datatype="String">
            1
         </owl:cardinality>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>


<owl:Class rdf:ID="Camion">

   <rdfs:subClassOf rdf:resource="#Vehiculo_de_traccion_mecanica" />
   <owl:disjointWith rdf:resource="#Ciclomotor" />
   <owl:disjointWith rdf:resource="#Motocicleta" />
   <owl:disjointWith rdf:resource="#Sidecar" />
   <owl:disjointWith rdf:resource="#Coche" />
   <owl:disjointWith rdf:resource="#Furgoneta" />

   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#ruedas" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#finalidad" />
         <owl:cardinality rdf:datatype="String">
            1
         </owl:cardinality>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>
```

Finalmente los cuatro tipos de coche: turismo, todoterreno, ranchera y deportivo se codificarían así en OWL:

```
<owl:Class rdf:ID="Turismo">

   <rdfs:subClassOf rdf:resource="#Coche" />
   <owl:disjointWith rdf:resource="#Todoterreno" />
   <owl:disjointWith rdf:resource="#Ranchera" />
   <owl:disjointWith rdf:resource="#Deportivo" />

   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#traccion" />
         <owl:cardinality rdf:datatype="String">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#capacidadMaxima" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>


<owl:Class rdf:ID="Todoterreno">

   <rdfs:subClassOf rdf:resource="#Coche" />
   <owl:disjointWith rdf:resource="#Turismo" />
   <owl:disjointWith rdf:resource="#Ranchera" />
   <owl:disjointWith rdf:resource="#Deportivo" />

   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#traccion" />
         <owl:cardinality rdf:datatype="String">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#capacidadMaxima" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>


<owl:Class rdf:ID="Ranchera">

   <rdfs:subClassOf rdf:resource="#Coche" />
   <owl:disjointWith rdf:resource="#Turismo" />
   <owl:disjointWith rdf:resource="#Todoterreno" />
   <owl:disjointWith rdf:resource="#Deportivo" />

   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#traccion" />
         <owl:cardinality rdf:datatype="String">
            1
         </owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#capacidadMaxima" />
         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
            1
         </owl:cardinality>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>


<owl:Class rdf:ID="Deportivo">

   <rdfs:subClassOf rdf:resource="#Coche" />
   <owl:disjointWith rdf:resource="#Turismo" />
   <owl:disjointWith rdf:resource="#Todoterreno" />
   <owl:disjointWith rdf:resource="#Ranchera" />

   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#traccion" />
         <owl:cardinality rdf:datatype="String">
```

```
                1
             </owl:cardinality>
          </owl:Restriction>
          <owl:Restriction>
             <owl:onProperty rdf:resource="#capacidadMaxima" />
             <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
                1
             </owl:cardinality>
          </owl:Restriction>
       </owl:intersectionOf>
    </owl:Class>
```

Nos quedaría ahora generar instancias de cada una de estas clases. De esta manera podríamos tener objetos como:

- Vehiculo
  objetivo=desplazamiento
- Vehiculo_de_traccion_animal
  traccion=animal
- Carro
  ruedas=4
  pedales=false
- Carro
  ruedas=2
  pedales=false
- Bicicleta
  ruedas=2
  pedales=true
- Vehiculo_de_traccion_mecanica
  traccion=mecanica
- Ciclomotor
  ruedas=2
  cilindrada=49
  capacidadMaxima=1
  finalidad=transporte de personas
- Motocicleta
  ruedas=2
  cilindrada=500
  capacidadMaxima=2
  finalidad=transporte de personas
- Sidecar
  ruedas=3
  cilindrada=750
  capacidadMaxima=3
  finalidad=transporte de personas
- Coche
  ruedas=4
  finalidad=transporte de personas
- Turismo
  traccion=fija a dos ruedas
  capacidadMaxima=5
- Todoterreno
  traccion=variable a dos o cuatro ruedas
  capacidadMaxima=5
- Todoterreno
  traccion=variable a dos o cuatro ruedas
  capacidadMaxima=8
- Todoterreno
  traccion=variable a dos o cuatro ruedas
  capacidadMaxima=6
- Ranchera
  traccion=fija a dos ruedas
  capacidadMaxima=8
- Deportivo
  traccion=fija a cuatro ruedas
  capacidadMaxima=2
- Deportivo
  traccion=fija a dos ruedas
  capacidadMaxima=2
- Deportivo
  traccion=fija a dos ruedas
  capacidadMaxima=4
- Furgoneta
  ruedas=4
  finalidad=transporte de mercancías
- Camion
  ruedas=6
  finalidad=transporte de mercancías
- Camion
  ruedas=8

```
finalidad=transporte de mercancías
```

3.3.2.- Ejemplos conocidos

Toda la temática que ha sido esbozada en este apartado se ha puesto en práctica desde el advenimiento del fenómeno de la Web Semántica, lo que tuvo como consecuencia la creación de un buen número de ontologías. Por regla general, las ontologías creadas tienen un propósito concreto y suelen dar cuenta del conocimiento en un dominio determinado (por ejemplo, en medicina, en contextos empresariales, en aeronáutica, etc.). Sin embargo, también se han creado ontologías de propósito general que tratan conocimiento que puede ser útil en diversos contextos. A continuación se enumerarán unas cuantas ontologías de ambos tipos aunque se debe tener en cuenta que la lista no es exhaustiva.

Uno de los primeros ejemplos lo podemos encontrar en *SNOMED*. SNOMED es una ontología específica del ámbito de la medicina desarrollada con el objetivo de facilitar de una forma segura y eficaz el intercambio de información sobre cuestiones sanitarias. Está considerada como la ontología multilingüe más intuitiva e inteligible sobre temas de salud.[21]

En el ámbito de la lingüística, *WordNet* ha sido, desde su creación, una referencia relevante. WordNet es una ontología que organiza nombres, adjetivos y verbos en grupos de sinónimos. Inicialmente fue construida para el idioma inglés pero el proyecto se ha extendido a otros idiomas.[22]

Una ontología de carácter general que puede resultar de interés es *UNSPSC*. Desarrollada bajo el auspicio de las Naciones Unidas ofrece un sistema global de clasificación que puede ser útil para muchas empresas como herramienta de análisis y optimización de costes o para la explotación de las capacidades del comercio electrónico mediante la clasificación e identificación de artículos de venta.[23]

Otra ontología de propósito general es *Ontolingua*. Su cometido es proporcionar un entorno colaborativo distribuido no sólo para navegar a través de ontologías sino también para crearlas, modificarlas y utilizarlas.[24]

Finalmente, el lector interesado puede consultar *DAML Ontology Library*, que proporciona un listado de un buen número de ontologías desarrolladas en el lenguaje DAML y organizadas por diversos criterios de búsqueda (http://www.daml.org/ontologies/).

4.- Ejercicios

1. Formaliza, mediante el uso de un lenguaje formal para una Lógica de Primer Orden, la clasificación o taxonomía relativa al reino animal que figura en http://www.monografias.com/trabajos10/cani/cani.shtml teniendo en cuenta que:

[21] Para obtener más información sobre SNOMED consúltese la siguiente Web: http://www.ihtsdo.org/

[22] Para obtener más información sobre WordNet consúltese la siguiente Web: http://wordnet.princeton.edu/

[23] Para obtener más información sobre UNSPSC consúltese la siguiente Web: http://www.unspsc.org/

[24] Para obtener más información sobre Ontolingua consúltese la siguiente Web: http://www.ksl.stanford.edu/software/ontolingua/

a. La relación que une los distintos nodos de la clasificación debe ser transitiva.
b. La relación que une los distintos nodos de la clasificación debe ser asimétrica.

2. Diseña una red semántica que represente las relaciones entre las palabras de un fragmento de un diccionario de sinónimos teniendo en cuenta que:
   a. Las palabras pueden ser polisémicas, es decir, pueden tener varios significados o acepciones.
   b. Las palabras pueden ser homógrafas, es decir, dos palabras se pueden escribir de la misma manera pero ser palabras completamente distintas con significados completamente dispares (Por ejemplo, la palabra "desierto" en el ejemplo que figura abajo).
   c. Puede darse el caso de que una palabra figure como sinónima de una entrada del diccionario pero no figurar como entrada del mismo.
   d. Ejemplo de fragmento de diccionario de sinónimos:

```
abandonado>1>[“dejado”,”descuidado”,”desidioso”,”negligente”,”desamparado”]
abandonado>2>[“desaseado”,”desaliñado”,”sucio”,”ir hecho un pordiosero”]
abandonado>3>[“deshabitado”,”inhabitado”,”despoblado”,”desierto”,”yermo”]
abandonado>4>[“desvalido”,”desamparado”]

adán>1>[“dejado”,”desaseado”,”desaliñado”,”sucio”]

dejado>1>[“negligente”,”perezoso”,”descuidado”,”desidioso”,”abandonado”,”indolente”]
dejado>2>[“desaseado”,”desaliñado”,”sucio”,”adán”]
dejado>3>[“apático”,”impasible”,”indiferente”,”indolente”,”desidioso”,”abandonado”]

desaliñado>1>[“abandonado”,”desaseado”,”sucio”,”hecho un pordiosero”]

desamparado>1>[“abandonado”,”dejado”,”descuidado”,”desvalido”]

desaseado>1>[“descuidado”,”dejado”,”desaliñado”,”sucio”,”adán”,”estropajoso”]

descuidado>1>[“dejado”,”negligente”,”desidioso”,”abandonado”]
descuidado>2>[“desaliñado”,”desaseado”,”adán”]
descuidado>3>[“desapercibido”,”desprevenido”]

deshabitado>1>[“inhabitado”,”abandonado”,”despoblado”,”desierto”,”yermo”,”solitario”]

desidioso>1>[“abandonado”,”dejado”,”descuidado”,”negligente”,”desamparado”]

desierto1>1>[“yermo”]

desierto2>1>[“despoblado”,”deshabitado”,”inhabitado”,”solitario”]

despoblado>1>[“desierto”,”yermo”,”deshabitado”,”inhabitado”]

desvalido>1>[“desamparado”,”abandonado”]

inhabitado>1>[“yermo”,”deshabitado”,”despoblado”]

negligente>1>[“abandonado”,”dejado”,”desidioso”,”descuidado”,”indolente”]

sucio>1>[“manchado”,”impuro”,”sórdido”]
sucio>2>[“inmundo”,”puerco”,”cochino”,”desaseado”]
sucio>3>[“obsceno”,”deshonesto”]

yermo>1>[“inhabitado”,”deshabitado”,”despoblado”,”desierto”]
yermo>2>[“inculto”]
```

3. Dado el siguiente texto, crear un sistema de marcos que capte el conocimiento encerrado en él:

   Una **silla** es un mueble cuya finalidad es servir de asiento a una sola persona, aunque no se debe confundir con un **taburete** que también es un asiento para una persona, pero sin brazos ni respaldo. Suele tener cuatro patas, aunque puede haber de una, dos, tres o más. Pueden estar

elaboradas en diferentes materiales: madera, hierro, forja, plástico. Según su diseño puede ser clásica, rústica, moderna, o de oficina.

Existen diversos tipos de sillas. Las que cuentan con reposabrazos se denominan **sillones**. Pero además de éstos, existen algunos tipos más que vamos a enumerar.

Por un lado, estarían las sillas utilizadas por niños, entre las que se destacan:

- **Silla de coche**. Silla para transportar niños dentro de los coches, normalmente de plástico. Se fijan por medio del cinturón de seguridad y pueden disponerse de frente o de espaldas.
- **Trona**. Silla con repisa frontal utilizadas para que los niños puedan comer.
- Por el otro lado, estarían las sillas utilizadas por adultos, entre las que se destacan:
- **Silla plegable**. Silla generalmente de madera que se pliega ocupando muy poco espacio.
- **Silla de tijera**. Silla plegable con asiento y respaldo de tela y patas en aspa.
- **Silla de oficina**. Silla que se utiliza en las oficinas, con asiento regulable en altura y respaldo reclinable para adaptarla a las características de cada persona.
- **Sillón mecedora**. Silla típica de madera, con un borde curvado en la parte inferior que permite que se mueva adelante y atrás.

4. Descarga la herramienta de manejo y desarrollo de ontologías Protégé de http://protege.stanford.edu/download/download.html e implementa:
    a. La taxonomía del ejercicio 2.
    b. La red semántica del ejercicio 3.

Referencias

Begeer, S.; Chrisley, R. (2000): *Artificial intelligence: critical concepts, Volumen 2*. Taylor and Francis, Great Britain.

Brachman, R. J. (1983): 'What is-a is and isn't: An analysis of taxonomic links in semantic networks'. *Computer*, 16(10), pp. 30–36.

Burkert, C. (1995): 'Lexical semantics and terminological knowledge representation'. *Computational lexical semantics*, Cambridge University Press, pp. 165-184.

Davis, R.; King, J. J. (1984): 'The origin of rule-based systems in A.I.'. En Buchanan B. G.; Shortliffe, E. H. (eds.). *Rule Based Expert Systems*. Addison-Wesley, pp. 20-53.

Diekhoff, G. M.; Diekhoff, K. B. (1982): 'Cognitive maps as a tool in communicating structural knowledge', *Educational Technology*, vol. 22, 4, pp. 28–30.

Elmasri, R.; Navathe, S. B. (1997): *Sistemas de Bases de Datos*. Addison Wesley Longman, 2ª edición

Fernández Fernández, G. (2004): *Representación del conocimiento en sistemas inteligentes*, Ciberlibro Creative Commons.

Gruber, T. R. (1993-1): 'A Translation Approach to Portable Ontology Specification', *Knowledge Acquisition*, 5, 2, pp. 199.220.

Gruber, T. R. (1993-2): 'Toward principles for the design of ontologies used for knowledge sharing'. En Guarino, N.; Poli, R. (eds.): *Formal Ontology in Conceptual Analysis and Knnowledge Representation*. Kluwer, 1993.

Guarino N.; Giaretta P. (1995): 'Ontologies and knowledge bases'. En Mars, N. J. I. (ed.): *Towards Very Large Knowledge Bases*. IOS Press, Amsterdam, 1995, pp. 25-32.

Hanson, S.; Bauer, M. (1989): 'Conceptual clustering, categorization, and polymorphy', *Machine Learning*, 3, 4, pp. 343-372.

Hayes, P. J. (1979): 'The logic of frames'. En Metzing, D. (ed.), *Frame Conceptions and Text Understanding*, pp 46–61. Walter de Gruyter and Co., Berlin, Germany.

Jonassen, D. H. (1985): *The technology of text. Volume 2: Principles for structuring, designing and displaying text*, Educational Technology Publications, Englewood Cliffs NJ.

Laurière, J. L. (1982): 'Représentation et utilisation des connaissances-première partie: Les systèmes experts'. *Technique et Science Informatiques*, volume 1, pp. 25–42.

Luger, G. F. (2005): *Artificial intelligence: Structures and strategies for complex problem solving*. Addison-Wesley, England.

Michel F. (1990): 'Quést-ce qún expert? connaissances procédurale et déclarative dans línteraction médicale', *Réseaux*, volume 8, pp. 59–80.

Minsky, M. (1974): 'A framework for representing knowledge'. Technical report, Cambridge, MA, USA.

Negnevitsky, M. (2005): *Artificial intelligence: a guide to intelligent systems (2ª Edition)*. Pearson Education.

Newell, A (1981): 'The knowledge level', *AI Magazine*, 2, pp. 1–20.

Nilsson, N. J. (1982): *Principles of Artificial Intelligence*. Springer, Berlin, Heidelberg.

Noy, N. F.; McGuinness, D. L. (2001): *Ontology Development 101: A Guide to Creating Your First Ontology*. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880.

Partridge D.; Wilks, Y. (1990): *The Foundations of artificial intelligence: a sourcebook*. Cambridge University Press, Great Britain.

Platón (2003): *Diálogos*, Vol. V, Madrid, Editorial Gredos.

Quillian, M. R. (1967): 'Word concepts: a theory and simulation of some basic semantic capabilities', *Behavioral Science*, 12(5), pp. 410–430.

Russell, S.; Norvig, P. (2002): *Artificial Intelligence: A Modern Approach*, Prentice Hall. 2ª edición.

Ryle, G. (1946): 'Knowing how and knowing that', *Proceedings of the Aristotelian Society*. 46, S. 1-16.

Ryle, G. (1949): *The concept of mind*. The University of Chicago Press, Chicago.

Schank, R. C. (1975): *Conceptual Information Processing*. Elsevier Science Inc., New York.

Schank, R. C.; Kolodner, J. L.; DeJong, G. (1980): 'Conceptual information retrieval'. *SIGIR'80: Proceedings of the 3rd annual ACM conference on Research and development in information retrieval*, pp. 94–116.

Shafer, G. (1976): *A Mathematical Theory of Evidence*. Princeton University Press.

Shapiro, S. C.; Eckroth, D. (1987): *Encyclopedia of artificial intelligence*. New York : Wiley.

Shortliffe, E. H.; Buchanan, B. G. (1975): 'A model of inexact reasoning in medicine', *Mathematical Biosciences,* 23, pp. 351-379.

Simon, H. A. (1981): 'Otto Selz: His contribution to psychology'. En Frijda, N. H.; de Groot A. (Eds.) *Otto selz and information processing psychology*, Mouton De Gruyter, pp. 147-164.

Sowa, J. F. (1983): *Conceptual Structures: Information Processing in Mind and Machine*. Systems Programming Series. Addison-Wesley.

Thomason, R. H.; Touretzky, D. S. (1991): 'Inheritance theory and networks with roles'. Sowa, J. F. (ed.): *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, Kaufmann, San Mateo, pp. 231-266.

Touretzky, D. S. (1986): *The Mathematics of Inheritance Systems*. Morgan Kaufmann.

Vianu, V. (1997) 'Rule-based languages'. *Annals of Mathematics and Artificial Intelligence,* 19, 1-2, pp. 215-259.

Zadeh, L. (1965): 'Fuzzy sets'. *Information and Control*. 8. pp. 338–353.

Programación lógica e inteligencia artificial

M. Vilares[1], V.M. Darriba[1], C. Gómez-Rodríguez[2], and J. Vilares[2]

[1] Department of Computer Science, University of Vigo
Campus As Lagoas s/n, 32004 Ourense, Spain
{vilares,darriba}@uvigo.es
[2] Department of Computer Science, University of A Coruña
Campus de Elviña s/n, 15071 A Coruña, Spain
{cgomezr,jvilares}@udc.es

1 Introducción

Si preguntásemos a un usuario experto, que encuentra en la programación lógica y que echa de menos en otros paradigmas de programación, probablemente su respuesta se basase en algunas ideas extremadamente sencillas. La primera podría ser la práctica ausencia de sintaxis impuesta por el lenguaje. Unos pocos operadores lógicos y la noción de árbol dejan toda la fuerza expresiva en manos de la lógica de Horn.

Otro factor a menudo esgrimido en favor de la programación lógica es su potencia de cálculo, fruto de la combinación de los mecanismos de unificación y resolución, simples, eficaces y transparentes al programador.

En conjunto, estas características permiten centrar nuestra atención en los aspectos propios al desarrollo algorítmico, que denominamos declarativos, limitando el esfuerzo de programación asociado a factores operacionales; al contrario de lo que ocurre en los lenguajes imperativos. En la práctica, ello se traduce en un código más compacto, lo que ha contribuido a extender la idea de que unas pocas líneas de Prolog equivalen habitualmente a páginas enteras de código tradicional.

Desde un punto de vista más concreto, la programación lógica define un tratamiento extraordinariamente flexible de símbolos y estructuras, a la vez que permite poner en marcha de forma sencilla y rápida estrategias deductivas. Todo ello hace que a menudo se le considere como un paradigma de programación especialmente adaptado al tratamiento de problemas en el dominio de la inteligencia artificial.

En este capítulo repasaremos los conceptos básicos de la programación lógica, que ilustraremos mediante ejemplos prácticos cuyo objetivo será el de transmitir al lector el particular estilo de programación requerido si queremos exprimir todas las posibilidades asociadas a la combinación de la unificación y la resolución lógica. Para satisfacer este objetivo, el texto se estructura en secciones de temática diferenciada. Las primeras ideas, ilustradas con el ejemplo ya clásico de la gestión de relaciones genealógicas, se introducen en la sección 2. Se utiliza un lenguaje simple y no especializado, buscando orientar la intuición del lector hacia un paradigma de cálculo que imprime un estilo propio y no generalizable de

programación, algo nuevo y apasionante. Las especificaciones léxicas y sintácticas del lenguaje son el objeto de la sección 3, mientras que una introducción rápida a la semántica del mismo es presentada en la sección 4. A este respecto, hacemos especial hincapié en la diferenciación de los conceptos de semántica declarativa y procedural, abordando explícitamente el origen y tratamiento de las incongruencias declarativo/procedurales más comunes en programación lógica. La siguiente sección 5 introduce en detalle la utilización de los predicados de control típicos en PROLOG, a saber, el corte y el fallo. Ello a su vez sirve de base a la presentación de la forma común de tratamiento de la negación en este tipo de intérpretes, la negación por fallo, cuyas peculiaridades de tratamiento serán comentadas más adelante. Las listas y su estructura típicamente recursiva son el tema de la sección 6. En la sección 7 se presentan los predicados básicos que sirven para implementar el concepto de evaluación perezosa en PROLOG, permitiendo diferir la ejecución de objetivos hasta qus se satisfagan una serie de condiciones. También se muestra como el uso de esta técnica permite solventar algunas de las incongruencias entre semántica operacional y declarativa, en particular en lo tocante al tratamiento de la negación por fallo y de la recursividad izquierda. Posteriormente, en la sección 8 se explica cómo definir dinámicamente operadores y su posible uso en la contrucción de interfaces y programas más cercanos al lenguaje natural. Finalmente, en la sección 9 se atiende a la problemática de construír programas que hacen uso del aprendizaje automático a través de la modificación dinámica del conocimiento almacenado en la memoria del intérprete en ejecución. Un conjunto amplio de ejercicios propuestos, cubriendo toda la temática abordada, constituye la sección 10.

2 Un primer contacto

En programación lógica, los *programas* representan objetos y relaciones entre éstos codificadas mediante fórmulas que habitualmente denominamos *cláusulas*. De esta forma, un programa no es sino una base de conocimiento a partir de la cual se pueden contestar preguntas en un proceso de interpretación interactivo. Se trata, en esencia, de localizar aquellos objetos cuyo comportamiento satisface las relaciones establecidas. De este modo, un intérprete PROLOG puede ser entendido como un motor de inferencia que, partiendo de preguntas (objetivos a demostrar), intenta alcanzar los axiomas del programa mediante un conjunto de reglas de inferencia básicas que aplicamos al conocimiento acumulado en las fórmulas del programa.

2.1 La construcción de programas

Supongamos que queremos representar relaciones de parentesco entre miembros de la misma familia a través de un programa PROLOG. Una posibilidad sería la siguiente:

```
hombre("Juan").
```

```
hombre("Manuel").

mujer("Monica").
mujer("Ana").
mujer("Susana").

progenitor("Manuel","Monica").
progenitor("Manuel","Ana").
progenitor("Monica","Juan").
progenitor("Monica","Susana").
```

En este caso, los objetos que se pretenden relacionar son los nombres de las personas implicadas, expresados como cadenas de caracteres delimitadas por comillas dobles. Las relaciones entre los objetos manipulados se expresan, en este caso, mediante notaciones simples cuyo significado establece el propio programador y en relación al cual éste debe ser congruente en la posterior redacción del programa. Toda fórmula marca su final mediante un punto, indicando de este modo su independencia de las demás. Así, por ejemplo, si establecemos que la notación `hombre(Individuo)` debe interpretarse con la semántica *"Individuo es un hombre"* y que `progenitor(Hombre, Persona)` significa que *"Hombre es el progenitor de Persona"*, tendremos que las claúsulas:

```
hombre("Manuel").
progenitor("Manuel","Ana").
```

nos dicen que, respectivamente, Manuel es un hombre y que Manuel es el padre de Ana. Además, estas fórmulas son incondicionales en cuanto a su verificación, esto es, no dependen de relaciones establecidas por otras fórmulas. A saber, son axiomas. Sin embargo, los razonamientos pueden ser más complejos e incluir la verificación de condiciones para alcanzar el valor de prueba. En este sentido, los programas lógicos incluyen expresiones condicionales simples donde la conclusión se separa de las premisas a través del conectivo `":-"`. Un ejemplo simple serían las cláusulas siguientes, que podrían enriquecer nuestro programa original, pero cuya inclusión desecharemos por falta de generalidad, como comentaremos a continuación:

```
padre("Manuel","Ana"):- hombre("Manuel"),
                        progenitor("Manuel","Ana").
madre("Monica","Juan"):- mujer("Monica"),
                         progenitor("Monica","Juan").
```

cuyo significado es, respectivamente, que Manuel es el padre de Ana y Mónica la madre de Juan. Observar también que, por ejemplo, el hecho de que Manuel sea padre de Ana se condiciona a que Manuel sea hombre y a su vez progenitor de Ana.

Sin embargo, estas cláusulas conllevan una limitación fundamental en cualquier protocolo de razonamiento automático, lo que justificará su no

consideración en nuestro programa de gestión genealógica. Pura y simplemente no incluye el concepto de variable y, por tanto, excluye la posibilidad de generalizar las relaciones a un conjunto de objetos que, compartiendo cierto número de propiedades, no sean idénticos. Un ejemplo claro es el representado por las relaciones `padre` y `madre` que acabamos justo de definir. En efecto, las dos cláusulas consideradas son casos particulares que implican exclusivamente a los individuos Manuel y Ana por una parte, y a Mónica y Juan por otra. Nada podemos, en este sentido, decir acerca de la maternidad de Mónica en relación a Susana, cuando a partir de la información disponible en la base de datos resulta evidente la veracidad de la misma.

En este sentido, la introducción de variables en programación lógica se hace atendiendo, como en cualquier otro conjunto de fórmulas, a criterios de localidad a la propia fórmula y a su cuantificación universal en su ámbito de aplicación. Desde el punto de vista de la notación, cualquier identificador que comience por mayúscula es considerado una variable. De este modo, y siguiendo nuestro ejemplo, las relaciones de paternidad y maternidad deducibles a partir de la base de datos inicial pueden resumirse en las dos fórmulas siguientes:

```
padre(X,Y):-hombre(X),progenitor(X,Y).
madre(X,Y):-mujer(X),progenitor(X,Y).
```

Así, la primera regla significa que `padre(X,Y)` se verifica cuando también se cumplen `hombre(X)` y `progenitor(X,Y)`. Esto es, han de cumplirse las condiciones de que `X` sea un hombre y, además, que sea el progenitor de `Y`. En ese caso, entonces habremos demostrado que `X` es el progenitor de `Y`. Tanto los valores de `X` como los de `Y` son libres.

Aún tratándose de un ejemplo muy sencillo, las ideas que acabamos de exponer suponen un planteamiento de la actividad de programación radicalmente diferente al que marcan los paradigmas imperativo y funcional. En este sentido, el paradigma lógico delimita un estilo propio perfectamente identificable, basado en la inferencia de relaciones a partir de fórmulas de ámbito universal. Sin embargo, las particularidades del paradigma lógico no acaban aquí, sino que afectan igualmente al modo de interrogar a los programas. En efecto, tanto en el paradigma imperativo como en el funcional los papeles de argumentos de entrada y salida están fijados por el usuario de forma estática. No ocurre lo mismo en el lógico, mucho más flexible y, por tanto, potente desde el punto de vista expresivo.

2.2 La interrogación de programas

Notacionalmente, una *pregunta* es una fórmula lógica sin cabeza, que interpretamos declarativamente como una interrogación sobre las condiciones en las que los términos implicados son ciertos. Asumiendo, como es el caso, que en la mayoría de los intérpretes lógicos el *prompt* del sistema es `"?-"`. De esta forma, para preguntar si Manuel es el padre de Mónica, deberíamos escribir en la ventana del propio intérprete:

```
?- padre("Manuel","Monica").
```

y dado que tenemos como axiomas del programa a `hombre("Manuel")` y `progenitor("Manuel","Monica")`, el sistema responderá de forma afirmativa.

Sin embargo, el uso de variables se hace también extensible a estas preguntas, sin restricciones para el usuario, lo que otorga una flexibilidad de interrogación desconocida en otros paradigmas. Así, por ejemplo, la pregunta:

```
?- padre("Manuel",X).
```

se interpreta en clave de cuáles son los valores de la variable `X` para los cuales el término `padre("Manuel",X)` es cierto a partir de la información disponible en el programa; si es que dichos valores existen. En caso afirmativo, el intérprete proporcionará dichos valores. En otro, simplemente responderá de forma negativa. En el caso particular que venimos de considerar, obtendremos que los valores `X = Mónica` y `X = Ana` satisfacen la pregunta inicial[3], puesto que existe información suficiente en el programa para afirmar que ambas son hijas de Manuel.

En caso de interesarnos conocer quién es el padre de Mónica, la pregunta sería en la forma:

```
?- padre(X,"Monica").
```

obteniéndose una sola solución en este caso, dada por `X = Manuel`. Podemos, incluso interesarnos por el conjunto de relaciones padre/hijo deducibles a partir del programa inicial, para lo cual lo único que tendremos que considerar es la pregunta:

```
?- padre(X,Y).
```

que nos facilitará todas las combinaciones correctas posibles para valores de las variables `X` e `Y`. También podemos interesarnos por objetos que satisfacen varias relaciones a la vez. Si quisiéramos saber quien es la hija de Manuel y, a su vez, madre de Juan, escribiríamos:

```
?- padre("Manuel",X), madre(X,"Juan").
```

obteniendo en este caso el resultado `X = "Monica"`. De modo similar podemos interrogar al programa acerca de la verificación de una u otra relación. Por ejemplo, si desearamos establecer que valors de `X` e `Y` verifican una de las dos relaciones `padre("Manuel",X)` o `madre(X,"Juan")`, la pregunta se formularía del modo siguiente:

```
?- padre("Manuel",X); madre(X,"Juan").
```

asociando el valor de la disyunción al símbolo `";"`, de modo análogo a como hasta ahora hemos asociado al símbolo `","` la semántica de la conjunción.

[3] una vez obtenida una solución, los intérpretes lógicos requieren de la entrada por teclado de un símbolo `";"` para proporcionar la siguiente respuesta, si es que ésta existe.

2.3 La recursividad

Como no podía ser de otra forma, el paradigma de cálculo lógico hace un uso extensivo de la recursividad, lo que permite expresar dependencias relacionales no inmediatas entre objetos. Supongamos que, a partir de la información genealógica inicial y de los nuevas cláusulas introducidas para determinar la maternidad y paternidad de los individuos, queremos determinar quienes son los ancestros o descendientes de cualquier persona de la familia. Podríamos considerar las siguientes reglas a incluir en nuestro programa:

```
ancestro(X,Y):-progenitor(X,Y).
ancestro(X,Y):-progenitor(X,Z),ancestro(Z,Y).
```

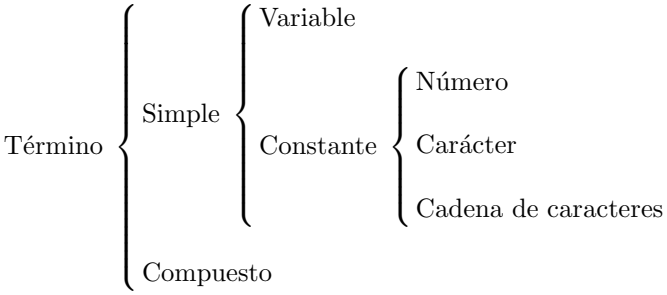
cuya interpretación es inmediata asumiendo que la notación `ancestro(X,Y)` expresa que `X` es ancestro de `Y`. En efecto, estamos diciendo que `X` es ancestro de `Y` es su progenitor, o bien si se puede establecer que `X` es progenitor de `Z`, donde `Z` a su vez es un ancestro de `Y`. De este modo, por ejemplo, ante la pregunta:

```
?- ancestro("Manuel",Y).
```

obtendríamos las respuestas `Y = "Monica"`, `Y = "Ana"`, `Y = "Juan"` y `Y = "Susana"`.

3 Léxico y sintaxis

La clase más general de objeto manejado en programación lógica es el *término*. La jerarquía de datos a partir de dicha estructura viene indicada por el siguiente diagrama:



Las *variables* suelen indicarse mediante un identificador cuyo primer carácter es una letra mayúscula. Los *términos compuestos* son los objetos estructurados del lenguaje. Se componen de un *funtor*[4] y una secuencia de uno o más términos llamados *argumentos*. Un funtor se caracteriza por su *firma*, a saber, el par formado por su *nombre*, que es un átomo, y su *aridad* o número de argumentos. Cuando el término compuesto no expresa ningún tipo de relación lógica[5],

[4] llamado el *funtor principal del término.*

[5] ello dependerá tan solo del contexto expresado por el programa lógico.

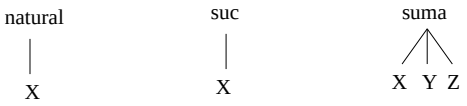
suele dársele el nombre de *función*. Cuando si expresa ese tipo de relación, lo denominamos *predicado*. Una *constante lógica* no es otra cosa que una función de aridad cero.

Estructuralmente, un objeto se representa mediante un árbol que responde al esquema indicado por su firma, y en el que la raíz está etiquetada con el nombre del funtor y los hijos con los argumentos del mismo. Desde el punto de vista semántico, un término establece una relación entre el funtor y el conjunto de sus argumentos. Esta relación, junto con el término al que se asocia, viene fijada por el programador y constituirá el elemento básico dotado de significado de un programa, es lo que denominaremos *átomo*.

Ejemplo 31 *Consideraremos tres términos diferentes, uno asociado al concepto de* "número natural"*, otro al concepto de* "número siguiente a otro" *y el tercero asociado a la noción de* "suma de dos números para obtener un tercero"*. Resumimos el conjunto de firmas y relaciones definidas en el diagrama:*

Sintaxis	**Semántica**	**Firma**
natural(X)	*X es un número natural*	*natural/1*
suc(X)	*El número que sigue a X*	*suc/1*
suma(X,Y,Z)	*Z = X + Y*	*suma/3*

términos que estructuralmente se corresponden con el conjunto de árboles que sigue



y que determinarán los procesos de unificación y resolución que introduciremos más tarde.

Intuitivamente, el lector puede pensar en este tipo de términos como si de un *tipo registro* en los lenguajes imperativos se tratara. De hecho, el principio de gestión es el mismo con la salvedad de que aquí la recuperación y asignación de lo que serían los campos del registro no se realiza mediante funciones de acceso, sino utilizando el concepto de *unificación* que introduciremos más adelante.

La unidad fundamental de un programa lógico es el *átomo*, un tipo especial de término compuesto, distinguido sólo por el contexto en el cual aparece en el programa[6]. A partir del concepto de átomo, definimos el de *literal*, que no es otra cosa que un átomo o su negación.

En este punto, podemos ya introducir la noción de *cláusula* como un conjunto de literales ligados por conectores y cuantificadores lógicos. En nuestro caso, nos limitaremos a un tipo especial de cláusulas denominadas de Horn. Una *cláusula de Horn* es una estructura compuesta por una *cabeza* y un *cuerpo*. La cabeza consiste o bien de un simple átomo, o bien está vacía. El cuerpo está formado por una secuencia de cero o más literales ligados mediante conectivas conjuntivas o

[6] esto es, expresa una relación lógica.

disyuntivas. El final del cuerpo suele indicarse mediante un punto, y todas las variables de la cláusula están cuantificadas universalmente y son locales a la misma. Cuando la cabeza está vacía se dice que la cláusula es una *pregunta*.

La cabeza y el cuerpo de una misma cláusula están separados por un símbolo de implicación lógica cuya representación notacional puede variar según el autor considerado. Esto es, en general representaremos una cláusula en la forma :

$$P \ :- \ Q_1, Q_2, \ldots, Q_n.$$

que podemos leer bien *declarativamente*:

"P es cierto si Q_1 es cierto, Q_2 es cierto, ..., y Q_n es cierto."

bien *operacionalmente*:

"Para satisfacer P, es necesario satisfacer los átomos Q_1, Q_2, ..., y Q_n."

donde el átomo de cabeza y los literales de la cola suelen referirse como *objetivos*. En este contexto, un *programa lógico* se define simplemente como una secuencia de cláusulas de Horn.

Ejemplo 32 *El siguiente programa define recursivamente el conjunto de los números naturales:*

```
natural(0).
natural(suc(X)) :- natural(X).
```

Se puede observar que hemos utilizado los términos `natural` *y* `suc` *previamente definidos en el ejemplo 31, donde resulta obvio que* `suc` *es una simple facilidad notacional que no expresa ningún tipo de relación lógica, en contra de lo que ocurre con el predicado* `natural/1`*. Por tanto,* `suc` *refiere a una función y no a un predicado, que si sería el caso de* `natural`*. Declarativamente, podemos leer ambas cláusulas en la forma:*

"El cero es natural."
"El número `X` es natural, si `X` también lo es."

Observar que la notación considerada para los números naturales es puramente simbólica y basada en la función `suc`*, dada por la tabla 1 que sigue:*
Ello nos permitirá, más tarde, explotar plenamente la potencia deductiva de la unificación y de la resolución. Como preguntas para este programa podemos considerar, por ejemplo, las siguientes:

```
:- natural(suc(0)).      :- natural(0).
:- natural(suc(suc(X))). :- natural(X).
```

cuya interpretación declarativa es, respectivamente, la siguiente:

¿ es suc(0) natural ? ¿ es 0 natural ?
¿ existe X, tal que suc(suc(X)) sea natural ? ¿ existe X, tal que sea natural ?
Observar que las dos últimas preguntas tienen una infinitud de posibles respuestas.

Natural	Notación
0	`0`
1	`suc(0)`
2	`suc(suc(0))`
⋮	⋮
n	`suc(`$\overset{n}{\ldots}$`(suc(0))...)`

Table 1. Notación para la representación de los números naturales.

4 Semántica

Una vez introducida brevemente la sintaxis y el léxico utilizados en programación lógica, necesitamos ahora dotar de significado a los programas. Lo haremos desde dos puntos de vista complementarios, aunque no los únicos posibles, las semánticas declarativa y procedural. En el primer caso centraremos nuestra atención en los programas como teorías en lógica de primer orden; mientras que en el segundo lo haremos en relación a la forma en que explícitamente se resuelven las preguntas. Formalmente, necesitamos primero introducir algunos conceptos previos, básicos para el entendimiento de la mecánica del motor de un intérprete lógico y que determinan inequívocamente el estilo de programación propio de los lenguajes de este tipo.

Definición 41 *Una* sustitución *es una lista de pares* (variable, término)*. Utilizaremos la notación*

$$\Theta \equiv \{X_1 \leftarrow T_1, \ldots, X_n \leftarrow T_n\}$$

para representar la sustitución Θ que asocia las variables X_i a los términos

$$T_i,\ i \in \{1, 2, \ldots, n\}$$

La aplicación de una sustitución Θ a un término lógico T será denotada $T\Theta$ y se dirá que es una instancia *del término T.*

4.1 Semántica declarativa

Ahora ya podemos considerar una definición formal recursiva de la *semántica declarativa* de las cláusulas, que sirve para indicarnos aquellos objetivos que pueden ser considerados ciertos en relación a un programa lógico:

> *"Un objetivo es cierto si es una instancia de la cabeza de alguna de las cláusulas del programa lógico considerado y cada uno de los objetivos que forman el cuerpo de la cláusula instanciada son a su vez ciertos."*

En este punto, es importante advertir que la semántica declarativa no hace referencia al orden explícito de los objetivos dentro del cuerpo de una cláusula, ni al orden de las cláusulas dentro de lo que será el programa lógico. Este orden será, sin embargo, fundamental para la semántica operacional de PROLOG. Ello es la causa fundamental de las divergencias entre ambas semánticas en las implementaciones prácticas de dicho lenguaje y fuente de numerosos errores de programación, que además no siempre son fáciles de detectar.

Las sustituciones son utilizadas en programación lógica para, mediante su aplicación a las variables contenidas en una cláusula, obtener la expresión de la veracidad de una relación lógica particular a partir de la veracidad de una relación lógica más general incluida en el programa. Más formalmente, dicho concepto se conoce con el nombre de *unificación*, que pasamos a definir inmediatamente.

Definición 42 *Un* unificador *de dos términos lógicos T_1 y T_2 es una sustitución Θ, tal que $T_1\Theta = T_2\Theta$. Cuando al menos existe un unificador para dos términos lógicos T_1 y T_2, existe un unificador particular Θ llamado el* unificador más general (umg) de T_1 y T_2*, tal que para cualquier otro unificador Θ', existe una sustitución σ tal que $\Theta' = \Theta\sigma$.*

Intuitivamente, el $umg(T_1, T_2)$ representa el número mínimo de restricciones a considerar sobre dos términos para hacerlos iguales.

Ejemplo 41 *Dados los términos lógicos:*

$$T_1 = f(X, g(X, h(Y)))$$
$$T_2 = f(Z, g(Z, Z))$$

un conjunto de posibles unificadores es el siguiente:

$$\Theta_1 \equiv \{X \leftarrow h(1), Z \leftarrow h(1), Y \leftarrow 1\}$$
$$\Theta_2 \equiv \{X \leftarrow Z, Z \leftarrow h(Y)\}$$
$$\Theta_3 \equiv \{X \leftarrow Z, Z \leftarrow h(1), Y \leftarrow 1\}$$

donde el $umg(T_1, T_2)$ es Θ_2.

Desde un punto de vista práctico, el umg nos permitirá efectuar nuestro razonamiento lógico conservando la mayor generalidad posible en las conclusiones. Es por ello que el umg es el unificador utilizado en el proceso de demostración que constituye la interpretación lógica. Para su obtención, la mayoría de los dialectos PROLOG actuales consideran el algoritmo de Robinson [**?**], que pasamos a describir inmediatamente.

Algoritmo 41 *Sean T_1 e T_2 dos términos lógicos, el siguiente seudocódigo describe el* método de unificación de Robinson*, calculando el $umg(T_1, T_2)$.*

Entrada: *Dos términos lógicos T_1 y T_2.*

Salida: $\Theta = umg(T_1, T_2)$, *si existe; en otro caso* `fail`.

inicio
$\Theta := \emptyset$;
`meter (`$T_1 \equiv T_2$`, Pila)` ;
mientras `Pila` $\neq \emptyset$ **hacer**
  $(X \equiv Y)$ `:= sacar(Pila)` ;
  caso
    X `e` Y `constantes,` $X = Y$: **nada**
    $X \notin Y$, X `variable` : `sustituir (`X, Y`);`
      : `añadir (`$\Theta, X \leftarrow Y$`)`
    $Y \notin X$, Y `variable` : `sustituir (`Y, X`);`
      : `añadir (`$\Theta, Y \leftarrow X$`)`
    $(X \leftarrow f(X_1, \ldots, X_n))$ **y**
    $(Y \leftarrow f(Y_1, \ldots, Y_n))$: **para** i `:=` n **hasta** *1* **paso** *-1* **hacer**
      `meter(`$X_i \leftarrow Y_i$`, Pila)`
      fin para
    sino : **devolver** `fail`
  fin caso
fin mientras ;
devolver Θ
fin

donde $X \notin Y$ *expresa que el valor de* X *no forma parte de la estructura correspondiente a* Y*. Es lo que se conoce como* test de ciclicidad en unificación[7]*. La función* `meter(`$T_1 \equiv T_2$`, Pila)` *introduce la ecuación lógica* $T_1 \equiv T_2$ *y la función* `sacar(Pila)` *extrae la ecuación lógica* $X \equiv Y$ *de la estrucura* LIFO[8] `Pila`*. La función* `sustituir`(X, Y) *sustituye la variable* X *por* Y *en la pila y en la sustitución* Θ*. Finalmente, la función* `añadir`(Θ, σ) *añade al unificador* Θ *la sustitución* σ*.*

Para ilustrar el algoritmo anterior, consideramos dos ejemplos. Uno de ellos, el último, incluye un ciclo de unificación y, por tanto la unificación debiera abortarse.

Ejemplo 42 *Calcularemos el umg para los términos* T_1 *y* T_2 *del anterior ejemplo 41, y al que denominaremos* Θ*. Lo haremos de forma incremental, estudiando cada una de las ramas de los términos implicados, dos a dos, y actualizando continuamente el contexto de trabajo con los nuevos valores asignados a las variables durante el proceso. En la misma línea, usaremos una pila como estructura de priorización del tratamiento de unos u otros términos en cada momento. Así, podemos sintetizar el cálculo del* $umg(T_1, T_2)$ *como sigue:*

[7] *occur-check* en terminología anglosajona.
[8] **L**ast **I**nput **F**irst **O**utput.

$$\underset{\boxed{T_1 = T_2}}{\Theta \equiv \{\}} \vdash \underset{\boxed{g(Z,Z) = g(X,h(Y))}}{\overset{\Theta \equiv \{\}}{\boxed{X = Z}}} \vdash \underset{\boxed{g(Z,Z) = g(Z,h(Y))}}{\Theta \equiv \{X \leftarrow Z\}} \vdash \underset{\boxed{Z = h(Y)}}{\overset{\Theta \equiv \{X \leftarrow Z\}}{\boxed{Z = Z}}} \vdash$$

$$\vdash \underset{\boxed{Z = h(Y)}}{\Theta \equiv \{X \leftarrow Z\}} \vdash \underset{}{\Theta \equiv \{X \leftarrow Z,\ Z \leftarrow h(Y)\}}$$

Donde $\vdash$ *simplemente es una conectiva que enlaza las diferentes fases del proceso. Esto es, el resultado final es el dado por la sustitución*

$$\Theta \equiv \{X \leftarrow Z, Z \leftarrow h(Y)\}$$

Ejemplo 43 *Para ilustrar el efecto de las estructuras circulares en el proceso de unificación, consideraremos ahora los dos términos lógicos siguientes:* `igual(X,X)` *e* `igual(Y,f(Y))`*. Veremos que nuestro algoritmo de unificación, si no se aplica el test de ciclicidad, entra en un ciclo sin fin. En efecto, la secuencia de configuraciones en la pila que sirve de sustento al algoritmo, es la que sigue:*

$$\underset{\boxed{igual(X,X) = igual(Y,f(Y))}}{\Theta \equiv \{\}} \vdash \underset{\boxed{X = f(Y)}}{\overset{\Theta \equiv \{\}}{\boxed{X = Y}}} \vdash \underset{\boxed{Y = f(Y)}}{\Theta \equiv \{X \leftarrow Y\}} \vdash \text{fail}$$

Observemos que en la última de las configuraciones Y aparece en f(Y)*. Si continuamos con el proceso ocurrirá que sustituiremos toda ocurrencia de* Y *por* f(Y)*, y como consecuencia obtendremos:*

$$X \leftarrow Y \leftarrow f(Y) \leftarrow f(f(Y)) \leftarrow f(f(f(Y))) \leftarrow f(f(f(f(Y)))) \leftarrow \ldots$$

Esto es, la unificación ha entrado en un ciclo.

En este punto podemos ya introducir la noción de resolución que proporciona capacidad deductiva a la programación lógica. Lo haremos sobre la base del algoritmo más popular en las implementaciones PROLOG, la resolución SLD[9] [**?**]

Algoritmo 42 *El siguiente seudocódigo describe el* método de resolución SLD.

Entrada: *Un programa lógico* P *y una pregunta* Q*.*

Salida: $Q\Theta$*, si es una instancia de* Q *deducible a partir de* P*; en otro caso* `fail`*.*

[9] por **S**electing a literal, using a **L**inear strategy and searching the space of possible deductions **D**epth-first.

inicio
`Resolvente :=` $\{Q\}$;
mientras `Resolvente` $\neq \emptyset$ **hacer**
  $A \in$ `Resolvente` ;
  si $\exists\, P : -Q_1, \ldots, Q_n$ *tal que* $\exists\, \Theta = mgu(A, P)$ **entonces**
    `borrar (Resolvente,` A`)` ;
    `añadir (Resolvente,` $Q_1\Theta, \ldots, Q_n\Theta$`)` ;
    `aplicar (`Θ`, Resolvente)`
  sino devolver `fail`
  fin si
fin mientras ;
devolver Θ
fin

donde la función `borrar(Resolvente, A)` *borra el objetivo* `A` *de* `Resolvente`*, mientras que la función* `añadir(Resolvente,` $Q_1\Theta, \ldots, Q_2\Theta$`)` *añade los objetivos indicados a* `Resolvente`*. La función* `aplicar(`Θ`, Resolvente)` *aplica sobre el conjunto de objetivos de* `Resolvente` *la restricción representada por la sustitución* Θ*. En general consideraremos que una* resolvente *contiene en cada instante el conjunto de objetivos a resolver.*

El proceso de resolución SLD puede representarse en forma arboresecente, de forma que la existencia de una rama cuya última hoja es la resolvente vacía se traduce en la existencia de una instancia que es consecuencia lógica del programa, esto es, de una respuesta.

Definición 43 *Sean P un programa lógico y Q una pregunta, entonces el* árbol de resolución *correspondiente se construye en la forma siguiente:*

1. *El nodo raíz del árbol es una resolvente con Q.*
2. *Seleccionado un objetivo A en la resolvente, la construcción del árbol en cada nodo continúa como sigue:*
   (a) Si A se puede unificar con la cabeza de cada una de las cláusulas

$$\begin{array}{l} P_1 : -Q_1^1, \ldots, Q_1^{m_1}. \\ P_2 : -Q_2^1, \ldots, Q_2^{m_2}. \\ \vdots \\ P_n : -Q_n^1, \ldots, Q_n^{m_n}. \end{array}$$

   mediante las sustituciones $\{\Theta_i,\ i=1,2,\ldots,n\}$, entonces construimos n ramas para ese nodo. En cada una escribimos la nueva resolvente derivada de la cláusula C_i y de la sustitución Θ_i, renombrando[10] *automáticamente las variables de los nuevos objetivos incorporados a la resolvente. Los átomos de la cola de la cláusula unificada se añaden a la resolvente, mientras que el objetivo a resolver es eliminado de la misma.*

[10] dicho renombramiento puede efectuarse simplemente mediante la incorporación de un subíndice a los nombres de las variables. Es importante indicar que esta técnica es necesaria para indicar qué variables de igual nombre en cláusulas distintas son diferentes, evitando de este modo confusiones en las futuras sustituciones.

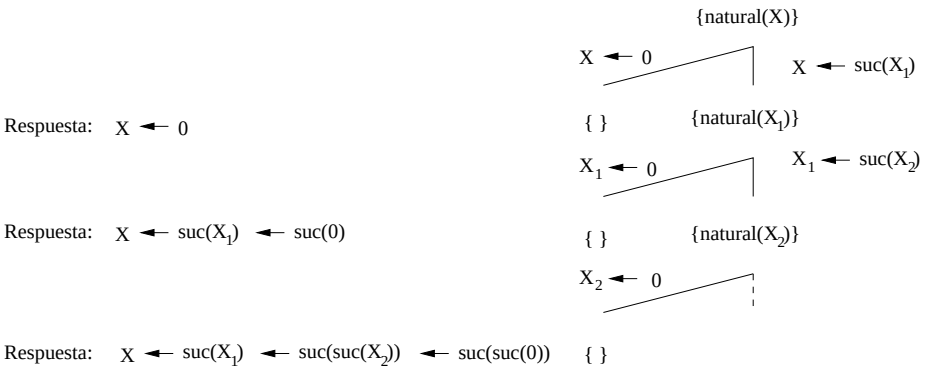


Fig. 1. Resolución para la pregunta `:- natural(X)`.

(b) Si el átomo seleccionado no se puede unificar con la cabeza de ninguna de las cláusulas de P, se paraliza la construcción de la rama del árbol y se devuelve `fail` *como respuesta.*

(c) Si la resolvente resultante está vacía es porque no queda ningún objetivo por resolver. En este caso, también se paraliza la construcción de dicha rama y se devuelve `true`*, además del resultado de la composición de las sustituciones $\tilde{\Theta}_1\tilde{\Theta}_2\ldots\tilde{\Theta}_l$ que han sido consideradas desde la raíz hasta el nodo en cuestión. Ello constituye la respuesta a la pregunta Q.*

Es importante observar aquí que, ni la resolución SLD ni el árbol asociado determinan orden alguno en el tratamiento de las cláusulas del programa lógico, ni tampoco en relación a los objetivos dentro de las propias cláusulas.

Ejemplo 44 *Para ilustrar el concepto de árbol de resolución, retomamos el programa del ejemplo 32,y suponemos que nuestra pregunta es la dada por:*

$$:-\texttt{natural(X)}.$$

que declarativamente se interpretaría como:

"¿ Existen valores para `X`, tal que `X` es natural ?"

un posible árbol de resolución asociado es el mostrado en la figura 1, pero también lo es el mostrado en la figura 2. Observar que las respuestas encontradas son las mismas, pero que los árboles son diferentes. Las líneas discontinuas indican que el árbol es, en cada caso, infinito.

4.2 Semántica procedural

Aunque declarativamente el proceso de resolución pueda definirse en términos propios de la lógica de primer orden, expresando la elección de cláusulas

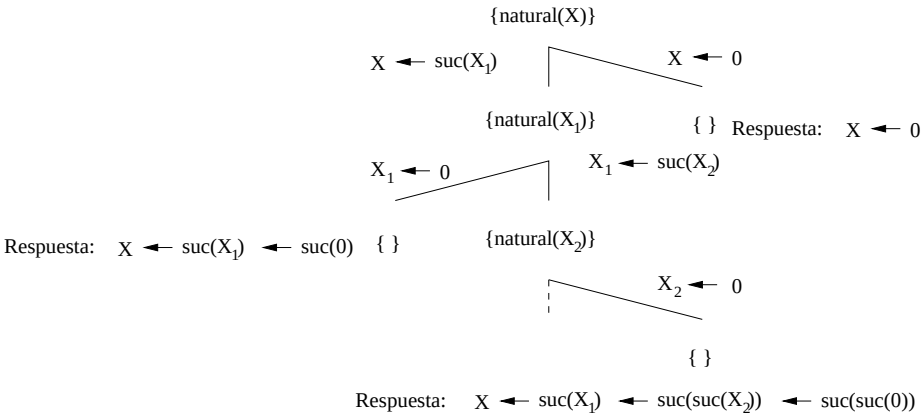


Fig. 2. Resolución alternativa para la pregunta `:- natural(X)`.

y objetivos en términos de cuantificadores existenciales y universales, las implementaciones prácticas no tienen otro remedio que establecer órdenes de exploración del espacio de cálculo.

Definición 44 *Sean P un programa lógico y Q una pregunta, entonces el* árbol de resolución PROLOG *para el programa P, dada la pregunta Q, se construye de la misma forma que en la anterior definición 43, salvo que:*

1. *El objetivo a resolver es siempre el primer átomo A de la resolvente Q, si consideramos esta como una lista ordenada de izquierda a derecha.*
2. *En el proceso de unificación del objetivo a resolver, con las cabezas de las cláusulas del programa, se impondrá un orden de exploración de éstas que será de arriba hacia abajo. Este ordenamiento se trasladará en una exploración asociada de las ramas del árbol de izquierda a derecha, en profundidad.*
3. *Al añadir a la resolvente los átomos de la cola de la cláusula unificada con el objetivo a resolver, estos sustituyen la posición que tenía en la resolvente el objetivo resuelto, a la vez que conservan el orden local que mantenían en la cola de la cláusula.*

Ejemplo 45 *Retomando los términos del ejemplo 32, sólo el árbol de resolución para la pregunta:*

```
:- natural(X).
```

que se muestra en la figura 1 refleja la semántica operativa del intérprete lógico SLD*, no así la que se reflejaba en la figura 2.*

Al construirse el árbol de resolución en profundidad es necesario articular un protocolo para que, una vez agotada o truncada la exploración de una

rama, podamos localizar la siguiente rama a explorar. Para ello remontaremos la estructura arborescente ya construida, revisitando cada nodo y aplicando el proceso siguiente:

- Si existe una rama no explorada por la derecha, esto es, queda alguna cláusula aplicable como alternativa a la anteriormente escogida para resolver el primer objetivo de la resolvente de ese nodo, entonces esa nueva posibilidad es estudiada.
- En otro caso, remontamos un nodo más en nuestra rama y recomenzamos el proceso para el mismo, hasta haber agotado todas las posibilidades. Si esto último ocurre devolvemos `fail` como respuesta.

en un proceso que denominamos *retroceso*[11].

Ejemplo 46 *Para ilustrar el concepto de retroceso, consideremos el siguiente programa, cuyo objeto es implementar el concepto de suma de números naturales:*

```
suma(0,N,N).
suma(suc(N_1),N_2,suc(N_3)) :- suma(N_1,N_2,N_3).
```

donde `suc(X)` *es la función ya considerada en el ejemplo 32, y* `suma(S_1,S_2,R)` *es la notación que determina la semántica declarativa del predicado* `suma/3`*, cuyo significado será:*

> "El resultado de la adición de los sumandos `S_1` y `S_2` es `R`."

a partir de la cual hemos construido las dos cláusulas del predicado `suma/3` *en la forma:*

> "La suma del cero y de un número cualquiera, es dicho número si éste es natural."
> "La suma del sucesor del número `N_1` y un número `N_2`, es el sucesor del número `N_3`, si `N_3` es el resultado de sumar `N_1` y `N_2`."

Entonces la pregunta `:- suma(X,Y,Z).` *se interpreta como:*

> "¿ Existen valores para las variables `X`, `Y` y `Z`; tales que `X + Y = Z`."

y el proceso de resolución debiera, por tanto, proveer la capacidad de cálculo para una infinitud de posibles soluciones. Ello implicará, en este caso, la aplicación de retrocesos sobre cada nodo del árbol de resolución, tal y como se ilustra en la figura 3, donde las respuestas se obtienen por encadenamiento de las sustituciones aplicadas y estas sustituciones aparecen asociadas a la rama correspondiente.

Dado que todas las ramas se construyen con éxito, y por tanto ninguna falla, el retroceso se fuerza por el usuario, lo que habitualmente se realiza mediante la introducción de un carácter "`;`*" desde el teclado.*

[11] *backtracking* en terminología anglosajona.

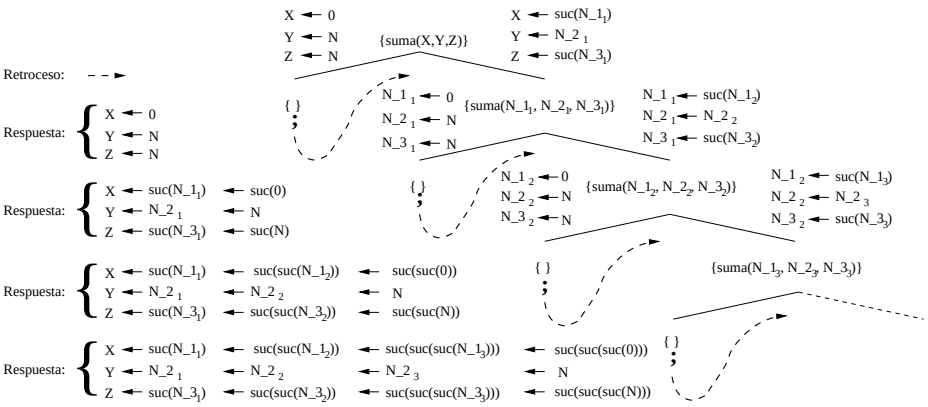


Fig. 3. Resolución para la pregunta `:- suma(X,Y,Z)`.

4.3 Incongruencias declarativo/procedurales

La sutil diferencia entre la semántica declarativa y la procedural que hemos introducido para la implementación de la resolución SLD, introduce incongruencias que el programador deberá tener en cuenta para asegurar la corrección de su código. Más exactamente, el alejamiento de la semántica declarativa puede llevar a situaciones en las que un programa sea declarativamente correcto, pero no así desde el punto de vista procedural.

Al origen del problema situaremos las alteraciones que en el tratamiento de los objetivos de la resolvente introducen las implementaciones prácticas del algoritmo SLD. Más concretamente, nos referimos a la fijación de un orden de exploración en el árbol de resolución que rompe la dinámica de tratamiento de conjunto de las resolventes para trasladarlas a un protocolo LIFO[12], extremadamente eficaz desde el punto de vista computacional, pero que compromete la completud de la resolución.

Ejemplo 47 *Supongamos un conjunto de cláusulas que define la siguiente semántica declarativa:*

"Un individuo es humano si tiene una madre que es humana"

y que podríamos definir de la forma siguiente:

```
humano(X) :- humano(Y), madre(X,Y).
humano("Elena").
madre("Juan","Elena").
```

donde `madre(X,Y)` *denota que la madre de* `X` *es* `Y`*, y* `humano(X)` *se interpreta como que* `X` *es humano. Consideremos en estas circunstancias la pregunta:*

[12] por *Last Input First Output.*

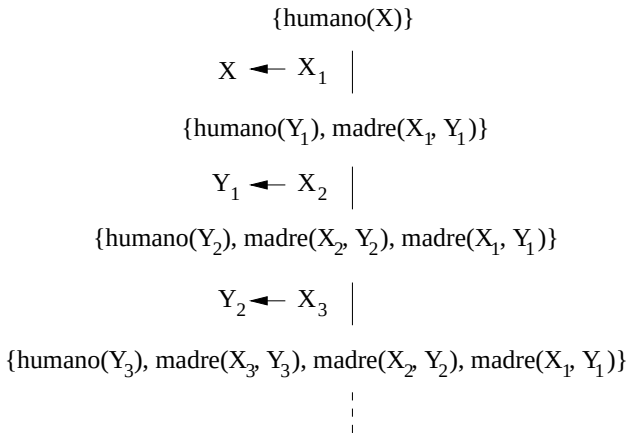


Fig. 4. Resolución para la pregunta `:- humano(X)`.

```
:- humano(X).
```

cuya semántica declarativa sería:

" ¿Existen valores para la variable `X`, tal que `X` sea humano ?"

Resulta evidente que el programa refleja perfectamente la semántica declarativa considerada y, en consecuencia, el programa debiera ser correcto. En consecuencia, ante la pregunta:

```
:- humano(X).
```

y en función de la información disponible, debiéramos deducir que `"Juan"` *y* `"Elena"` *son humanos.*

Sin embargo, la semántica procedural considerada, nos lleva a construir el árbol de resolución de la figura 4, donde el proceso no lleva a respuesta alguna. El origen de tal comportamiento no es otro que el de la consideración del axioma `humano("Elena")` *en una posición no prioritaria y la introducción de una recursividad izquierda en la primera cláusula, incompatible con la construcción descendente del árbol. Consideremos, para comenzar, la siguiente variante del programa inicial, que mantiene intacta la semántica declarativa del mismo:*

```
humano("Elena").
humano(X) :- humano(Y), madre(X,Y).
madre("Juan","Elena").
```

lo que no varía la semántica declarativa, pero permite obtener una respuesta, asociada a la primera cláusula del predicado `humano/1`*, aún a pesar de la recursividad izquierda de la segunda, tal y como puede verse en la figura 5.*

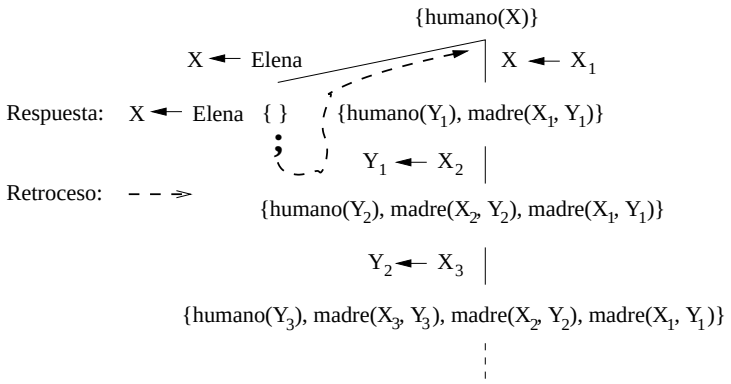


Fig. 5. Resolución alternativa para la pregunta `:- humano(X)`.

Finalmente, consideremos ahora la siguiente variante del programa, en el que además hemos cambiado el orden, declarativamente irrelevante y que permite eliminar la recursividad izquierda antes comentada, de los objetivos de la cola en la cláusula recursiva del predicado `humano/1`*:*

```
humano("Elena").
humano(X) :- madre(X,Y), humano(Y).
madre("Juan","Elena").
```

tal y como podemos ver en la figura 6, en este caso si hemos obtenido todas las respuestas esperadas. Ello demuestra el impacto en la resolución tanto del orden de los objetivos en las resolventes como de las cláusulas del programa lógico.

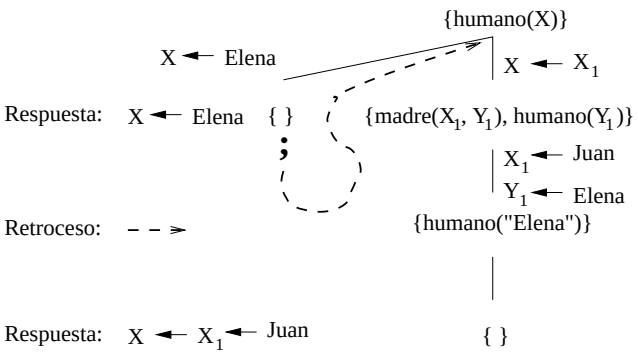


Fig. 6. Resolución alternativa para la pregunta `:- humano(X)`.

5 Control

Si en el paradigma imperativo la única referencia para la programación es la propia semántica procedural, en el caso lógico esto no ocurre. En particular, cualquier usuario que pretenda trasladar un programa imperativo en uno lógico, constatará la total ausencia de estructuras de control, omnipresentes en el primer caso.

5.1 El corte

El *corte* es el método más utilizado para establecer un control de la resolución en un programa lógico. Sin embargo, su utilización romperá la semántica declarativa del lenguaje, cercenando la potencia de cálculo derivadas de la unificación y resolución.

Formalmente, el *corte* es un predicado sin argumentos, que se verifica siempre y que se representa mediante la notación "`!`". Como efecto colateral, suprime en el árbol de resolución todas las alternativas que puedan quedar por explorar para los predicados que se encuentren entre la cabeza de la cláusula en la que aparece, y la posición en la que el corte se ha ejecutado.

Ejemplo 51 *Se trata de implementar la estructura de control por autonomasia, el* `if_then_else`*. Su semántica será la habitual. Así, interpretaremos la notación* `if(P, Q, R)` *en la forma:*

"Si `P` es cierto, entonces probar `Q`, sino probar `R`."

cuya implementación es la siguiente:

```
if(P,Q,R) :- P, !, Q.
if(P,Q,R) :- R.
```

El corte indica que una vez probado el objetivo `P`*, el único camino a seguir es el indicado por la primera cláusula, esto es, probar* `Q`*.*

5.2 El fallo

Este predicado se denota por `fail`, y como resultado de su ejecución se produce un fallo. Proceduralmente ello implica que la resolución se paraliza en la rama actual del árbol, forzándose un retroceso en busca de la siguiente rama a explorar.

Ejemplo 52 *Para ilustrar el concepto de fallo, consideramos la implementación de la relación* inferior que *en el conjunto de los números naturales:*

```
inferior(0,0) :- fail.
inferior(0, suc(Y)).
inferior(suc(X),suc(Y)) :- inferior(X,Y).
```

cuya semántica declarativa viene dada por:

"El cero no es inferior al cero"
"El cero es inferior a cualquier número que sea sucesor de otro"
"El sucesor de un natural `X` es inferior al sucesor de otronatural `Y`, si `X` es inferior a `Y`".

y para el que el cálculo de respuesta negativa para la pregunta:

```
:- inferior(suc(0), suc(0)).
```

se muestra en la figura 7.

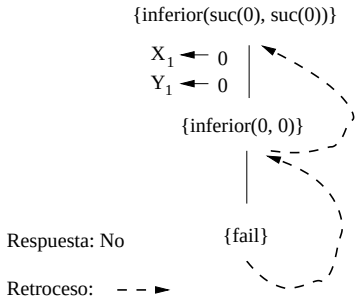


Fig. 7. Resolución para la pregunta `:- inferior(suc(0),suc(0))`.

Observar que en este caso el retroceso no es provocado por el usuario mediante la introducción de un ";" desde el teclado, sino forzado automáticamente al evaluarse el `fail`*. El hecho de que dicho retroceso sea infructuoso al no quedar ramas por explorar es lo que conlleva la respuesta negativa a la pregunta.*

5.3 La negación

La conjunción de los predicados corte y `fail` permiten la consideración de una forma de negación habitual en programación lógica. Es lo que se denomina la *negación por fallo*. Intuitivamente el principio de funcionamiento que la define es muy simple y puede resumirse en la siguiente semántica declarativa:

"La negación por fallo, `not(X)`*, de un predicado* `X` *es* `fail` *si podemos demostrar que* `X` *es cierto"*
" En otro caso, `not(X)` *es cierta."*

y aunque aparentemente tal definición se corresponde con la noción de negación lógica, esto no es exactamente así. Basta pensar que no poder demostrar la veracidad de algo, no quiere decir que sea falso, simplemente que hemos fallado en tal intento. De ahí la denominación de negación por fallo, lo que tendrá profundas implicaciones en la construcción de programas que la incluyan. Su implementación se resume en dos cláusulas:

```
not(X) :- X, !, fail.
not(X).
```

lo que se corresponde exactamente con la semántica declarativa antes comentada. La combinación "`!, fail`" permite provocar el fallo, a la vez que el corte evita el retroceso sobre la segunda cláusula de la negación por fallo.

Ejemplo 53 *Para ilustrar tanto el funcionamiento de la negación, como de la combinación "*`!, fail`*", consideraremos un ejemplo entorno a la idempotencia de la negación, de hecho una de sus propiedades fundamentales. En concreto consideraremos las dos preguntas siguientes:*

1. *La primera será* `:- not(not(fail)).`*, cuya respuesta debiera ser* `fail`*.*
2. *La segunda será* `:- not(not(true)).`*, cuya respuesta debiera ser* `true`*.*

donde `true` *es un predicado habitualmente predefinido, que siempre es cierto. Para ello nos remitimos a la figura 8, donde con el fin de diferenciar los diferentes cortes implicados, nos referiremos a ellos mediante subíndices que nos permitan distinguirlos. De este modo, en los dos árboles de resolución mostrados podemos ver cortes "*$!_1$*" y "*$!_2$*". Tal como se puede comprobar, en ambos casos se alcanzan las respuestas esperadas.*

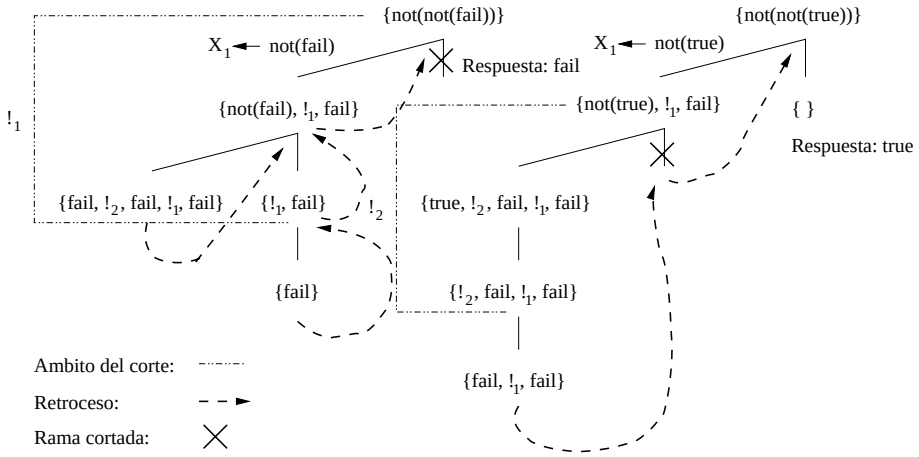


Fig. 8. Resolución para las preguntas `:- not(not(fail))` y `:- not(not(true))`.

6 Listas

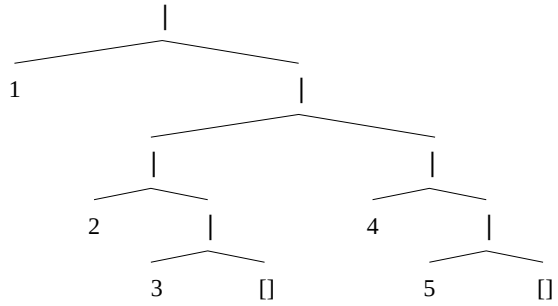
Las listas consituyen una estructura de programación básica en el paradigma lógico, y ello por su adaptabilidad a dos características fundamentales del mismo.

La primera al estilo de programación aquí requerido, netamente recursivo. La segunda a la unificación como mecanismo de gestión dinámico.

Una lista se representa por la sucesión de sus elementos separados por comas, y encerrada entre corchetes. Así, la lista cuyo primer elemento es `1`, cuyo segundo elemento es `2` y cuyo tercer elemento es `3`, se representará en la forma `[1,2,3]`.

La implementación física de la estructura de datos es un *doblete* cuyo primer elemento denominamos `car`, siendo el segundo el `cdr`. El `car` es el primer elemento de la lista y el `cdr` es lo que queda de la lista original una vez eliminado el primer elemento. Los dobletes conectan `car` y `cdr` mediante el conectivo `cons`, que habitualmente se representa mediante el símbolo "`|`".

Ejemplo 61 *Consideremos la lista de números naturales* `[1, [2, 3], 4, 5]`*, entonces su representación física se corresponde con el siguiente árbol binario:*



que podemos referir igualmente, entre otras, mediante las notaciones

$$[1 \mid [[2,3],4,5]] \qquad [1,[2,3] \mid [4,5]] \qquad [1,[2,3],4 \mid [5]]$$
$$[1,[2,3],4,5 \mid []] \qquad [1,[2 \mid [3]],4,5] \qquad [1,[2,3 \mid []],4,5]$$

donde "`[]`*" es el símbolo de fin de lista, comúnmente denominado* lista vacía*, pero que curiosamente no es una lista al carecer de* `car` *y* `cdr`*.*

Nuestra intención ahora será la de utilizar todo el potencial de la unificación como mecanismo para manejar la recursividad en programas cuya estructura básica de datos sean las listas.

Ejemplo 62 *Consideraremos la implementación de un predicado* `concat(L_1, L_2, R)` *describiendo la concatenación de las listas* `L_1` *y* `L_2` *para obtener la lista* `R`*. La implementación podría ser la siguiente:*

```
concat([],L,L).
concat([Car|Cdr],L,[Car|R]) :- concat(Cdr,L,R).
```

cuya semántica declarativa viene dada por:

> "La concatenación de la lista vacía con otra cualquiera, es esta última."
> "La concatenación de una lista `[Car | Cdr]` con otra lista `L` es el resultado de concatenar el `Car` al resultado `R` previamente obtenido de la concatenación de `Cdr` con `L`."

Para ilustrar la potencia de cálculo en este caso, consideraremos la pregunta:

```
:- concat(X, Y, Z).
```

cuyas respuestas son obviamente infinitas. En este sentido, la figura 9 muestra el árbol de resolución correspondiente, así como algunas de las respuestas que airosamente provee en este caso la resolución SLD.

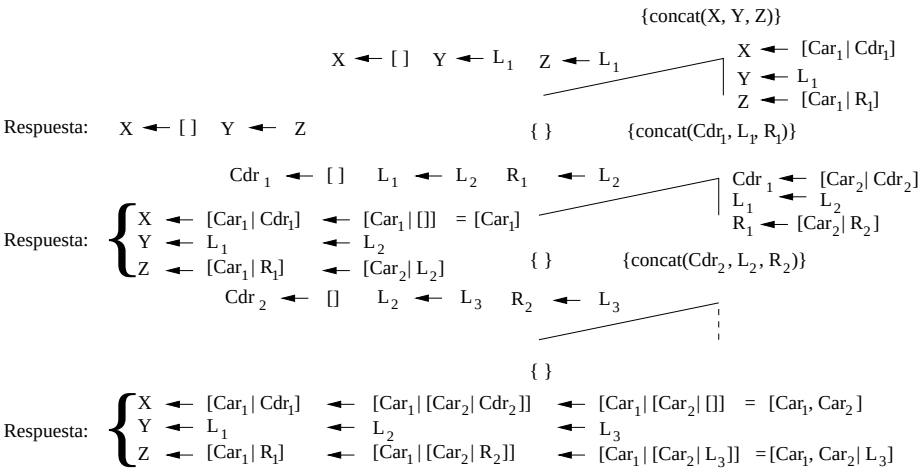


Fig. 9. Resolución para la pregunta `:- concat(X, Y, Z)`.

Observar en particular la técnica utilizada para introducir un elemento en cabeza de una lista en la segunda cláusula del predicado `concat/3`. *En efecto, se ha utilizado la unificación para insertar el* `Car` *al principio de la lista* `R`.

Ejemplo 63 *Nuestro objetivo ahora es la implementación del conocido algoritmo de ordenación de listas numéricas habitualmente conocido como* quicksort. *Técnicamente el método se sustancia en los siguientes pasos:*

1. *Elegir un elemento de la lista, al que denominaremos* pivote, *y que servirá como referencia comparativa para los demás. En nuestro caso, y para simplificar la exposición, el pivote será siempre el primer elemento de la lista a ordenar.*
2. *Elegido el pivote, dividimos la lista original en dos partes. Una, que situaremos a su izquierda, que contendrá a los elementos en la lista cuyo valor sea menor o igual al pivote. La otra, que situaremos a la derecha del pivote, contendrá a los elementos en la lista cuyo valor sea mayor al referido elemento.*
3. *El proceso se aplica ahora recursivamente sobre cada elemento, izquierdo y derecho, de la partición de la lista original hasta que las particiones se agoten. En ese momento la lista original estará totalmente ordenada.*

que implementaremos usando tres predicados diferentes:

- *El primero será nuestro predicado principal* `quicksort(L,R)`*, cierto cuando* `R` *sea el resultado de ordenar la lista numérica* `L` *mediante el algoritmo del* quicksort*.*
- *El segundo será el predicado* `partir(Pivot,L,MenIg,May)`*, cierto cuando* `MenIg` *y* `May` *sean respectivamente la lista de elementos menores o iguales y mayores que el número* `Pivot` *en la lista* `L`*.*
- *El tercero será el predicado* `concat/3`*, ya comentado en el ejemplo 62.*

a partir de los cuales construimos el siguiente conjunto de cláusulas que reflejan el algoritmo declarativo antes descrito:

```
quicksort([],[]).
quicksort([Car|Cdr],R) :- partir(Car,Cdr,Izq,Der),
                          quicksort(Izq,Izq_ordenada),
                          quicksort(Der,Der_ordenada),
                          concat(Izq_ordenada,[Car|Der_ordenada],R).

partir(Pivot,[],[],[]).
partir(Pivot,[Car|Cdr],[Car|Izq],Der) :- Car =< Pivot,
                                         !,
                                         partir(Pivot,Cdr,Izq,Der).
partir(Pivot,[Car|Cdr],Izq,[Car|Der]) :- partir(Pivot,Cdr,Izq,Der).

concat([],L,L).
concat([Car|Cdr],L,[Car|R]) :- concat(Cdr,L,R).
```

Observar el uso del corte en la segunda cláusula del predicado `partir/4`*. Ello asegura que la tercera cláusula del referido predicado sólo se ejecutará cuando el test* `Car > Pivot` *sea cierto. De prescindirse del uso del corte, sería necesario incluir explítamente dicho test como primer elemento de la cola esa tercera cláusula en* `partir/4`*.*

7 Evaluación perezosa

Tal y como ya hemos comentado, las incongruencias declarativo/procedurales habitualmente asociables a PROLOG, tienen su origen en la alteración de las directrices de evaluación de la resolución SLD original y, en particular, en el tratamiento de términos no instanciados. A este respecto, los intérpretes lógicos introducen el concepto de *evaluación perezosa* como mecanismo de control de la evaluación, sujeta a condiciones fijadas por el propio usuario.

7.1 El predicado `freeze/2`

La función del predicado `freeze(Variable, Objetivo)` es demorar la evaluación de `Objetivo` hasta que `Variable` esté instanciada. De esta manera,

podemos considerar dos casos en la evaluación de este predicado. Cuando `Variable` está instanciada, `Objetivo` se evaluará normalmente. En caso contrario el objetivo de la resolvente actual correspondiente a la instanciación del término `freeze(Variable, Objetivo)` se retira y su evaluación queda pendiente. De este modo, el intérprete continúa con la evaluación del resto de objetivos, hasta que `Variable` haya sido por fin instanciada. Es entonces cuando `Objetivo` vuelve a la cabeza de la resolvente para ser evaluado.

Esta forma de control en la ejecución de objetivos en función de la disponibilidad de sus instanciaciones permite al programador asegurar ésta última, evitando las incongruencias antes referidas.

Ejemplo 71 *Un primer ejemplo ilustrativo de la situación nos lo proporciona el predicado extralógico* `is`*, que activa la evaluación de los operadores aritméticos clásicos en* PROLOG*. En particular, su uso exige la instanciación previa de todas y cada una de las variables implicadas en la expresión aritmética. Por tanto, por ejemplo, cuando queremos calcular el doble de un número mediante la cláusula:*

```
doble(X,Y) :- Y is X*2.
```

el valor de `X` *debe ser conocido previamente, lo que podemos asegurar mediante* `freeze/2` *en la forma:*

```
evaluar(X,Y,Z):- ..., freeze(X, doble(X,Y)), ...
```

En este contexto, dos son los ejemplos paradigmáticos en el control de la evaluación, justamente referidos a la resolución de las incongruencias declarativo/procedurales típicas de PROLOG. Se trata del tema de la recursividad izquierda y de los problemas planteados por el uso de la negación por fallo. Ilustraremos ambas situaciones mediante ejemplos.

Ejemplo 72 *Vamos a retomar el ejemplo 47 en el que se describía un posible programa para definir a un individuo como humano y, en concreto la primera versión propuesta:*

```
humano(X) :- humano(Y), madre(X,Y).
humano("Elena").
madre("Juan","Elena").
```

donde la existencia de recursividad izquierda frustraba el cálculo de cualquier respuesta a la pregunta `:- humano(X).`*, llevando al programa a una derivación sin fin de resolventes cada vez mayores. En aquella ocasión la solución pasaba por la eliminación directa de dicha recursividad. Sin embargo, podemos considerar otra alternativa el uso de* `freeze/2` *para asegurar que los objetivos de tipo* `humano(X)` *se evalúen sobre variables ya instanciadas,en la forma:*

```
humano(X) :- freeze(Y, humano(Y)), madre(X,Y).
humano("Elena").
madre("Juan","Elena").
```

lo que evita la entrada en el lazo recursivo, al obligar a evaluar primero `madre(X,Y)`.

Ejemplo 73 *Supongamos que queremos diferenciar un conjunto de comidas respecto a la temperatura a la que se consumen, esto es, en calientes o frías. Para ello, construímos el siguiente programa:*

```
caliente("sopa").
caliente("asado").
fria(Comida):- not(caliente(Comida)).
```

que declarativamente pueden interpretarse como:

"La sopa es una comida caliente"
"El asado es una comida caliente"
"Una comida está fría, si no está caliente".

aunque sabemos que la última cláusula, operacionalmente, tiene una interpretación sensiblemente diferente. A saber:

"Una comida está fría, si no podemos probar que está caliente".

Para poner en evidencia las incongruencias declarativo/procedurales a las que puede dar lugar una mala gestión de la negación, introduciremos el siguiente predicado `igual/2`*, que especifica la igualdad de dos términos cuando éstos se pueden unificar:*

```
igual(X,X).
```

En este contexto, podemos preguntar si una ensalada es una comida fría, entre otras, de las siguientes dos maneras:

```
:- fria("ensalada").
:- fria(X), igual("ensalada",X).
```

Ambas preguntas tienen la misma semántica declarativa, y, a pesar de ello, las respuestas que obtendremos, serán distintas: `true` *y* `fail`*, respectivamente. La diferencia fundamental entre ambos casos es que, en el primero, la variable afectada por la negación,* $\texttt{Comida}_1$ *está instanciada al valor* `ensalada` *en el momento de la negación. Esto es, conocemos perfectamente lo que estamos negando, situación en la que la negación por fallo garantiza su equivalencia con la negación lógica. Ello permite deducir si, en efecto, la ensalada es una comida caliente según los hechos del programa, tal y como se muestra en la figura 10.*

Sin embargo, en el caso de la segunda pregunta, tal y como se puede ver en la figura 11, ocurre exactamente lo contrario. Esto es, la variable `Comida` *no está instanciada cuando evaluamos la negación. Como resultado, obtenemos una respuesta negativa, totalmente incongruente.*

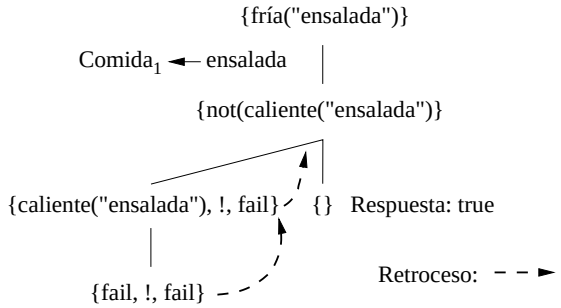


Fig. 10. Resolución para la pregunta `:- fria("ensalada")`.

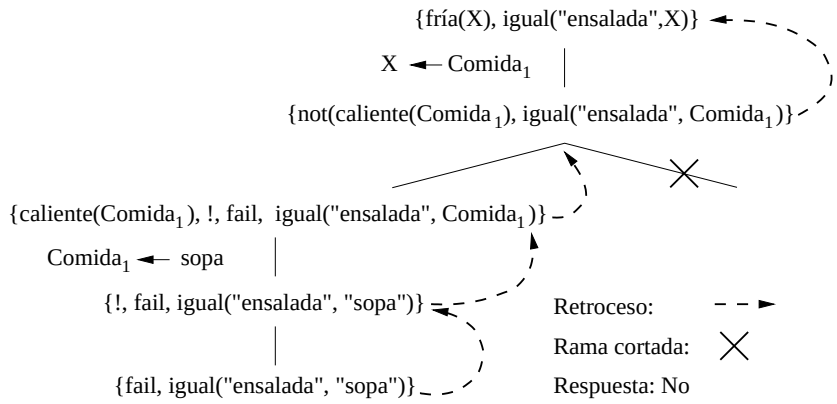


Fig. 11. Resolución para la pregunta `:- fria(X), igual("ensalada",X)`.

Para solventar el problema lo que haremos será retrasar la evaluación de la negación hasta el momento en el que todas las variables implicadas en ésta puedan ser conocidas. Esto es, hasta el momento en el que la negación por fallo garantice un comportamiento lógico congruente, tarea para la que echaremso mano del predicado `freeze/2` *en la forma:*

```
caliente("sopa").
caliente("asado").
fria(Comida):- freeze(Comida, not(caliente(Comida))).
```

obteniendo como resultado la respuesta correcta, tal y como se muestra en la figura 12.

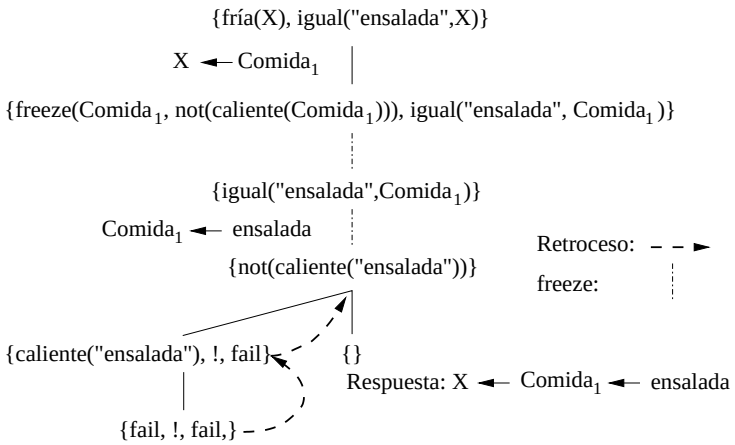


Fig. 12. Resolución para `:- fria(X), igual("ensalada",X)` con `freeze/2`.

7.2 El predicado `when/2`

Existen dos posibilidades al evaluar `when(Condición, Objetivo)`. Si `Condición` se verifica, se ejecuta `Objetivo`. En caso contrario, el átomo se retira de la resolvente y `Objetivo` sólo se ejecutará cuando al fin se verifique `Condición`. A este respecto, `Condición` suele construirse a partir de alguno de los siguientes predicados, algunos extralógicos, habituales en PROLOG:

- `?=(X,Y)`: se verifica si `X` e `Y` son idénticos o no pueden unificar.
- `nonvar(X)`: se cumple si `X` está instanciada.
- `ground(X)`: se verifica si `X` está instanciada a un término sin variables no instanciadas.
- `(Cond1, Cond2)`: conjunción (AND) de las condiciones `Cond1` y `Cond2`.
- `(Cond1; Cond2)`: disyunción (OR) de las condiciones `Cond1` y `Cond2`.

En esencia, al igual que `freeze/2`, el predicado `when/2` demora la ejecución de un objetivo, aunque en este caso en función de condiciones más complejas. De hecho, `freeze/2` puede definirse a partir de `when/2`, en la forma:

```
freeze(Variable,Objetivo) :- when(nonvar(Variable),Objetivo).
```

de modo que podemos utiliar `when/2` para solucionar el mismo tipo de anomalías de la negación que `freeze/2`.

Ejemplo 74 *El problema que nos ocupa ahora es decidir qué coche, de una determinada marca y modelo, queremos comprar. La única restricción que se nos impone es que la marca no sea europea. Consideremos, por ejemplo, el siguiente programa, que pretende resolver esta situación:*

```
coche("Dodge","Caliber").
coche("Opel","Corsa").
coche("Toyota","Prius").
europea("Opel").
comprar(Marca, Modelo) :- not(europea(Marca)), coche(Marca,Modelo).
```

cuya sem ántica declarativa viene dada por:

"El Dodge Caliber es un coche"
"El Opel Corsa es un coche"
"El Toyota Prius es un coche"
"Compra un modelo de coche si su marca no es europea".

Si ahora interrogamos a este programa con la pregunta `:- comprar(X.Y).`*, nos encontramos con que la respuesta es* `fail`*, a pesar de que modelos de marcas no europeas están listados como hechos.*

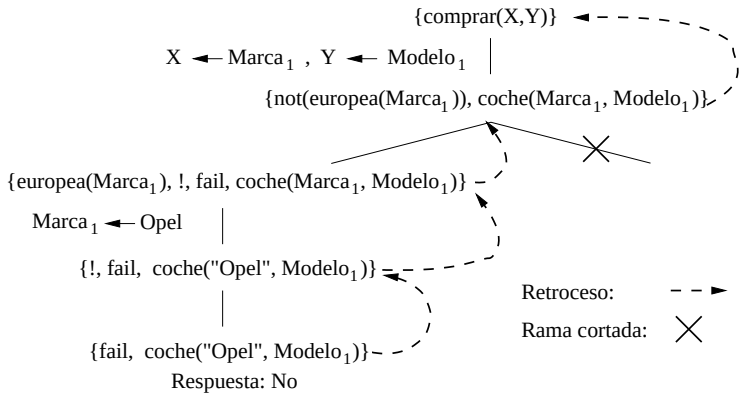


Fig. 13. Resolución para la pregunta `:- comprar(X,Y).`

La incongruencia tiene su origen en la no instanciación de la variable `Marca` *antes de evaluar la negación, como se puede ver en la figura 13. Para subsanar ese problema debemos asegurarnos que que* `Marca` *esté instanciada en el momento de la negación, lo que podemos conseguir a través de* `when/2`*, en la forma:*

```
comprar(Marca, Modelo) :- when(nonvar(Marca), not(europea(Marca))),
                          coche(Marca,Modelo).
```

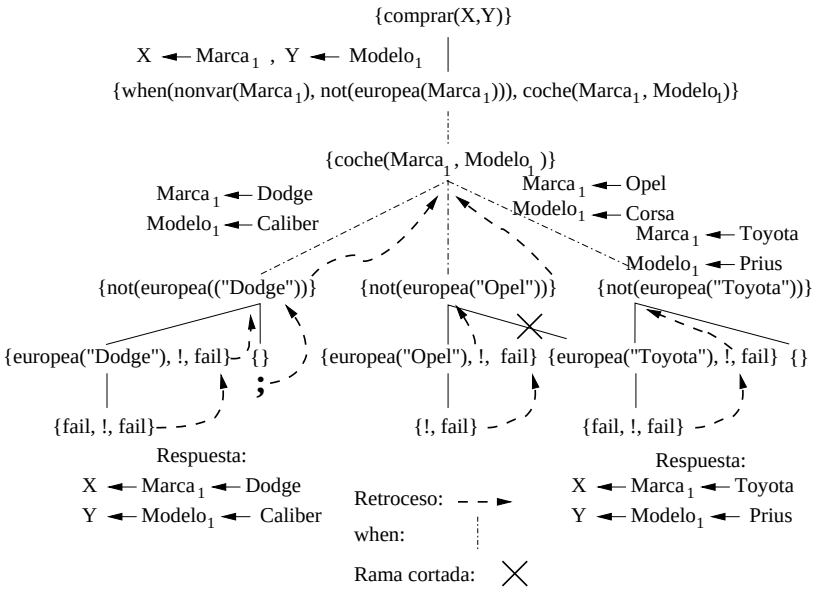


Fig. 14. Resolución para la pregunta `:- comprar(X,Y)` con `when`.

cuyo árbol de resolución para la pregunta referida se muestra en la figura 14. Podemos observar que, al forzar la evaluación de `coche/2` *antes de la negación, se prueba cada modelo de coche por separado. Como resultado, el* `Opel Corsa` *queda descartado por la negación, al constar en los hechos del programa que* `Opel` *es una marca europea.*

8 Operadores y capacidad expresiva

La posibilidad de definir dinámicamente nuevos operadores en programación lógica dota a este paradigma de una potencia expresiva que no encontraremos en otros lenguajes y que, de hecho, permitirá acercar la utilización y comprensión de programas a los usuarios no expertos de una forma efectiva, sobre la base de un acercamiento real al lenguaje natural que sirve de vínculo de comunicación entre humanos.

Formalmente, distinguimos tres tipos de operadores en relación a la posición que guardan respecto a sus argumentos: *infijos*, *prefijos* y *sufijos*; según aparezcan entre aquellos, delante de ellos o después. También podemos clasificarlos en relación al número de argumentos que manejan. Así podemos, por ejemplo, hablar de operadores *binarios* cuando poseen dos argumentos; y de *unarios* si sólo tienen uno.

Desde un punto de vista analítico, un operador necesita fijar tres parámetros para que su uso en programación no conlleve la introducción de ambigüedades en la interpretación del código:

1. La *notación* que lo representará.
2. La *prioridad* de evaluación en relación a otros operadores. Ello eliminará cualquier posibilidad de ambigüedad en la interpretación de expresiones que incluyan diferentes operadores.
3. La *asociatividad* en su evaluación. Ello eliminará cualquier posibilidad de ambigüedad en la interpretación de expresiones que incluyan al mismo operador repetido varias veces. Distinguiremos tres tipos de asociatividad: izquierda, derecha e inexistente.

En PROLOG, estos tres parámetros se definen por parte del usuario a través del predicado `op/3`, cuya sintaxis es la siguiente:

```
op(Prioridad, Asociatividad, Lista_de_notaciones)
```

y que ahora pasamos a describir detalladamente. En cuanto a la prioridad, ésta se designa mediante un número en el intervalo $[1, 1200]$, siendo más alta cuanto más pequeño es el número. Así, por ejemplo, podríamos asumir perfectamente que la suma tuviera una prioridad de 500, y la multiplicación una de 400.

En relación a la asociatividad, el lenguaje considera tres tipos distintos de operador binario: `xfx`, `xfy` y `yfx`. También permite la consideración de operadores unarios, cuya asociatividad se expresa de forma diferente según sean prefijos o sufijos. En el caso de los prefijos consideraremos asociatividades del tipo `fx` y `fy`, en el de los sufijos serán del tipo `xf` e `yf`. En cualquier caso, la interpretación de los valores `y`, `x` y `f` es común a todos ellos:

- `f` representa al operador binario, respecto al cual intentamos definir la asociatividad.
- `x` indica que dicha subexpresión debe tener una prioridad estrictamente menor[13] que la del funtor `f`.
- `y` indica que dicha subexpresión puede tener una prioridad menor o igual que la del funtor `f`.

considerando que la prioridad en la evaluación de una expresión viene dada por la prioridad de su funtor principal, esto es, del funtor que aparece en la raíz de su árbol de análisis sintáctico. En definitiva, lo que estamos diciendo es que

[13] es decir, un indicativo de prioridad mayor.

un operador del tipo `xfy` tendrá una asociatividad por la derecha, mientras que un operador de tipo `yfx` tendrá un asociatividad por la izquierda. El valor `xfx` indicará la ausencia de asociatividad. Así, por ejemplo, `op(500,yfx,[+])` declararía al operador "`+`" como asociativo por la izquierda y con una prioridad de 500. Además, dado que se trata de un simple predicado de interfaz con el sistema, no cabe esperar respuestas congruentes a preguntas del tipo:

```
:- op(P,yfx,[+]).    :- op(500,A,[+]).    :- op(500,yfx,N).
```

Ya en el caso unario, todo operador `op` declarado de tipo `fy` o `yf` tiene carácter asociativo, por lo que es válido escribir[14] `op op ... op operando` puesto que la expresión `op operando` tiene igual prioridad que `op` y en consecuencia puede ser utilizada como operando de este último operador. En cambio, un operador `op` definido como `fx` o `xf` no puede ser utilizado asociativamente, puesto que una expresión como `op op ... op operando` no será válida al no ser `op operando` de menor prioridad que `op`, lo cual implica que no puede ser utilizada como operando de este último operador.

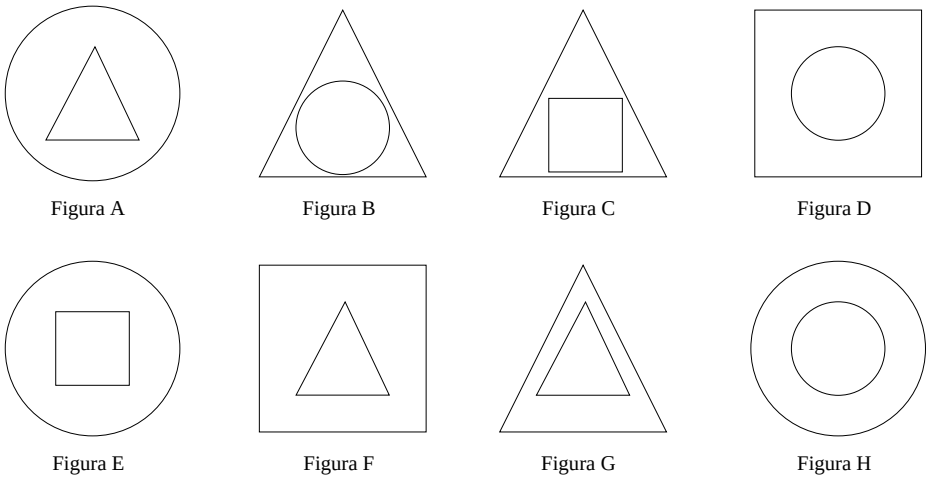


Fig. 15. Un conjunto de figuras para el problema de la analogía

Finalmente, es posible definir más de un operador con el mismo nombre, siempre que sean de diferente tipo. Es lo que se conoce como *sobrecarga* de un operador. El intérprete PROLOG identifica el operador concreto que se está utilizando mediante el examen de los operandos en análisis sintáctico. La utilización de operadores sobrecargados debe restringirse en lo posible.

En cualquier caso, la definición de nuevos operadores es una tarea delicada que debe tener muy en cuenta el comportamiento deseado de la nueva estructura

[14] en este caso suponemos que `op` es de tipo `fy`, es decir, un operador unario prefijo.

en relación al comportamiento de los operadores ya existentes. Exije, por tanto, un conocimiento profundo del problema cuya solución intentamos determinar.

Ejemplo 81 *Para ilustrar las ventajas e la definición y uso de operadores en programación lógica, consideraremos un problema clásico en inteligencia artificial, el* Problema de la Analogía.

Esencialmente se trata de establecer analogías entre formas geométricas. Como ejemplo concreto, a partir del conjunto de formas representadas en la figura 15, podemos considerar la existencia de algún tipo de relación entre ellas.

Figura	**Descripción**	**Figura**	**Descripción**
A	`triángulo dentro_de círculo`	B	`círculo dentro_de triángulo`
C	`cuadrado dentro_de triángulo`	D	`círculo dentro_de cuadrado`
E	`cuadrado dentro_de círculo`	F	`triángulo dentro_de cuadrado`
G	`triángulo dentro_de triángulo`	H	`círculo dentro_de círculo`

Table 2. Tabla de nombres para el problema de la analogía

Nuestro objetivo será, a partir de una relación entre dos objetos y un tercero, el de encontrar el análogo a este tercer objeto según la relación establecida entre los dos primeros. Declarativamente, la semántica del problema podría expresarse como sigue:

> "La figura `A` `es_a` la figura `G`, `como` la figura `D` `es_a` la figura `H`, `mediante` una relación de tipo `inversión`."

donde hemos subrayado los elementos que permiten ligar las relaciones y que, en nuestro código, estarán al origen de la definición de diversos operadores que introduciremos más tarde.

Para resolver el problema, la forma de proceder es sistemática. Primero escogeremos una notación adecuada para referirnos a las figuras, una notación que no queremos sea meramente nominativa sino también descriptiva. Ello nos permitirá extraer información relativa a las relaciones entre los componentes de las figuras mediante unificación para, luego, determinar las posibles relaciones con otras. De este modo, asociaremos a cada una de los elementos considerados en la figura 15, las denominaciones que mostramos en la tabla 2, lo que justifica la introducción del siguiente operador que,para nosotros, tendrá la mayor prioridad de los definidos en el código:

```
op(200,xfy,[dentro_de])
```

y consideraremos además los conectivos antes introducidos en la expresión declarativa del problema:

```
op(300,xfy,[es_a])   op(400,xfy,[como])   op(500,xfy,[mediante])
```

lo que nos permitirá ligar las relaciones de analogía existentes entre los elementos de la figura 15, y que denominaremos `igualdad`, `inversión`, `interior` *y* `contorno`. *De este modo podremos trasladar casi textualmente nuestro código al lenguaje natural utilizado por los humanos.*

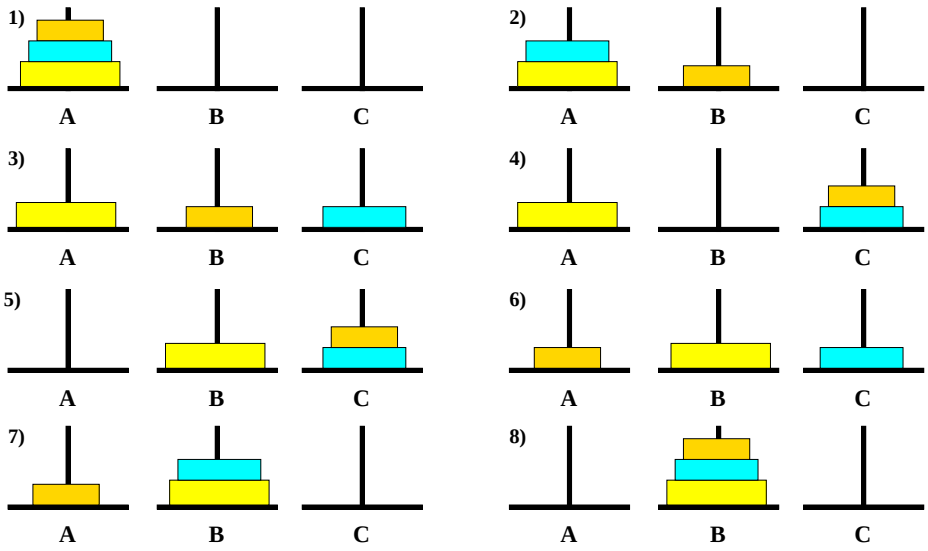


Fig. 16. Resolución de las Torres de Hanoi con tres discos.

Es importante observar que hemos definido `mediante` *como un operador con menor prioridad que* `como`, *éste a su vez menos prioritario que* `es_a`, *y éste a su vez como un operador con menor prioridad que* `dentro_de`. *Ello es imprescindible para la buena marcha del programa, puesto que queremos que* `mediante` *se evalúe más tarde que* `como`, *éste más tarde que* `es_a`, *y éste a su vez más tarde que* `dentro_de` *en las expresiones en las que los operadores aparezcan juntos. En este caso las asociatividades son irrelevantes ya que estos operadores nunca aparecen repetidos en secuencia*[15]. *Ahora ya podemos escribir nuestro programa:*

```
:- op(200,xfy,[dentro_de]).
:- op(300,xfy,[es_a]).
:- op(400,xfy,[como]).
:- op(500,xfy,[mediante]).

X es_a Y como Z es_a W mediante Relación :-
          figura(X), figura(Y),
```

[15] observar que, por ejemplo, si las figuras consideradas tuvieran al menos tres formas imbricadas, al menos el operador `dentro_de` si estaría en situación de definir una asociatividad real.

```
          verifican(X, Y, Relación),
          figura(Z), figura(W),
          verifican(Z, W, Relación).

verifican(Figura_1 dentro_de Figura_2,
          Figura_1 dentro_de Figura_2,igualdad).
verifican(Figura_1 dentro_de Figura_2,
          Figura_2 dentro_de Figura_1,inversión).
verifican(Figura_1 dentro_de Figura_2,
          Figura_1 dentro_de Figura_3,interior).
verifican(Figura_1 dentro_de Figura_2,
          Figura_3 dentro_de Figura_2,contorno).

figura(triángulo dentro_de círculo).
figura(círculo dentro_de triángulo).
figura(cuadrado dentro_de triángulo).
figura(círculo dentro_de cuadrado).
figura(cuadrado dentro_de círculo).
figura(triángulo dentro_de cuadrado).
figura(triángulo dentro_de triángulo).
figura(círculo dentro_de círculo).
```

que, por ejemplo, ante la pregunta:

```
:- X es_a triángulo dentro_de círculo
   como
   cuadrado dentro_de círculo es_a Y
   mediante
   Relación.
```

proporciona nada menos que 17 respuestas diferentes.

9 Aprendizaje automático

Tal y como ya hemos comentado, una característica propia de la programación lógica es su capacidad natural para la manipulación simbólica mediante la unificación. Si a ello sumamos el hecho de que en un programa lógico no existe una diferencia real entre datos y objetos, siendo todos gestionados de idéntica forma en razón de su estructura arborescente, tendremos la herramienta perfecta para abordar uno de los problemas recurrentes en inteligencia artificial, la interpretación de programas que se modifican a si mismos.

En nuestro contexto, modificar un programa supone introducir o eliminar cláusulas de su base de datos. A este respecto, los intérpretes de programación lógica proveen un conjunto de primitivas que permiten su puesta en marcha:

- El predicado `asserta/1` se verifica siempre, siendo su efecto colateral el de introducir al principio de nuestro programa la cláusula que le sirve de argumento. El predicado `assertz/1` realiza, sin embargo, la introducción de la nueva cláusula al final del programa.
- En cuanto al predicado `retract/1`, éste se verifica también siempre, siendo su efecto colateral en esta ocasión el de retirar de la base de datos de nuestro programa la cláusula que le sirve de argumento. La búsqueda se realiza comenzando por el principio del programa.

Este tipo de predicados resulta de utilidad en programas que ofrezcan al usuario la posibilidad de considerar dinámicamente la introducción de restricciones, o en aquellos en los que la repetición de esquemas recursivos múltiples aconsejen la generación de nuevas cláusulas que eviten la multiplicación de cálculos que previamente hemos realizado.

Ejemplo 91 *Las* Torres de Hanoi *son un conocido juego, en el que se parte de un escenario con los siguientes elementos:*

- *Tres palos en posición vertical, que denominamos* `A`*,* `B` *y* `C`*.*
- *Un conjunto de discos de diferente diámetro, que pueden ser insertados en cualquiera de los palos.*
- *Inicialmente todos los discos están insertados en el palo* `A`*, ordenados de mayor a menor diámetro, comenzando por la posición más baja.*

tal y como se muestra en la figura 16. Definido el escenario, el juego consiste en mover los discos desde el palo inicial `A`*, a un palo de destino* `B`*, sirviendo el tercer palo* `C` *de paso temporal de los discos en su movimiento. En todo momento debe satisfacerse, cualquiera que sea el palo considerado, la condición siguiente:*

> "Sobre un disco cualquiera solamente pueden colocarse discos de menor diámetro."

La complejidad del problema es de 2^N *movimientos para* N *discos. Por tanto, en el caso* $N = 3$ *mostrado en la figura 16, el número de movimientos sería del orden de* $2^3 = 8$*, lo que refiere a una complejidad exponencial. En cuanto al algoritmo, éste se resume en el siguiente conjunto de cláusulas:*

1. *Mover* `N-1` *discos de* `A` *hacia* `C`*, utilizando* `B` *como palo intermedio. Con ello dejamos en* `A` *un único disco: el que estaba inicialmente en la base.*
2. *Mover el disco situado en* `A` *a* `B`*.*
3. *Mover los* `N-1` *discos situados en* `C` *a* `B`*, usando* `A` *como palo intermedio.*

lo que claramente establece una estrategia doblemente recursiva en la que los pasos 1. y 3. son análogos, tal y como se reflejará en la implementación, para la cual consideraremos el predicado `hanoi/5`*, cuya semántica declarativa viene expresada por:*

> "`hanoi(N, A, B, C, Movs)` es cierto sii `Movs` es la serie de movimientos a realizar para trasladar `N` discos desde el palo `A` al `B`, tomando el palo `C` como paso intermedio"

obteniendo como resultado posible, el programa siguiente:

```
:- op(600,yfx,a).

concat([],L,L) :- !.
concat([Car|Cdr],L,[Car|R]) :- concat(Cdr,L,R).

hanoi(1,A,B,_,[A a B]).
hanoi(N,A,B,C,Movs) :- N > 1, N1 is N - 1,
                       hanoi(N1,A,C,B,Movs_1),
                       hanoi(N1,C,B,A,Movs_2),
                       concat(Movs_1,[A a B|Movs_2],Movs).
```

en el que hacemos uso de la concatenación de listas a través del predicado auxiliar `concat/3`*, y del operador infijo "*`a`*" para facilitar la lectura de la salida. Evidentemente, la primera cláusula de* `hanoi/5` *describe el caso trivial, cuando en la columna* `A` *de partida sólo hay un disco.*

La idea ahora no es otra que la de aprovechar la analogía ya descrita entre las dos llamadas recursivas de `hanoi/5`*, de manera a evitar* de facto *los cálculos que corresponderían a la segunda de esas llamada. El nuevo código sería entonces:*

```
:- op(600,yfx,a).

concat([],L,L) :- !.
concat([Car|Cdr],L,[Car|R]) :- concat(Cdr,L,R).

hanoi(1,A,B,_,[A a B]).
      hanoi(N,A,B,C,Movs):- N>1, N1 is N-1,
                            hanoi(N1,A,C,B,Movs_1),
                            asserta((hanoi(N1,A,C,B,Movs_1):-!)),
                            hanoi(N1,C,B,A,Movs_2),
                            retract((hanoi(N1,A,C,B,Movs_1):-!)),
                            concat(Movs_1,[A a B|Movs_2],Movs).
```

de esta forma, no sólo evitamos la multiplicación de cálculos mediante el uso de `asserta/1`*, sino que una vez estamos seguros de que éstos han cumplido su función, las cláusulas introducidas dinámicamente a tal efecto son eliminadas.*

10 Ejercicios propuestos

1. Implementar un predicado `fib(X,Y)` que verifique:
   (a) Que `Y` es el valor de la función de Fibonacci sobre `X`.
   (b) Que evita el cálculo reiterado de valores `fib(X',Y')` donde `X'< X`.
   (c) Que deje intacto el programa inicial.

La función de Fibonacci se define por:

$$\texttt{Fibonacci(X)} = \begin{cases} 0 & \textbf{si } \texttt{X=0} \\ 1 & \textbf{si } \texttt{X=1} \\ \texttt{Fibonacci(X-1) + Fibonacci(X-2)} & \textbf{en otro caso} \end{cases}$$

2. Implementar un predicado `mult(F_1,F_2,R)`, tal que se verifique cuando `R` sea el resultado del producto de los factores `F_1` y `F_2`.
   Ejemplo: La respuesta, expresada usando la notación `suc/1` para representar los naturales, a `:- mult(suc(suc(0)),suc(suc(suc(0))),X).`, es `X=suc(suc(suc(suc(suc(suc(0))))))`.
3. Implementar un predicado `invertir(L,R)` que sea cierto cuando `R` es la lista resultante de invertir el orden de los elementos en la lista `L`.
   Ejemplo: La respuesta a la pregunta `:- invertir([1,a],X).`, es `X=[a,1]`.
4. Implementar un predicado `longitud(L,R)` que sea cierto cuando `R` sea la longitud de la lista `L`.
   Ejemplo: La respuesta a la pregunta `:- longitud([1,a],X).`, es `X=2`.
5. Implementar un predicado `elimina(E,L,R)` que se verifique cuando `R` es la lista resultante de eliminar todas las apariciones del elemento `E` en la lista `L`.
   Ejemplo: La respuesta a la pregunta `:- elimina(1,[1,a,1,2],X).`, es `X=[a,2]`.
6. Implementar un predicado `assertb` con la funcionalidad de `asserta`, pero que en caso de retroceso elimine del universo del discurso la cláusula antes introducida.
7. Implementar un predicado `extrae(L,P,E)` que se verifique cuando `E` es el elemento de la lista `L` en la posición `P`.
   Ejemplo: La respuesta a la pregunta `:- extrae([1,a,1,2],2,X).`, es `X=a`.
8. Implementar un predicado `exp(X,Y,Z)` que se verifique cuando $\texttt{X}^{\texttt{Y}} = \texttt{Z}$.
9. Implementar un predicado `inserta(L,E,P,R)` que se verifique sii `R` es la lista resultante de insertar el elemento `E` en la posición `P` de la lista `L`.
   Ejemplo: La respuesta a la pregunta `:- inserta([1,2,3,4],a,2,X).`, es `X=[1,a,2,3,4]`.
10. Implementar un predicado `ordenar(L,R)`, que ordene por inserción la lista numérica `L`, para calcular la lista ordenada `R`.
    Ejemplo: La respuesta a la pregunta `:- ordenar([4,1,2,3],X).`, es `X=[1,2,3,4]`.
11. Implementar un predicado `aplanar(L,R)` que aplane la lista `L`, eliminando imbricaciones sobre sus elementos e instanciando la respuesta en `R`.
    Ejemplo: La respuesta para `:- aplanar([4,1,[2],[3,[4,5]],X).`, es `X=[4,1,2,3,4,5]`.
12. Resolver el problema de cripto-aritmética definido por la operación numérica siguiente:

```
  SEND
+ MORE
------
 MONEY
```

Esto es, implementar un programa que asigne los valores posibles a las letras en juego de manera que la suma sea posible. Suponer que los dígitos asignados a letras diferentes son asimismo diferentes.

13. Implementar un predicado `comprimir(L,C)` tal que `C` es la lista resultado de comprimir la lista `L`, al eliminar repeticiones consecutivas de un elemento dado.
    Ejemplo: La respuesta a `:- comprimir([a,a,b,c,c,a,a,d,e,e,e],X)` sería `X = [a,b,c,a,d,e]`.
14. Implementar un predicado `codificar(L,R)` tal que `R` es la lista resultante de codificar la lista `L` de tal forma que una sucesión de elementos repetidos consecutivos en `Lista` se sustituye por un par de la forma `[Longitud, Elemento]` con `Longitud` la longitud de la serie de repeticiones consecutivas de `Elemento`.
    Ejemplo: La respuesta a `:- codificar([a,a,b,c,c,a,a,d,e,e,e,],X).`, sería `X = [[2,a],[1,b],[2,c],[2,a],[1,d][3,e]]`.
15. Implementar un predicado `multiplicar_n(L,N,R)`, tal que `R` es la lista obtenida a partir de la lista `L` al repetir `N` veces cada elemento en `L`.
    Ejemplo: La respuesta a `:- multiplicar_n([a,b,c],3,X).`, sería `X = [a,a,a,b,b,b,c,c,c]`.
16. Implementar un predicado `extraer_n(L,P_1,P_2,R)`, tal que `R` es la lista resultante de extraer de la lista `L` la sublista formada por los elementos situados entre las posiciones `P_1` y `P_2`.
    Ejemplo: La respuesta a `:- extraer_n([a,b,c,d,e,f,g,h,i,k],3,7,X).`, es `X= [c,d,e,f,g]`.
17. Supongamos que un conjunto se representa mediante una lista en la que no aparezcan elementos repetidos. En estas condiciones, implementar los siguientes predicados:
    (a) `conjunto(X)`, que sea cierto si `X` es un conjunto, esto es, una lista sin elementos repetidos.
    (b) `intersección(C_1,C_2,R)`, tal que `R` es el resultado de la intersección de los conjuntos `C_1` y `C_2`.
    (c) `unión(C_1,C_2,R)`, tal que `R` es el resultado de la unión de los conjuntos `C_1` y `C_2`.
    (d) `cartesiano(C_1,C_2,R)`, tal que `R` es el resultado del producto cartesiano de los conjuntos `C_1` y `C_2`.
    (e) `dif_simétrica(C_1,C_2,R)`, tal que `R` es el resultado de la diferencia simétrica de los conjuntos `C_1` y `C_2`. La diferencia simétrica de dos conjuntos es su unión, menos su intersección.

Capítulo 5.

Las redes neuronales: consideraciones prácticas

1. Introducción a las redes neuronales

El cerebro está compuesto de neuronas, las cuales son elementos individuales de procesamiento. Una red neuronal es un método de computación inspirado en modelos biológicos. Las redes neuronales artificiales ambicionan imitar la operación básica del cerebro. La información viaja entre las neuronas, y basado en la estructura y ganancia de los conectores neuronales, la red se comporta de forma diferente.

1.1. Conceptos básicos

El cerebro humano contiene aproximadamente $100,000$ millones de neuronas. Cada neurona está conectada aproximadamente a otras 1000 neuronas, excepto en la corteza cerebral donde la densidad neuronal es mucho mayor. En las redes neuronales artificiales, cada neurona está conectada con otra neurona por medio de un peso o coeficiente de ajuste representado por la letra w, el primer subíndice indica el número de la neurona destino, mientras que el segundo subíndice indica el número de la neurona de origen. Ya que en cada capa las neuronas se numeran comenzando con el número uno, es muy importante siempre indicar a que capa pertenece la neurona en cuestión. La Figura 1 muestra una red neuronal de dos capas.

Los pesos o coeficientes que unen la entrada de la red con las neuronas de entrada se conocen como pesos de entrada. Para redes que tienen más de una capa, los pesos que unen las neuronas internas se conocen como pesos de capa. MATLAB tiene una caja de herramientas para simular redes neuronales, en esta caja de herramientas los pesos de entrada se denotan por IW mientras que los pesos de capas internas por LW.

Una neurona artificial está formada por un sumador y una función de activación, representada por $f(y)$ como se muestra en la Figura 2. Note que la señal de polarización siempre tiene un valor de 1 (denotada por *Bias* en la figura.) La

Figura 1. Red neuronal de dos capas mostrando la conexión entre neuronas.

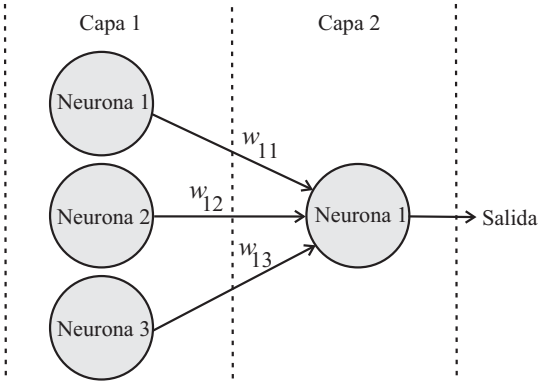
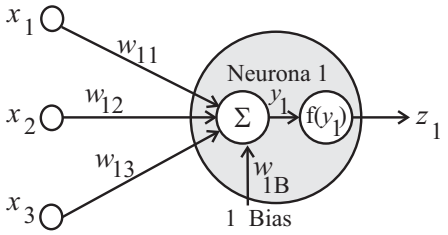


Figura 2. Red neuronal de tres entradas y una salida



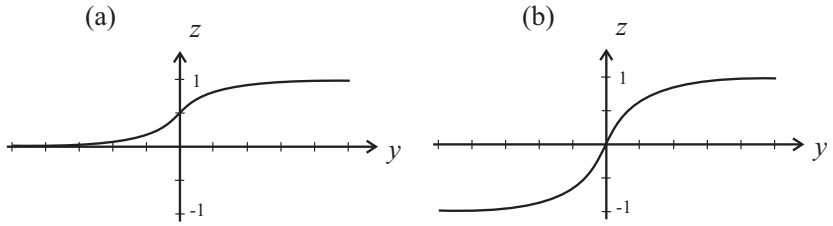
función de activación $f(y)$ debe ser un sigmoide (tener forma de S), continua, real, de rango acotado y tener una derivada positiva. Las funciones más populares, en las redes de varias capas, son la tangente hiperbólica y la función logística:

$$f(y) = logsig(y) = \frac{1}{1+e^{-y}} \tag{1}$$

$$f(y) = tanh(ay) \tag{2}$$

donde a es una constante positiva con un valor típico de 1.5. La Figura 3 muestra las funciones de activación más usadas en las redes neuronales.

Figura 3. Funciones de activación neuronal: (a) $z = logsig(y)$, (b) $z = tanh(ay)$.



Antes de continuar, es muy importante mencionar que hoy en día las redes neuronales son entrenadas para resolver problemas que son difíciles para las computadoras convencionales o los seres humanos.

Problema 1. Encuentre la derivada $f'(y)$ de la función de activación $f(y) = logsig(y)$. Exprese su resultado en función de $f(y)$.

Solución 1.

$$f'(y) = (-1)\frac{-e^{-y}}{(1+e^{-y})^2} = \frac{1}{(1+e^{-y})}\frac{e^{-y}}{(1+e^{-y})}$$

$$= f(y)\frac{1-1+e^{-y}}{(1+e^{-y})} = f(y)[-f(y)+1] = f(y)([1-f(y)]$$

Problema 2. Encuentre la salida, z_1, de la red neuronal de la Figura 2 cuando la entrada es x y los coeficientes de la red son W, suponga que $f(y) = logsig(y)$.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 9 \\ -10 \end{bmatrix} \tag{3}$$

$$W = \begin{bmatrix} w_{11}\ w_{12}\ w_{13}\ w_{1B} \end{bmatrix} = \begin{bmatrix} -2\ 4\ 5\ 11 \end{bmatrix} \tag{4}$$

Solución 2.

$$y_1 = (7)(-2) + (9)(4) + (-10)(5) + (1)(11) = 17$$

$$z_1 = \frac{1}{1 + e^{17}} = 4.14 \times 10^{-8}$$

En el caso más general, ver Figura 4, cuando la entrada a la red está data por x y los coeficientes de la red por W,

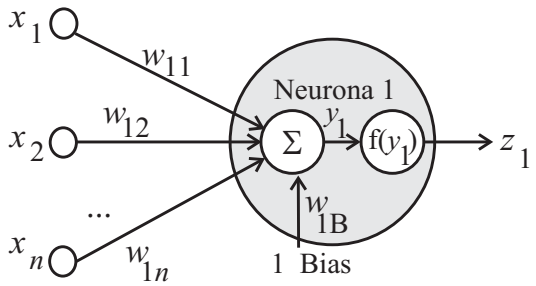
$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix}, \tag{5}$$

$$W = \begin{bmatrix} w_{11}\ w_{12}\ w_{13}\ \dots\ w_{1B} \end{bmatrix}, \tag{6}$$

la salida de la red está determinada por

$$z_1 = f(y_1) = f(W \begin{bmatrix} x \\ 1 \end{bmatrix}). \tag{7}$$

Figura 4. Red neuronal de n entradas y una salida



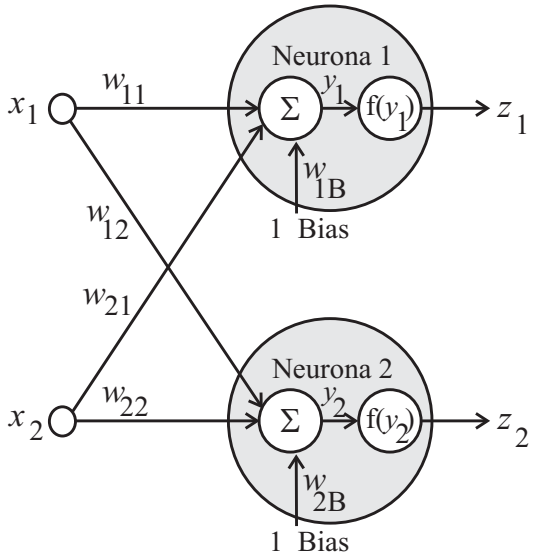
Una red neuronal puede tener múltiples salidas y entradas; cada neurona en la capa de salida corresponde a una salida de la red como se muestra en la Figura 5. En este caso la matriz de coeficientes toma la forma

$$W = \begin{bmatrix} w_{11}\ w_{12}\ w_{1B} \\ w_{21}\ w_{22}\ w_{2B} \end{bmatrix}, \tag{8}$$

y la salida está determinada por

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = f(W \begin{bmatrix} x \\ 1 \end{bmatrix}). \tag{9}$$

Figura 5. Red neuronal de dos entradas y dos salidas



Problema 3. Encuentre la salida de la red con capa escondida de la Figura 6 cuando la entrada está definida por u, y los coeficientes de la red por H y W como se muestra. Use la función de activación $f(y) = logsig(y)$ para ambas capas.

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.5 \end{bmatrix}$$

$$H = \begin{bmatrix} \text{-}4.8 & 4.6 & \text{-}2.6 \\ 5.1 & \text{-}5.2 & \text{-}3.2 \end{bmatrix}$$

$$W = \begin{bmatrix} 5.9 & 5.2 & 30.3 \end{bmatrix}$$

Solución 3.

$$v_1 = (0.1)(\text{-}4.8) + (0.5)(4.6) + (1)(\text{-}2.6) = \text{-}0.780$$

$$x_1 = logsig(\text{-}0.780) = 0.314$$

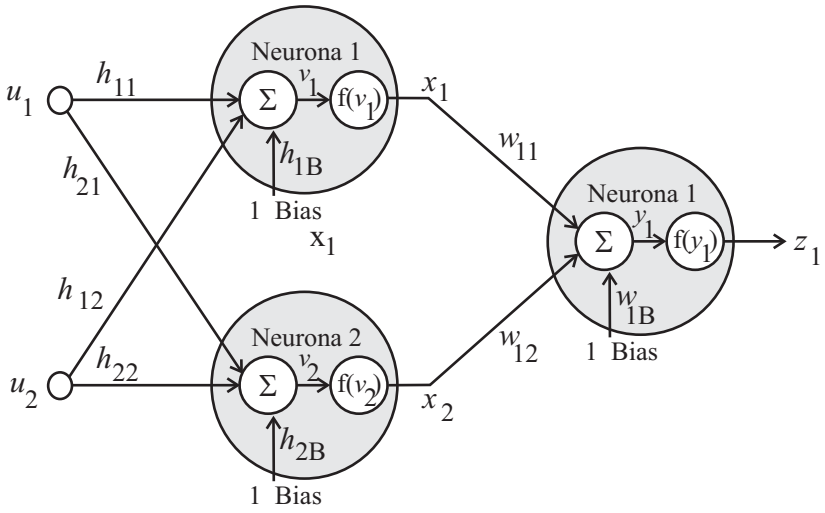
$$v_2 = (0.1)(5.1) + (0.5)(\text{-}5.2) + (1)(\text{-}3.2) = \text{-}5.29$$

$$x_2 = logsig(\text{-}5.29) = 0.00502$$

$$y_1 = (0.314)(5.9) + (0.00502)(5.2) + (1)(30.3) = 32.2$$

$$z_1 = logsig(32.2) = 1.00$$

Figura 6. Red neuronal de dos entradas, una salida, y una capa escondida



Durante el entrenamiento de una red neuronal las neuronas pueden activarse miles de veces haciendo extremadamente importante optimizar la velocidad de operación de las neuronas. Una forma de optimizar la operación de una neurona es acelerando el cálculo de la función de activación por medio:

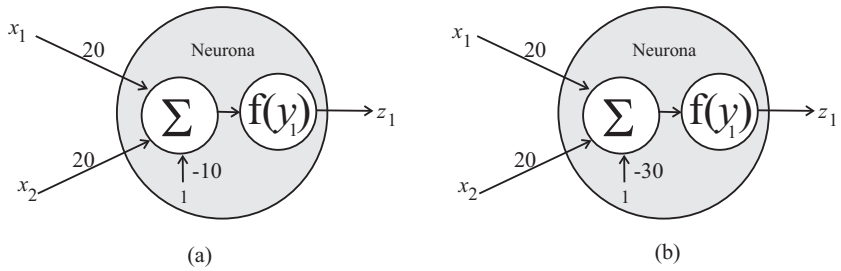
- De una tabla que contiene un conjunto de valores uniformemente distribuidos de la función de activación. Para mejorar la precisión de este método se puede usar interpolación lineal, agregando una segunda tabla con las diferencias entre valores consecutivos de la función de activación.
- Del uso de las propiedades de la función sigmoide, la cual establece que para un valor Y dado, la función de activación tiene un valor aproximado de 1. De igual forma se puede decir que para valores menores a $-Y$, la función de activación $f(y) = logsig(y)$ toma un valor de 0, mientras que la función $f(y) = tanh(ay)$ toma un valor -1.

1.2. Modelando las compuertas lógicas

Una forma sencilla de ilustrar la operación de una red neuronal es simulando las compuertas lógicas como se muestra en la Figura 7. Es importante notar que los coeficientes de la red no son únicos. Se invita al lector a comprobar que las redes neuronales producen las salidas deseadas.

Problema 4. Cree una red neuronal de una sola neurona para simular la compuerta lógica OR usando MATLAB.

Figura 7. Simulación de operaciones lógicas: (a) OR, (b) AND.



Solución 4. Para crear una red neuronal multi-capa se usa el comando `newff`. En su forma básica, este comando toma tres parámetros: la matriz de rangos de entrada, la matriz de la estructura de la red y el tipo de la función de activación a usar. En el caso de una compuerta lógica de dos entradas, la matriz de rangos contiene dos renglones: el primer renglón indica el rango de la primera variable de entrada, el segundo renglón muestra el rango de la segunda variable de entrada. Como sólo se requiere una neurona para implementar la red, la matriz de la estructura de la red contiene un sólo elemento (un uno). El comando mostrada crea un red neuronal con una única neurona con función de activación $z = logsig(y)$ y almacena la red creada en la variable `net`.

```
net = newff([0 1; 0 1], [1], {'logsig'});
```

Una vez que la red se ha creado se procede a ajustar los coeficientes de la red.

```
net.IW {1,1} =[20, 20];
net.b{1}=[-10];
```

Finalmente, se crea la entrada x de la red y se procede a calcular la salida usando el comando de MATLAB `sim`. Como resultado de la simulación se obtiene la salida de la compuerta lógica OR en la variable z.

```
x=[0, 0, 1, 1; 0, 1, 0, 1];
z=sim(net, x);
```

Aunque es posible ajustar los coeficientes de una red neuronal en forma manual, ésto no es un procedimiento típico o recomendado. En su lugar, la red debe recibir entrenamiento de tal forma que sus coeficientes sean ajustados automáticamente para adaptarse a los distintos eventos contenidos en el entrenamiento.

Problema 5. Diseñe una red neuronal en MATLAB para que ésta aprenda la compuerta lógica OR usando el método del gradiente conjugado.

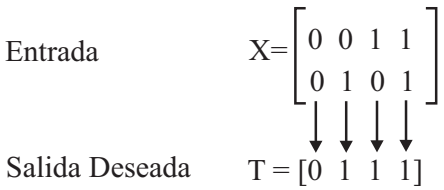
Solución 5. Primeramente se debe crear la red neuronal usando el comando de MATLAB `newff` indicando que método se utilizará durante el entrenamiento, MATLAB representa el método del gradiente conjugado como `traingdx`.

```
net = newff([0 1; 0 1], [1], {'logsig'}, 'traingdx');
```

Posteriormente se debe crear el conjunto de datos de entrenamiento como se muestra en la Figura 8, el cual contiene cuatro casos de entrenamiento (training cases).

```
X=[0 0 1 1; 0 1 0 1];
T=[0 1 1 1];
```

Figura 8. Conjunto de datos de entrenamiento para la compuerta OR.



El entrenamiento de una red es un procedimiento largo que requiere de varios parámetros de configuración, los parámetros más importantes de entrenamiento son el número de 'epochs' (iteraciones) y el 'goal' (error deseado). Antes de iniciar el entrenamiento se deben ajustar estos valores, de otro modo, los valores pre-determinados serán usados.

```
net.trainParam.epochs=10000;
net.trainParam.goal=0.0001
```

Una vez que el procedimiento de entrenamiento se ha configurado, la red puede entrenarse como se muestra. Es importante notar que el entrenamiento puede terminar por varias razones. Siendo indispensable verificar si al final del entrenamiento la red alcanzó el error deseado.

```
net = train(net, X, T);
```

Una vez terminado el entrenamiento se procede a simular la red para verificar su operación. Una tendencia típica al revisar los resultados de una simulación es esperar que los valores de salida obtenidos sean idénticos a los deseados, usualmente siempre existirá una ligera diferencia como se puede apreciar debajo.

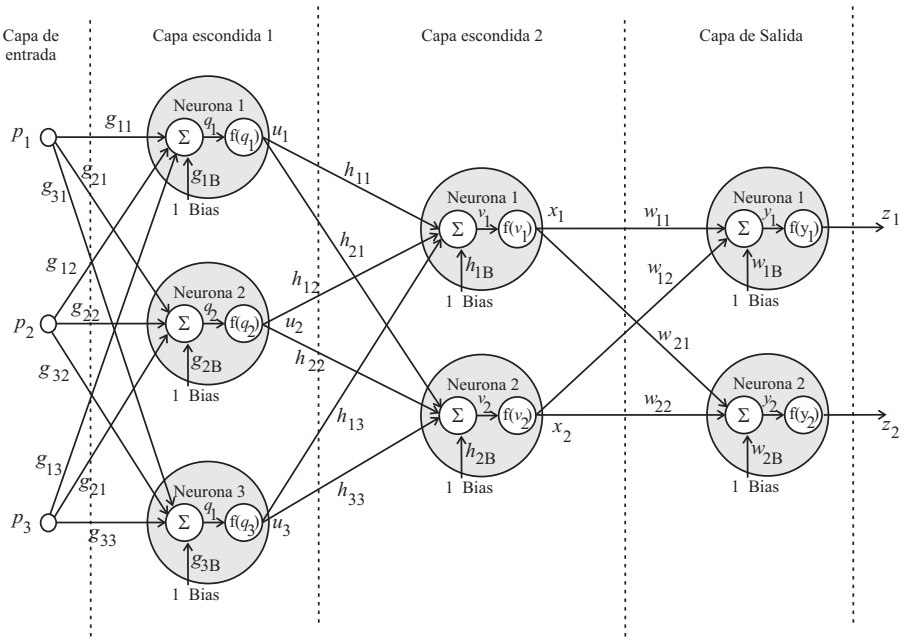
```
sim(net, X)
ans =  0.0149    0.9906    0.9907    1.0000
```

1.3. Redes de varias capas y su entrenamiento

Sería imposible hablar de redes neuronales, sin mencionar a su creador. Rosenblatt diseñó la primera red neuronal artificial en 1958 (el perceptrón). Desafortunadamente dos compañeros de clases, muy respetados y reconocidos, publicaron un análisis matemático que explicada las deficiencias de las redes neuronales en 1969. Debido a esto y a la muerte de Ronsenblatt en 1971, los fondos para investigación en redes neuronales se terminaron. En 1986, se demostró que las redes neuronales podían se usadas para resolver problemas prácticos y fue entonces que éstas comenzaron ha ser importantes.

En general una red puede tener una, dos o más capas. La Figura 9 muestra una red de tres entradas, dos salidas y dos capas escondidas.

Figura 9. Red de dos capas escondidas



Los resultados obtenidos por una red neuronal dependen en su mayoría del entrenamiento de la misma. El proceso de entrenamiento consiste en crear un conjunto de datos que le permitan a la red aprender. Posteriormente, se debe proceder a crear otro conjunto de datos semejante al de entrenamiento, conocido como conjunto de datos de validación. El procedimiento usando para la creación de estos conjuntos de datos determina el éxito del uso de redes neuronales. A

continuación se enumeran los pasos más comunes para el diseño y uso de una red neuronal:

1. Crear el conjunto de datos de entrenamiento conocido en Inglés como "Training Set".
2. Crear el conjunto de datos validación conocido en Inglés como "Validation Set".
3. Crear la red.
4. Entrenar la red (usando el conjunto de datos de entrenamiento).
5. Validar la red para averiguar si aprendió y generalizó (usando el conjunto de datos de validación.)
6. Usar la red aplicando datos nuevos, posiblemente diferentes a los de entrenamiento y validación.

En redes neuronales se utiliza la palabra 'Generalizar' para expresar la habilidad de la red para trabajar con datos diferentes a los del conjunto de datos de entrenamiento. Un efecto adverso llamado sobre-ajuste (overfitting en Inglés) ocurre cuando la red opera adecuadamente con el conjunto de datos de entrenamiento, pero se comporta mal con otros conjuntos de datos. Este efecto indeseable es un síntoma de un número excesivo de neuronas o un conjunto de datos de entrenamiento pobre.

2. Usando redes neuronales

Existen varios procesos de normalización comunes en procesamiento de datos para transformar los datos a valores más apropiados para su manipulación y análisis.

2.1. Pre/Post procesamiento de datos

Las redes neuronales pueden ser más eficientes si pre-procesamiento de datos es aplicado a los datos de entrada y de salida. Idealmente y dependiendo del tipo de red, el rango de valores de operación de una red puede ser de 0 a 1, ó -1 a 1. Adicionalmente, en la práctica el rango de 0 a 1 se reduce a sólo de 0.1 a 0.9, mientras que el rango de -1 a 1 se reduce de -0.9 a 0.9. Este proceso se conoce como normalización de rango.

Una vez que la red ha sido entrenada, la red puede ser usada en aplicaciones reales por medio de un procedimiento llamado simulación o activación de la red. Así, para simular una red, todos los datos de entrada deben ser ajustados al rango de la red. De igual modo, los datos de salida de la red deben ser transformados al rango original de los datos.

Para realizar la normalización de rango MATLAB cuenta con las funciones `premnmx` y `postmnmx`, la función `tramnmx` es usualmente usada en pre-procesamiento de normalización de rango. Para ilustrar como usar estos comando se creará un vector X con valores uniformemente espaciados entre $[-4\ \ 4]$, así mismo se crea el vector Y como se muestra.

Figura 10. Normalización de rango para una red con función de activación (a) $z = logsig(y)$, (b) $z = tanh(ay)$


```
X=[-4:0.1:4];
T=2*sin(X);
```

Se procede ahora a aplicar el pre-procesamiento usando el comando `premnmx`, el cual regresa seis parámetros: los valores normalizados de X y T, así como los valores mínimos y máximos de cada vector. Adicionalmente se ejecuta el comando `minmax` de MATLAB para verificar que los rangos de XN y TN se han ajustado en forma apropiada, observe que, a fin de mostrar los valores mínimos y máximos de estos vectores, las últimas dos líneas no terminan con punto y coma.

```
[XN, minx, maxx, TN, mint, maxt]=premnmx(X, T);
minmax(XN)
minmax(TN)
```

Una vez que los datos han sido normalizados se procede a usarlos y finalmente se aplica el post-procesamiento. Para verificar que el proceso de normalización no implica ninguna pérdida de información se calcula la suma de las diferencias entre los valores originales y los recuperados como se muestra. Este valor debe ser un número pequeño.

```
RECUPERADO = postmnmx(TN, mint, maxt);
sum(abs(T-RECUPERADO))
```

Otro método de pre-procesamiento consiste en transformar los datos de tal forma que la media de los datos sea de 0 y la varianza sea 1. Una vez terminado el entrenamiento se debe aplicar el post-procesamiento para regresar al dominio original de los datos. Este tipo de pre-procesamiento es conocido como normalización. Los comandos de MATLAB para este tipo de pre/post procesamiento son `prestd` y `poststd` respectivamente.

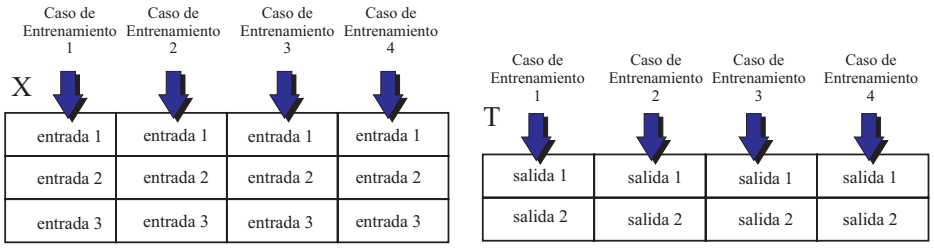
Antes de terminar de hablar de pre-procesamiento es importante mencionar que las funciones `premnmx` y `postmnmx` de MATLAB no utilizan el rango práctico de operación neuronal (0.1 a 0.9 o -0.9 a 0.9) por lo que en la mayoría de los casos el pre/post-procesamiento se debe realizar manualmente.

2.2. Entrenando una red neuronal para las funciones trigonométricas

El proceso de entrenamiento de una red requiere que se tenga un conjunto de datos de entrenamiento (training set); éste debe contener un número apropiado de casos de entrenamiento (training cases). Por otro lado, el conjunto de datos de entrenamiento debe seguir el formato requerido por el programa que se utiliza para simular las redes neuronales. El formato del conjunto de datos de entrenamiento en MATLAB se muestra en la Figura 11.

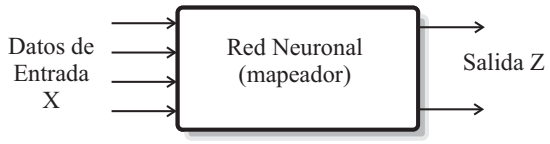
Típicamente, el conjunto de datos de entrenamiento contiene dos grupos de datos: los datos de entrada y los datos de salida deseados; representados por X y T en la Figura 11 respectivamente. Debido a las propiedades de las funciones de activación neuronal, es muy importante escalar los datos de salida deseados al rango de 0.1 a 0.9 cuando se use la función $z = logsig(y)$ y de -0.9 a 0.9 cuando se use la función $z = tansig(y)$. Por otro lado, aunque el rango de los datos de entrada no es crítico, también se recomienda escalar éstos al rango de -1 a 1.

Figura 11. Formato usado en MATLAB para el conjunto de datos de entrenamiento



Una red neuronal puede ser usada como un mapeador como se muestra en la Figura 12. La red es entrenada para cada uno de los casos de entrenamiento aplicando los datos de entrada X a la red y especificando la salida deseada T. El proceso de entrenamiento debe disminuir el error entre la señal de salida de la red Z y la deseada T.

Figura 12. Operación de una red neuronal como mapeador.



Problema 6. Diseñe una red del tipo mapeador con una entrada y dos salida para aprender las funciones seno y coseno usando MATLAB como se muestra en la figura. Las neuronas de la red deben usar la función de activación $z = tansig(y)$ y debe operar para valores de x en el rango de $-\pi$ a π.

Solución 6. Para crear la red se crea el archivo `SenoCoseno.m` mostrado debajo. El archivo comienza borrando cualquier otra variable previamente definida antes de ejecutar el archivo m. Se procede entonces a crear los datos de entrada X y los datos de salida deseados T. Una vez que se tiene el conjunto de datos de entrenamiento (formado por las variables X y T en este caso) se aplica el proceso de normalización usando el comando `premnmx`. A continuación se crea la red y se inicia el entrenamiento por medio del algoritmo de minimización de Levenberg Marquardt y el conjunto de datos de entrenamiento normalizado (formado por las variables XN y TN.)

```
clear;
X=[-pi: 0.01 : pi];
T= [sin(X); cos(X)];
[XN,minx,maxx,TN,mint, maxt] = premnmx(X, T);
net = newff([-1 1],[6, 2], {'tansig', 'tansig'}, 'trainlm');
%
net.trainParam.goal=0.0001;
net.trainParam.epochs=1500;
net=train(net, XN, TN);
```

Debido a que durante el entrenamiento se usan procedimientos aleatorios, es recomendable realizar múltiples entrenamientos (ejecuciones del comando train de MATLAB) y escoger aquel en él que se consiguió un error menor.

Una vez terminado el entrenamiento se procede a la simulación de la red. En este caso, se usarán otros valores de entrada X para los cuales la red no fue entrenada como se muestra en la lista de comandos de MATLAB.

```
X=[-pi: 0.005 : pi];
XN= tramnmx(X, minx, maxx);
YN = sim(net,XN);
Y = postmnmx(YN, mint, maxt);
```

Finalmente se procede a graficar la salida de la red y la salida deseada.

```
plot(Y');
hold on;
T = [sin(P); cos(P)];
plot(T');
hold off;
```

Problema 7. Diseñar una red neuronal usando MATLAB para aprender las funciones seno y coseno en el rango de $-\pi$ a π. Usar la función de activación $z = logsig(y)$.

Solución 7. Primero se debe crear el conjunto de datos de entrenamiento X, TN, recordando que TN debe encontrarse en el rango de 0.1 a 0.9 y que por lo tanto no es posible usar las funciones `premnmx` o `postmnmx` de MATLAB.

```
X=[-pi: 0.01 : pi];
T= [sin(X); cos(X)];
TN = (T+1).*(0.8/2)+0.1;
```

Luego se procede a crear y entrenar la red como se muestra.

```
net = newff([-pi pi], [6, 2], {'logsig', 'logsig'}, 'trainlm');
net.trainParam.goal=0.0001;
net.trainParam.epochs=10000;
net=train(net, X, TN);
```

Repita el proceso de entrenamiento o ajuste el número de neuronas en la capa escondida de la red hasta obtener el menor error posible.

Se procede a simular la red con la entrada X, la cual produce la salida YN. Entonces se desnormaliza YN para obtener valores en el dominio de los datos entre -1 y 1. Finalmente, se grafican la salida deseada y la salida producida por la red.

```
YN = sim(net, X);
Y = (YN-0.1).*(2/0.8)-1;
plot(Y');
hold on;
TR =[sin(X); cos(X)];
plot(TR');
hold off;
```

3. Consideraciones prácticas

Aunque siempre es indispensable tener buenas bases teóricas, el uso de redes neuronales requiere la consideración de factores prácticos en la implementación de las mismas. Éstos serán discutidos en detalle en esta sección.

3.1. El conjunto de datos de entrenamiento

El éxito del uso de una red neuronal radica principalmente en el diseño del conjunto de datos de entrenamiento. Se revisarán a continuación los principales factores que hay que tomar en cuenta para el diseño de éste.

1. Sobre-entrenamiento. Un síntoma de sobre entrenamiento es que la red trabaja muy bien con el conjunto de datos de entrenamiento pero produce malos resultados con el conjunto de datos de validación.

2. El conjunto de datos de validación y de entrenamiento debe representar en forma apropiada el experimento. Éstos debe contener todos los distintos tipos casos de entrenamiento (training cases) existentes en el problema real a resolver.
3. Bajo ninguna circunstancia, se puede usar el conjunto de datos de validación para entrenamiento. Esto sería equivalente a robar el examen y estudiar de éste en lugar de usar las notas de clase.
4. El conjunto de datos de entrenamiento no debe ser más grande que lo necesario, esto es, no debe contener casos de entrenamiento repetidos.
5. Las redes más grandes requieren conjuntos de datos de entrenamiento más grandes.
6. El conjunto de datos de entrenamiento no debe contener desviaciones creadas por factores humanos; estas desviaciones serán aprendidas por la red.
7. El conjunto de datos de entrenamiento puede obtenerse a partir de un conjunto de datos muy grande y un generador de números aleatorios, para seleccionar en forma aleatoria parte de los datos del conjunto original.
8. El conjunto de datos de entrenamiento debe ser escalado apropiadamente para acoplarse a las funciones de activación de las neuronas.

3.2. Diseño de redes neuronales

Para el diseño de una red neuronal se requiere determinar el número de neuronas así como el número de capas en la red. A continuación se mencionan algunos puntos a considerar durante el diseño de una red neuronal multicapa.

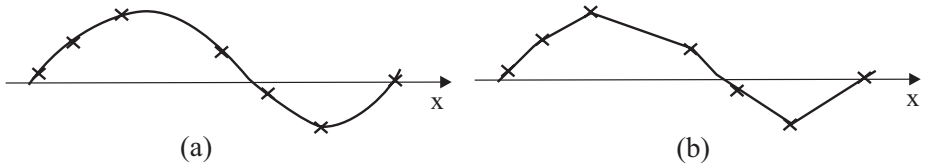
1. Se debe comenzar con cero niveles escondidos. Si el problema no se resuelve adecuadamente, se debe intentar con un solo nivel escondido y el menor número de neuronas. Si un número grande de neuronas escondidas en este nivel no resuelven el problema satisfactoriamente, entonces se puede intentar incrementar el número de neuronas en el segundo nivel y posiblemente reducir el número total de neuronas. Usar más de dos niveles escondidos no es conveniente, ya que se inestabiliza el proceso de entrenamiento debido al incremento en el número de falsos mínimos.
2. Solo se deben usan dos niveles escondidos cuando la función a aprender presenta discontinuidades.
3. Dentro de lo que el tiempo lo permita, entrenar hasta conseguir el menor error.

Un número excesivo de neuronas producirá el aprendizaje de efectos particulares que no son generales entre todas las muestras, lo cual no es deseable. Esto se conoce como sobre ajuste (en Inglés Over fitting), ver Figura 13.

3.3. El proceso de entrenamiento

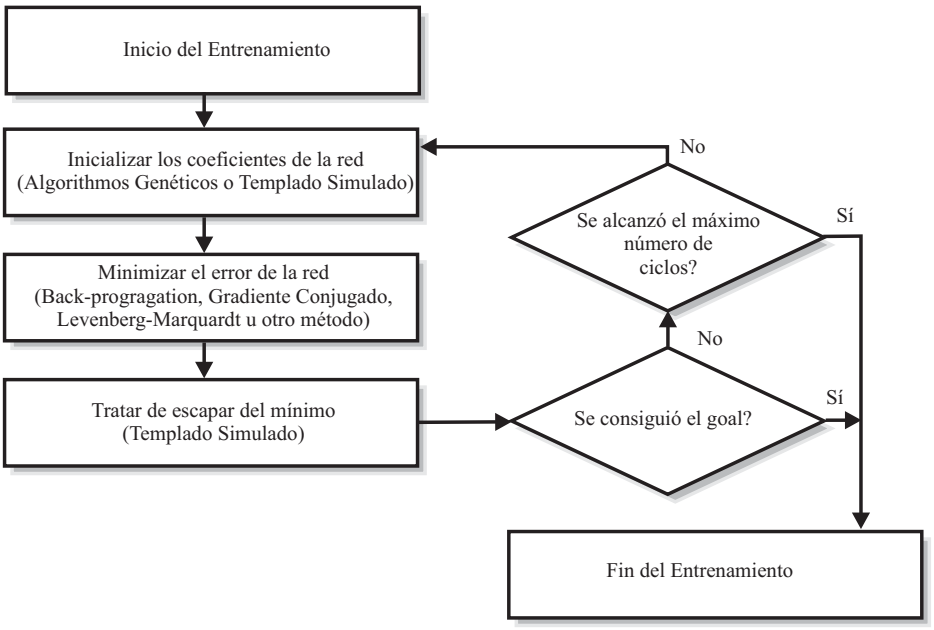
Debido a que el entrenamiento de una red es un proceso aleatorio se recomienda seguir el procedimiento descrito en el diagrama de flujo de la Figura 14, el

Figura 13. Aprendizaje de una red: (a) Normal, (b) Sobre ajuste.



cual consiste en repetir el proceso de inicialización y optimización un número fijo de veces hasta que se consigue el error requerido.

Figura 14. Diagrama de flujo del proceso de entrenamiento de una red neuronal multicapa.



3.4. Inicialización usando templado simulado

El templado simulado (en Inglés simulated annealing) es un método de optimización que imita el proceso de templado. El templado (annealing) es el proceso

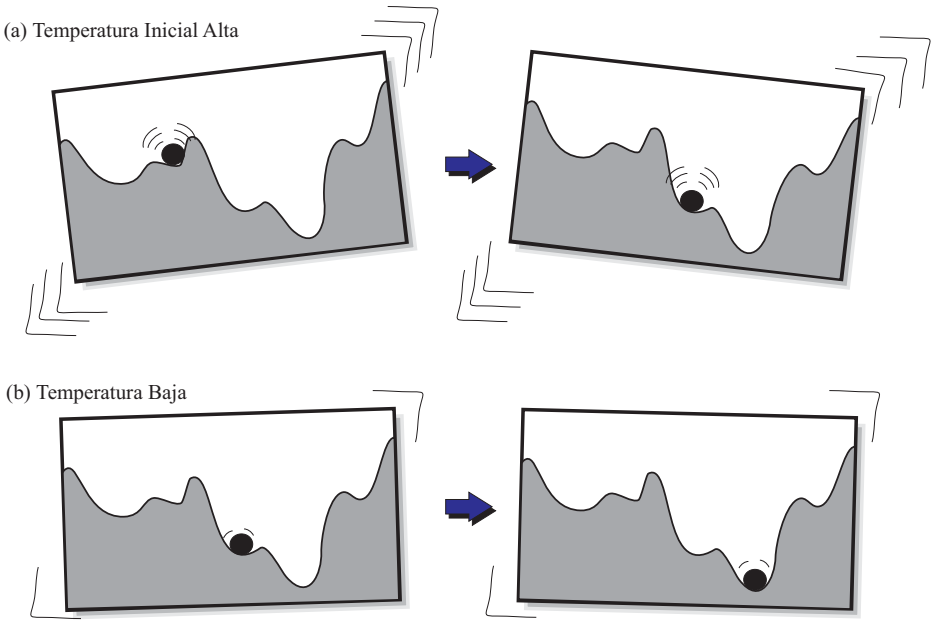
de calentar y después enfriar una sustancia en forma controlada. Las propiedades de un sólido dependen de la razón de enfriamiento después que el sólido se ha calendado más allá de su punto se fusión. El resultado deseado es una estructura cristalina fuerte. Si la sustancia se enfría rápidamente el resultado es una estructura defectuosa quebradiza.

En el templado simulado la estructura representa una solución codificada del problema, y la temperatura es usada para determinar como y cuando nuevas soluciones deben ser aceptadas.

Para implementar el templado simulado es necesario generar una gran cantidad de números aleatorios. Desafortunadamente, los generadores aleatorios proporcionados por los compiladores producen secuencias periódicas y presentan correlación serial que impiden la implementación apropiada del templado simulado. Generadores de números aleatorios más sofisticados son requeridos, (vea http://www.library.cornell.edu/nr/).

El proceso del templado simulado se entiende mejor por medio de un ejemplo. Considere un sólido que ha alcanzado su punto de fusión, y que por lo cual grandes cantidades de energía están presentes en el éste. A medida que la temperatura se reduce, la energía interna del material disminuye. Otra forma de ver al proceso de templado es como una sacudida o una perturbación. El objetivo de sacudir es encontrar un mínimo global como se muestra en la Figura 15.

Figura 15. El algoritmo de templado simulado para encontrar un mínimo global.



El proceso de templado simulado se encuentra regulado por la función Metrópolis, la cual es responsable de aceptar o rechazar nuevas soluciones. Ésta acepta todas las veces una nueva solución si la nueva solución tiene un error menor a la solución anterior. Por otro lado, la función Metrópolis puede aceptar nuevas soluciones con errores mayores a la solución anterior con la probabilidad de la ecuación (10). En general, a altas temperaturas la probabilidad de aceptar una solución es alta ya que la soluciones contienen un gran cantidad de error y la búsqueda del mínimo es del tipo global. A medida que la temperatura disminuye, la calidad de las soluciones es mejor y por lo tanto la probabilidad de aceptar soluciones es más baja.

El diagrama de la Figura 16 muestra el algoritmo típico del templado simulado para redes neuronales. La Figura 17 muestra una posible implementación de las funciones para sacudir y de Metrópolis del templado simulado usando el lenguaje C++.

$$ProbabilidaddeAceptar = \begin{cases} e^{-\frac{k*deltaError}{Temperatura}}, & deltaError > 0 \\ 1, & deltaError \leq 0 \end{cases} \tag{10}$$

3.5. Inicialización usando algoritmos genéticos

Un algoritmo genético es un técnica de optimización basada en el fenómeno de la evolución, en el cual las especies se adaptan para sobrevivir en ambientes complejos. El proceso de optimización ocurre en la estructura genética de los individuos, la cual afecta la supervivencia y reproducción de las especies.

En el caso de las redes neuronales cada solución es un individuo, el cual puede heredar sus mejores características a las nuevas generaciones. En cada generación se escogen sólo los mejores individuos (las soluciones con el menor error) para reproducción. Una vez que la nueva generación se ha creado se aplica la mutación que consiste en alterar en forma aleatoria, pero en cantidades extremadamente pequeñas, la solución. Todo este proceso se ilustra en las Figuras 18 y 19. El objetivo del algoritmo es obtener mejores individuos (soluciones con menor error) a medida que nuevas generaciones emergen.

Para generar la población inicial del algoritmo genético se puede usar el código mostrado en la Figura 20. Un rápido análisis del código revela que cada gene de cada individuo se genera en forma aleatoria. Para implementar la mutación del algoritmo genético se genera una número aleatorio por cada gene que tiene el individuo, si este número es menor que la probabilidad de mutación el gene se muta, esto es si es cero se fija en uno, y si es uno se fija en cero como se ilustra en el código de la Figura 20.

3.6. Tipos de entrenamiento

Existen dos tipos de entrenamiento para redes neuronales: supervisado y sin supervisión, ver la Figura 21. En el entrenamiento supervisado, a la red se le

Figura 16. Diagrama de flujo del algoritmo de templado simulado.

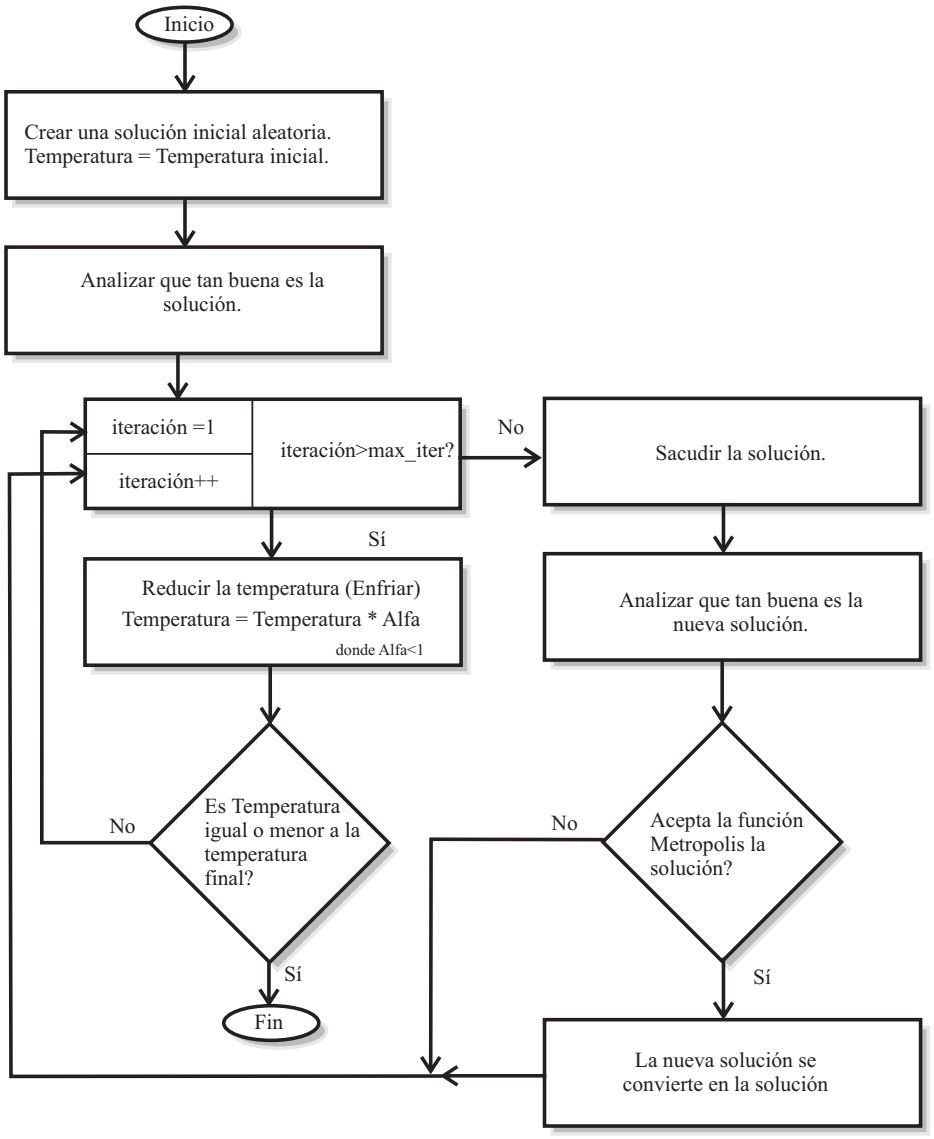


Figura 17. Códigos C++ de las funciones sacudir y Metrópolis.


Templado.cpp

```
void Templado::Sacudir(int numeroPesos, double *pesosEntrada, double *pesosSalida)
{
  for(int i = 0; i<numeroPesos; i++)
  {
    pesosSalida[i] = pesosEntrada[i] + random.generarValorEntre(-20.0, 20.0);

    //Asegurar que los pesos de salida tomen valores apropiados
    if (pesosSalida[i]<-20)
    {
      pesosSalida[i] = -20.0;
    }
    else if (pesosSalida[i]>20)
    {
      pesosSalida[i] = 20.0;
    }
  }
}

bool Templado::AceptaMetropolisLaSolucion(double error, double errorNuevo, double temperature)
{
  bool aceptar = false;
  double deltaError = errorNuevo-error;

  if (deltaError<0)
  {
    aceptar = true;  //Aceptar la solucion si el error es menor
  }
  else
  {
    //Aceptar la solucion con cierta probabilidad
    if (random.generarValorEntre(0, 1) < exp(-deltaError/temperature))
      aceptar = true;
    else
      aceptar = false;
  }
  return aceptar;
}
```

Figura 18. Operación básica del algoritmo genético.

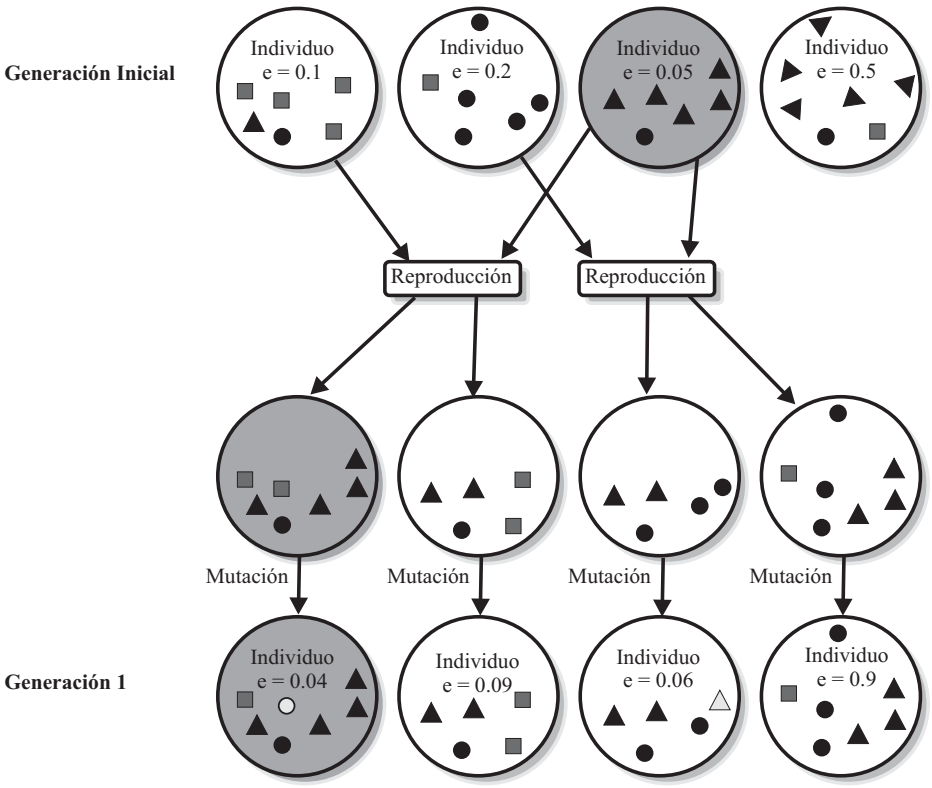


Figura 19. Diagrama de flujo del algoritmo genético.

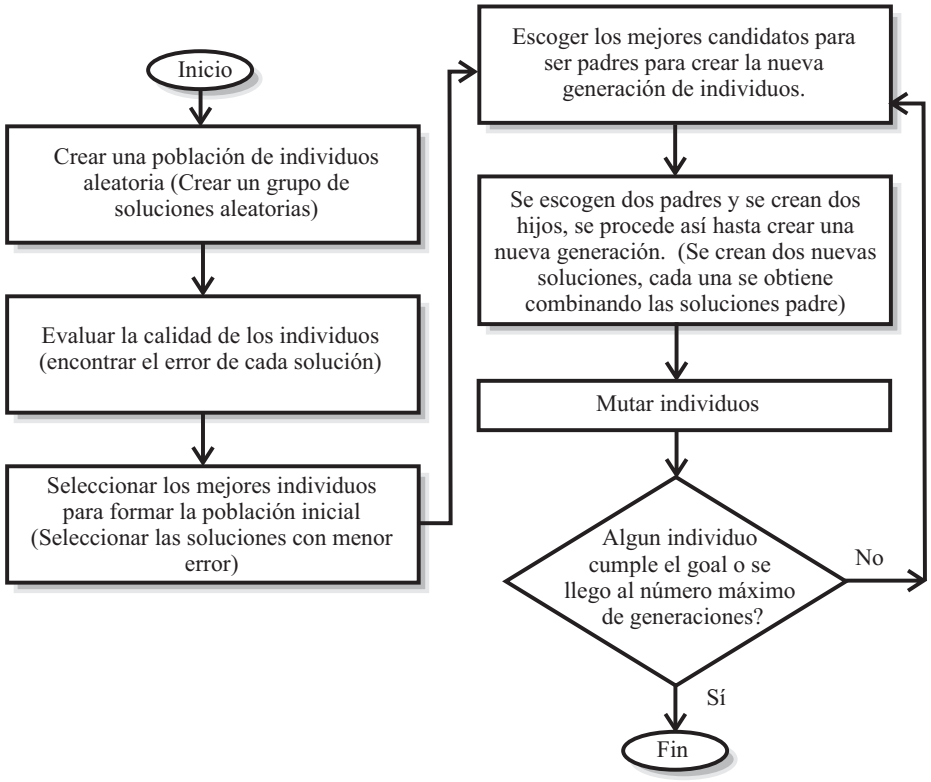


Figura 20. Implementación de C++ de las funciones para inicializar y mutar un individuo.

Genetic.cpp

```
void Genetic::mutar(Individuo& individuo)
{
  int indice = 0;
  int numeroGenes = individuo.numeroGenes;
  for(int i = 0; i<numeroGenes; i++)
  {
    //Se debe mutar el gene?
    if (random.generarNumeroEntre(0, 1) < probabilidadMutar)
    {
      indice = random.generarNumeroEntre(0, numeroGenes);
      if (individuo.gene[indice]==0)
      {
        individuo.gene[indice] = 1;
      }
      else
      {
        individuo.gene[indice] = 0;
      }
    }
  }
}
```

```
void Genetic::inicializar(Individuo& individuo)
{
  for(int i = 0; i<individuo.numeroGenes; i++)
  {
    if (random.generarNumeroEntre(0, 1)<0.5)
    {
      individuo.gene[i] = 0;
    }
    else
    {
      individuo.gene[i] = 1;
    }
  }
}
```

proporciona la salida deseada y la red se ajusta para producir esta salida. En el entrenamiento sin supervisión, la red descubre patrones por sí sola para clasificar objetos.

Para poder ajustar una red que usa entrenamiento supervisado, usualmente se usa el error medio cuadrático entre la salida deseada, t y la salida producida por la red z en una red, por ejemplo para una red de n neuronas de salida, el error medio cuadrático se calcula por medio de

$$MSE = \sum_{i=1}^{n} (t_i - z_i)^2. \tag{11}$$

3.7. Entrenamiento con back-propagation

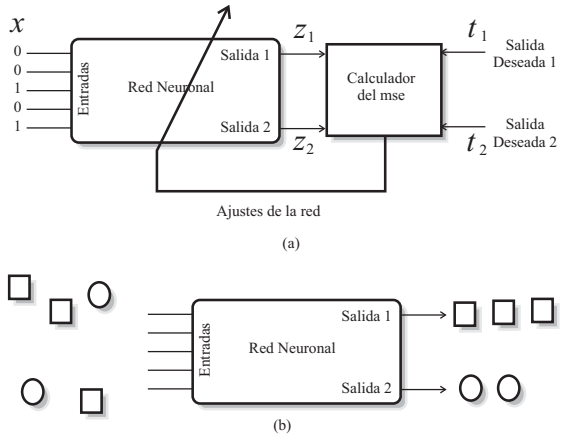
Existen distintos algoritmos para entrenar una red: Back-Propagation, Gradiente Conjugado, Levenberg-Marquardt, etc. El algoritmo más simple es el de Back-Propagation.

El método de Back-Propagation calcula la salida de la red y luego proporciona un ajuste a los pesos de la red usando un parámetro conocido como razón de aprendizaje, el cual toma valores entre 0 y 1. Valores cercanos a 1 producen una convergencia más rápida, sin embargo, estos valores pueden afectar la calidad del entrenamiento.

El algoritmo de Back-propagation tiene cinco pasos, los cuales son descritos a continuación:

1. Encontrar la salida de cada neurona de la red.

Figura 21. Tipos de entrenamiento: (a) Supervisado, (b) Sin supervisión.



2. Encontrar el error mínimo cuadrático de la red usando equation (11).
3. Encontrar el delta del error para cada neurona de salida,

$$\delta_i = (t_i - z_i)z_i(1 - z_i), \qquad i = 1, 2, 3, \ldots, m \tag{12}$$

4. Encontrar el delta del error para cada neurona interna,

$$\delta_i = \delta_j w_{ij} x_i (1 - x_i). \tag{13}$$

   Un error típico al usar el algoritmo de back-propagation es usar los índices equivocados cuando se calculan los delta del error; recuerde que el primer sub-índice del coeficiente indica el número de la neurona destino del coeficiente, mientras que el segundo sub-índice del coeficiente denota la neurona origen del coeficiente como se muestra en la Figura 22.
5. Ajustar los coeficientes de la red usando los delta del error previamente calculados

$$w_{ji,nuevo} = w_{ji,previo} + \rho \delta_j x_i, \tag{14}$$

   donde ρ es la razón de aprendizaje.

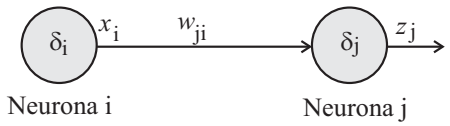
Problema 8. En la red neuronal de la Figura 23 encuentre los valores de x_1, x_2 y z_1; cuando

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0.01 \\ 0.97 \end{bmatrix}$$

$$H = \begin{bmatrix} 0.93 & 0.47 & 1.01 \\ -0.98 & 2.02 & 0.92 \end{bmatrix}$$

$$W = \begin{bmatrix} 1.51 & -0.97 & 1.07 \end{bmatrix}$$

Figura 22. Asignación de los sub-índices de los coeficientes de una red neuronal



Solución 8.

$$v_1 = (0.01)(0.93) + (0.97)(0.47) + (1.01)(1) = 1.47520$$

$$x_1 = 0.813842$$

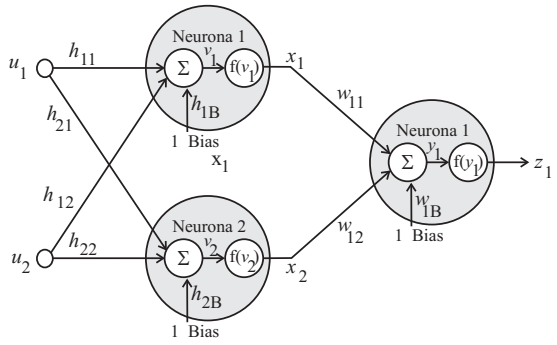
$$v_2 = (0.01)(\text{-}0.98) + (0.97)(2.02) + (0.92)(1) = 2.869600$$

$$x_2 = 0.946321$$

$$y_1 = (1.51)(0.813842) + (\text{-}0.97)(0.946321) + (1.07)(1) = 1.380970$$

$$z_1 = 0.799142$$

Figura 23. Entrenamiento de una red neuronal por medio de back-propagation



Problema 9. En la red de la Figura 23 encuentre el valor del error medio cuadrático (MSE). Suponga que la salida deseada de la red es de 1.05.

Solución 9.

$$MSE = (1.05 - 0.799142)^2 = 0.062930$$

Problema 10. En la red de la Figura 23 encuentre el valor del delta del error de la neurona de salida.

Solución 10.

$$\delta_{1,salida} = (1.05 - 0.799142)(0.799142)(1.0 - 0.799142) = 0.040266$$

Problema 11. En la red de la Figura 23 encuentre el valor del delta del error para cada neurona en la capa escondida.

Solución 11.

$$\delta_{1,escondida} = (0.040266)(1.51)(0.813842)(1.0 - 0.813842) = 0.009212$$

$$\delta_{2,escondida} = (0.040266)(\text{-}0.97)(0.946321)(1.0 - 0.946321) = \text{-}0.001984$$

Problema 12. Encuentre los nuevos pesos H y W de la red de la Figura 23 usando los deltas del error previamente calculados; use una razón de aprendizaje de 0.2.

Solución 12.

$$h_{11} = (0.93) + (0.2) * (0.009212) * (0.01) = 0.930018$$

$$h_{12} = (\text{-}0.98) + (0.2) * (\text{-}0.001984) * (0.01) = \text{-}0.980004$$

$$h_{1B} = (1.01) + (0.2) * (0.009212) * (1) = 1.011842$$

$$h_{21} = (0.47) + (0.2) * (0.009212) * (0.97) = 0.471787$$

$$h_{22} = (2.02) + (0.2) * (\text{-}0.001984) * (0.97) = 2.019615$$

$$h_{2B} = (0.92) + (0.2) * (\text{-}0.001984) * (1) = 0.915635$$

$$w_{11} = (1.51) + (0.2)(0.040266)(0.813842) = 1.516554$$

$$w_{12} = (\text{-}0.97) + (0.2)(0.040266)(0.946321) = \text{-}0.962379$$

$$w_{1B} = (1.07) + (0.2)(0.040266)(1) = 1.078053$$

Problema 13. En la red de la Figura 23 encuentre el valor del MSE con los nuevos coeficientes calculados en el problema anterior.

Solución 13.

$$v_1 = (0.01) * (0.930018) + (0.97) * (0.471787) + (1.011842) * (1) = 1.478776$$

$$x_1 = 0.814383$$

$$v_2 = (0.01) * (\text{-}0.980004) + (0.97) * (2.019615) + (0.915635) * (1) = 2.864862$$

$$x_2 = 0.946080$$

$$y_1 = (1.516554)*(0.814383)+(\text{-}0.962379)*(0.946080)+(1.078053)*(1) = 1.402621$$

$$z_1 = 0.802595$$

$$MSE = (1.05 - 0.802595)^2 = 0.061209$$

3.8. Entrenamiento usando regresión

Dada una tabla con puntos (x, y) obtenidos de un modelo matemático o un proceso físico, el procedimiento de regresión permite estimar cualquier valor de y dado un valor de x o un valor de x dado un valor de y.

La regresión lineal establece que la variable dependiente y es una función lineal de la variable independiente x. Por otro lado, la regresión no-lineal indica que y es una función no-lineal de la variable dependiente x. Si el escalar x, es reemplazado por el vector $\mathbf{x}$. Entonces se dice que y es una combinación lineal o no-lineal de los valores de x en el vector $\mathbf{x}$.

Aun cuando una red neuronal tiene una relación de entrada-salida no-lineal, el proceso de regresión lineal puede ser usado para estimar los coeficientes de ésta. Este procedimiento consiste en plantear un sistema de ecuaciones para cada neurona de salida. Usualmente, el sistema de ecuaciones resultante se encuentra mal condicionado y el método de eliminación de Gauss no funciona bien en estos casos.

Problema 14. Suponga que se conoce el valor de z_1, encuentre una expresión para el valor de y1 en la Figura 2, cuando la función de activación es (a) $z = logsig(y)$, (b) $z = tanh(ay)$.

Solución 14. (a)

$$z = \frac{1}{1 + e^{-y}}$$

$$1 + e^{-y} = \frac{1}{z}$$

$$e^{-y} = \frac{1}{z} - 1$$

$$y = -\ln\left(\frac{1}{z} - 1\right)$$

(b)

$$tanh(ax) = y$$

$$ax = tanh^{-1}(y)$$

$$ax = \frac{1}{2}\ln\left(\frac{1+y}{1-y}\right)$$

$$x = \frac{1}{2a}\ln\left(\frac{1+y}{1-y}\right)$$

El mejor método para resolver un sistema de ecuaciones mal condicionado está basado en la descomposición en valores singulares (Singular Value Decomposition.) El álgebra lineal básica indica que una matriz $\mathbf{A}$ cuyo número de renglones M es mayor que o igual o su número de columnas N, puede escribirse como el producto de una matriz ortogonal $\mathbf{U}$ de tamaño $M \times N$, una matriz diagonal $\mathbf{S}$

con elementos cero o positivos, y la transpuesta de una matriz ortogonal $\mathbf{V}$ de tamaño $N \times N$ como se muestra.

$$\mathbf{A} = \mathbf{USV}^T$$

$$\mathbf{U}^T\mathbf{U} = [1] \rightarrow \mathbf{U}^{-1} = \mathbf{U}^T$$

$$\mathbf{V}^T\mathbf{V} = [1] \rightarrow \mathbf{V}^{-1} = \mathbf{V}^T$$

$$\mathbf{S} = \begin{bmatrix} s_1 & 0 & 0 & \dots 0 \\ 0 & s_2 & 0 & \dots 0 \\ & \dots & & \\ 0 & 0 & 0 & \dots s_N \end{bmatrix}$$

Para resolver un sistema de ecuaciones lineales usando la descomposición en valores singulares se debe:

1. Encontrar la matriz A.
2. Descomponer la matriz A en las matrices U, S y V.
3. Usando los valores de U, S y V encontrar la solución del sistema de ecuaciones.

Problema 15. Suponga que la red de la Figura 24 implementa las compuertas lógicas AND y OR, con los cuatro casos de entrenamiento mostrados.

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{Z} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

Usando el valor de $\mathbf{Z}$ dado, encuentre el valor de y para los cuatro casos de entrenamiento, escriba sus resultados en forma de matriz. Suponga que la red usa la función de activación $z = logsig(y)$.

Solución 15.

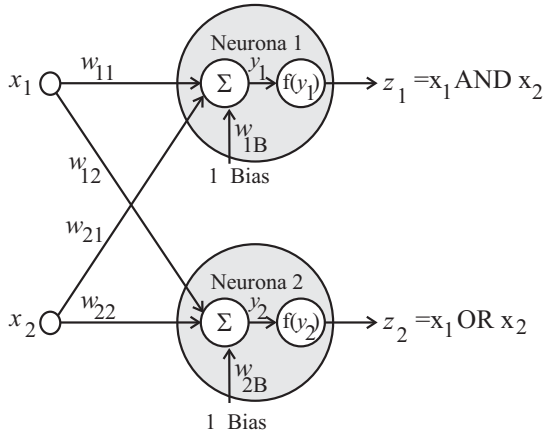
$$\mathbf{Y} = \begin{bmatrix} logsig^{-1}(0) & logsig^{-1}(0) & logsig^{-1}(0) & logsig^{-1}(1) \\ logsig^{-1}(0) & logsig^{-1}(1) & logsig^{-1}(1) & logsig^{-1}(1) \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} -20 & -20 & -20 & 20 \\ -20 & 20 & 20 & 20 \end{bmatrix}$$

Problema 16. Usando la definición mostrada encuentre el valor de la matriz A.

$$\mathbf{A} = \left[\mathbf{X}^T \mathbf{1}\right]$$

Figura 24. Red neuronal implementando las compuertas lógicas AND y OR.



Solución 16.

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Problema 17. Usando Matlab realize la descomposición en valores singulares de la matriz $\mathbf{A}$ para obtener las matrices $\mathbf{U}$, $\mathbf{S}$ y $\mathbf{V}$.

Solución 17. `[U,S,V] = svd(A,0)`

```
U =
   -0.3035    0.0000    0.8111
   -0.4835    0.7071    0.1273
   -0.4835   -0.7071    0.1273
   -0.6635    0.0000   -0.5565

S =
    2.5243         0         0
         0    1.0000         0
         0         0    0.7923

V =
   -0.4544   -0.7071   -0.5418
   -0.4544    0.7071   -0.5418
   -0.7662    0.0000    0.6426
```

Problema 18. Puede mostrarse fácilmente que:

$$\mathbf{A}\mathbf{W}^T = \mathbf{Y}^T$$

$$\mathbf{A}\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{1B} & w_{2B} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{21} \\ y_{12} & y_{22} \\ y_{13} & y_{23} \\ y_{14} & y_{24} \end{bmatrix}$$

lo cual puede escribirse como dos sistemas de ecuaciones como

$$\mathbf{A}\begin{bmatrix} w_{11} \\ w_{12} \\ w_{1B} \end{bmatrix} = \begin{bmatrix} y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \end{bmatrix}$$

$$\mathbf{A}\begin{bmatrix} w_{21} \\ w_{22} \\ w_{2B} \end{bmatrix} = \begin{bmatrix} y_{21} \\ y_{22} \\ y_{23} \\ y_{24} \end{bmatrix}$$

Usando los valores $\mathbf{A}$ y $\mathbf{Y}$ previamente calculados, escriba los dos sistemas de ecuaciones para la red neuronal de la Figura 24.

Solución 18.

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} w_{11} \\ w_{12} \\ w_{1B} \end{bmatrix} = \begin{bmatrix} -20 \\ -20 \\ -20 \\ 20 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} w_{21} \\ w_{22} \\ w_{2B} \end{bmatrix} = \begin{bmatrix} -20 \\ 20 \\ 20 \\ 20 \end{bmatrix}$$

Problema 19. Demuestre que los coeficientes de la primera y de la segunda neurona pueden ser calculados por medio de la descomposición en valores singulares como:

$$\begin{bmatrix} w_{11} \\ w_{12} \\ w_{1B} \end{bmatrix} = \mathbf{V}\mathbf{S}^{-1}\mathbf{U}^T\begin{bmatrix} y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \end{bmatrix}$$

$$\begin{bmatrix} w_{21} \\ w_{22} \\ w_{2B} \end{bmatrix} = \mathbf{V}\mathbf{S}^{-1}\mathbf{U}^T\begin{bmatrix} y_{21} \\ y_{22} \\ y_{23} \\ y_{24} \end{bmatrix}$$

Solución 19.

$$\mathbf{A}w = y$$

$$\mathbf{USV}^T w = y$$

$$\mathbf{V}^T w = \mathbf{SU}^T y$$

$$w = \mathbf{VS}^{-1}\mathbf{U}^T y$$

Problema 20. Por medio de MATLAB encuentre los valores de los coeficientes de la primera y segunda neurona usando los valores de V, S, U y Y.

Solución 20.
```
B = [-20; -20; -20; 20];
W1= V*inv(S)*U'*B

W1 =
   20.0000
   20.0000
  -30.0000

W2= V*inv(S)*U'*B

W2 =

   20.0000
   20.0000
  -10.0000
```

Antes de terminar el tema de la regresión es importante notar que el proceso de regresión puede ser usado para entrenar redes neuronales de una capa. En el caso de redes de dos o más capas, el proceso de regresión puede ser usado para estimar los coeficientes en la capa de salida solamente. Los coeficientes en las capas de entrada o capas intermedias deben estimarse usando otros procedimientos de entrenamiento de redes neuronales.

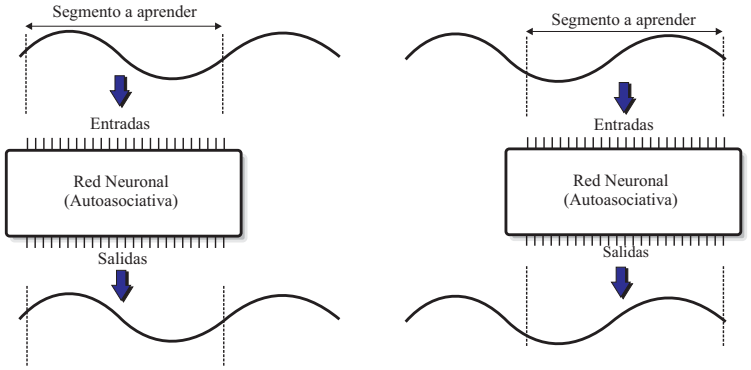
4. Reducción de ruido

Una característica muy importante de una red neuronal es su inmunidad al ruido. Una red neuronal es capaz de aprender de patrones puros o patrones con ruido y una vez que el entrenamiento ha concluido la red es capaz de reconocer estos patrones o bien de eliminar el ruido de estos.

4.1. Redes auto-asociativas

En esta sección se entrenará una red neuronal aplicando una onda senoidal pura con distintas fases como se ilustra en la Figura 25. La salida deseada de la red es la misma señal senoidal aplicada a la entrada de la red. Este tipo de redes se conocen como redes auto-asociativas. Una vez terminado el entrenamiento se le aplicará a la red una onda senoidal contaminada con ruido y se analizará su salida. Note que los resultados de esta sección son para ilustrar la operación de una red neuronal, y que dependiendo de la aplicación, otras técnicas de procesamiento digital de señales pueden ser más apropiadas para reducción de ruido.

Figura 25. Configuración típica de una red auto-asociativa.



4.2. Reducción de ruido en señales periódicas

Problema 21. Cree un archivo del tipo m de MATLAB para crear un conjunto de datos para entrenar una red neuronal de 32 salidas y 32 entradas. El conjunto de datos debe contener 64 casos de entrenamiento, cada uno de estos case debe contener una onda senoidal pura, la fase de la onda senoidal debe ser distinta para cada caso de entrenamiento y debe encontrarse en el rango de 2π. Normalice el conjunto de datos de entrenamiento para ser usado en una red neuronal con función de activación $z = logsig(y)$.

Solución 21.

```
clear;
CASOS_ENTRENAMIENTO = 64;
ENTRADAS = 32;
RANGO = 2.0*pi;
```

```
IN_DELTA = RANGO/ENTRADAS;
SET_DELTA = RANGO/CASOS_ENTRENAMIENTO;
P = zeros(ENTRADAS, CASOS_ENTRENAMIENTO);
PN = zeros(ENTRADAS,
CASOS_ENTRENAMIENTO);
for i=1 : CASOS_ENTRENAMIENTO
    fase = (i-1)*SET_DELTA;
    for j=1 : ENTRADAS;
        P(j, i) = sin(fase+(j-1)*IN_DELTA);
    end
end
```

Antes de normalizar el conjunto de datos de entrenamiento, se verificará que el conjunto tiene las dimensiones apropiadas, 32 renglones (una por cada entrada) y 64 columnas (una columna por cada caso de entrenamiento).

```
size(P)
```

Una vez que el conjunto de datos de entrenamiento ha sido creado, se procede a su normalización y, posteriormente, todos los casos de entrenamiento son graficados.

```
PN = (P + 1) .* (0.8/2)+0.1;
plot(PN);
```

Finalmente, se comprueba el rango de valores del conjunto de datos de entrenamiento normalizados PN, el cual debe ser de 0.1 a 0.9 para todos los casos de entrenamiento.

```
minmax(PN)
```

Problema 22. Usando MATLAB cree una red neuronal auto asociativa para aprender la señal senoidal del conjunto de datos de entrenamiento previamente creados. Debido a que la red tiene un número considerable de salidas el entrenamiento traingdx es más adecuado que el trainlm.

Solución 22. Primeramente se debe crear la matriz de rangos de entrada, la cual es una matriz de dos columnas. El primer elemento del primer renglón indica el valor mínimo de la primer entrada de la red, el segundo elemento del primer renglón indica el valor máximo de la primer entrada de la red.

```
RI = zeros(ENTRADAS, 2);
for i=1: ENTRADAS
    RI(i, 1) = 0.1;
    RI(i, 2) = 0.9;
end
```

Finalmente, se procede a crear la red y entrenarla hasta conseguir el menor error. A fin de evitar el sobre ajuste, se debe usar el menor número de neuronas.

```
net = newff(RI, [5, ENTRADAS], {'logsig', 'logsig'}, 'traingdx');
net.trainParam.goal = 0.00000015;
net.trainParam.epochs = 10000;
net = train(net, PN, PN);
```

Problema 23. Usando MATLAB y la red previamente entrenada, encuentre la salida de la red cuando a la entrada se aplica una onda senoidal pura.

Solución 23. Primeramente, se procede a calcular una onda senoidal pura compatible con el número de entradas de la red neuronal.

```
A = [1 : ENTRADAS];
for i=1 : ENTRADAS
    A(i) = -pi+ (i+1)*IN_DELTA;
end X = [1 : ENTRADAS];
X = sin(A);
```

Posteriormente, se procede a normalizar la onda senoidal para que sea compatible con la función de activación neuronal usada en la red.

```
XN= ( (X+1) .* (0.8/2.0)+0.1)';
```

Una vez que la señal de entrada ha sido normalizada, se realiza la simulación de la red y se grafica la salida de la red, con la salida deseada.

```
XN= ( (X+1) .* (0.8/2.0)+0.1)';
YN=sim(net, XN);
Y = (YN-0.1)./(0.8/2.0)-1;
plot(A, Y, 'r');
hold on; X = sin(A);
plot(A, X, 'g');
hold off;
```

Dependiendo del error conseguido durante el entrenamiento de la red, la señal de salida de la red debe ser similar a una onda senoidal pura.

Finalmente, se aplicará a la entrada de la red una onda senoidal contaminada con 10 % de ruido y se analizará la salida de la misma.

```
ruido = 2*rand(1, ENTRADAS)-1; %Pico a pico = 2
X = 0.9*sin(A) + 0.1*ruido;
plot(A, X, 'g');
hold on; XN= ( (X+1).* (0.8/2.0)+0.1)';
YN=sim(net, XN);
Y = (YN-0.1)./ (0.8/2.0)-1;
plot(A, Y, 'r');
hold off;
```

En el problema anterior se usaron solamente 64 casos de entrenamiento. Se recomienda al lector repetir el problema anterior usando 350 o mas casos de entrenamiento.

4.3. Reducción de ruido en señales no-periódicas

Una de las principales ventajas de trabajar con señales periódicas es que su espectro se encuentra limitado a bandas o frecuencias específicas. Por el contrario las señales no periódicas pueden exhibir espectros complejos haciendo difícil la eliminación de ruido por medio de filtros. En estos casos una red neuronal es una buena opción, sin embargo, el conjunto de datos de entrenamiento debe contener más casos de entrenamiento que en el caso de señales periódicas.

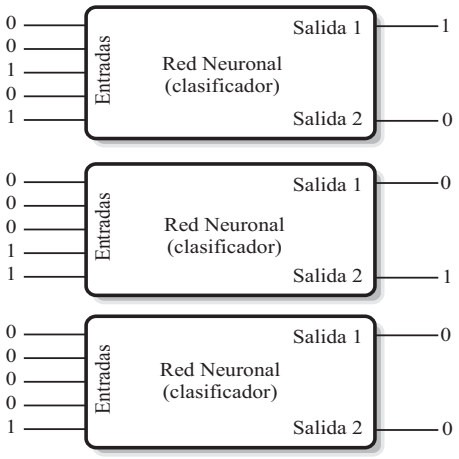
5. Usando una red neuronal como clasificador

Clasificación es el proceso de identificación de un objeto dentro de un conjunto posible de resultados. Una red puede ser entrenada para identificar y separar distintos tipos de objetos. Estos objetos pueden ser: números, imágenes, sonidos, señales, etc.

5.1. Introducción a los clasificadores

Una red neuronal en configuración de clasificador tiene un número de salidas igual al número de categorías de clasificación. La Figura 26 muestra un clasificador de los múltiplos de 2 y 3. Cuando el número en la entrada de la red es un múltiplo de 2, la salida 1 se activa completamente, mientras que la salida 2 se encuentre prácticamente sin activación. Cuando se aplica un múltiplo de 3 la salida 2 se activa. Cuando se aplica un número que no es múltiplo de 2 ni de 3, ambas salidas se encuentran inactivas.

Figura 26. Clasificador de múltiplos de 2 o de 3.



5.2. La matriz de confusión

Una matriz de confusión se utiliza en clasificación para indicar la calidad de la solución. Si la red tiene un número m de salidas, esta matriz tiene m renglones y m+1 columnas. Esta matriz indica cuantos elementos fueron clasificados correctamente. Cuando la red no comete errores la matriz de confusión es diagonal como se muestra en la Figura 27.a.

La matriz de confusión debe tener idealmente un sólo elemento distinto de cero en cada renglón. En caso de que la matriz de confusión contenga elementos distintos de cero fuera de la diagonal principal, esto indica que la red confundió un objeto de un tipo con otro tipo de objeto. Por otro lado, la última columna de esta matriz, (Rechazo) indica los elementos que no pudieron clasificarse, por lo que debe contener sólo ceros. Vea la Figura 27.b.

Figura 27. Matriz de confusión (a) Sin errores, (b) Con errores.

(a)

	salida 1	salida 2	Rechazo
salida 1	6	0	0
salida 2	0	11	0

(b)

	salida 1	salida 2	Rechazo
salida 1	4	1*	1*
salida 2	0	8	3*

5.3. Clasificación numérica

Problema 24. Diseñe una red neuronal en MATLAB para clasificar los números que son múltiplos de 2 o múltiplos de 3 en el rango de 1 a 16.

Solución 24. Primeramente se procede a crear el conjunto de datos de entrenamiento, el cual debe tener 16 casos de entrenamiento. La primer salida debe activarse cuando el número a la entrada de la red es múltiplo de 2. La segunda salida de la red debe activarse con los múltiplos de 3.

```
P=[ 0, 0, 0, 1; %1
    0, 0, 1, 0; %2
    0, 0, 1, 1; %3
    0, 1, 0, 0; %4
    0, 1, 0, 1; %5
    0, 1, 1, 0; %6
    0, 1, 1, 1; %7
    1, 0, 0, 0; %8
    1, 0, 0, 1; %9
    1, 0, 1, 0; %10
    1, 0, 1, 1; %11
```

```
    1, 1, 0, 0; %12
    1, 1, 0, 1; %13
    1, 1, 1, 0; %14
    1, 1, 1, 1; %15
    0, 0, 0, 0]'; %16
T=[ 0, 0; %1
    1, 0; %2
    0, 1; %3
    1, 0; %4
    0, 0; %5
    1, 1; %6
    0, 0; %7
    1, 0; %8
    0, 3; %9
    1, 0; %10
    0, 0; %11
    1, 1; %12
    0, 0; %13
    1, 0; %14
    0, 1; %15
    0, 0]'; %16
```

Se procede ahora a crear la matriz de rangos de entrada, crear la red e iniciar el entrenamiento.

```
rangos=[0, 1; 0, 1; 0, 1; 0, 1];
net = newff(rangos, [4, 2], {'logsig', 'logsig'}, 'traingdx');
net.trainParam.epochs=1000;
net.trainParam.goal=0.0001;
net = train(net, P, T);
```

Finalmente, para verificar la operación del clasificador se usa el comando `sim` de MATLAB.

```
X=sim(net, P)

X =

  Columns 1 through 4

    0.0001    1.0000    0.0000    0.9953
    0.0008    0.0000    0.9977    0.0050

  Columns 5 through 8

    0.0025    1.0000    0.0002    0.9991
```

```
    0.0000    0.9955    0.0005    0.0000

  Columns 9 through 12

    0.0000    1.0000    0.0000    1.0000
    1.0000    0.0012    0.0013    1.0000

  Columns 13 through 16

    0.0004    1.0000    0.0001    0.0034
    0.0001    0.0006    0.9999    0.0000
```

Un análisis del vector **X** revela que el clasificador opera correctamente para cualquier número entre 1 y 16.

6. Redes neuronales complejas

Una red neuronal compleja es aquella en la que las entradas, las salidas y los coeficientes de la red son números complejos como se muestra en la Figura 28 con

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix} \rightarrow \begin{array}{l} x_1 = x_{r1} + jxi1 \\ x_2 = x_{r2} + jxi2 \\ x_3 = x_{r3} + jxi3 \\ \dots \\ x_n = x_{rn} + jxin \end{array}$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1n} & w_{1B} \end{bmatrix} \rightarrow \begin{array}{l} w_{11} = w_{r11} + jwi11 \\ w_{12} = w_{r12} + jwi12 \\ w_{13} = w_{r13} + jwi13 \\ \dots \\ w_{1n} = w_{r1n} + jwi1n \\ w_{1B} = w_{r1b} + jwi1b \end{array}$$

donde subíndices r e i son usados para representar la parte real e imaginaria de los números complejos.

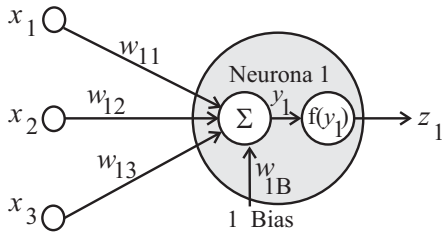
6.1. Introducción a las redes neuronales complejas

Para calcular el valor de y en una red neuronal compleja se usa el álgebra de los números complejos como se muestra.

$$y_1 = y_{r1} + jy_{i1}$$

$$y_{r1} = \sum_{k=1}^{n} x_{rk} w_{rk} - \sum_{k=1}^{n} x_{rik} w_{ik}$$

Figura 28. Red neuronal compleja.

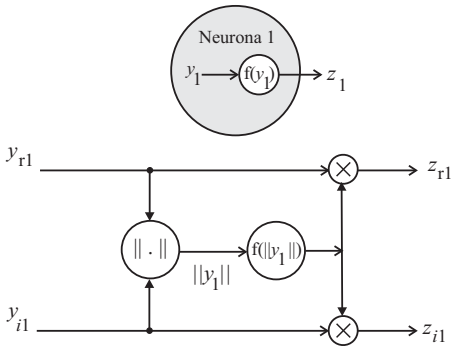


$$y_{i1} = \sum_{k=1}^{n} x_{rk} w_{rk} + \sum_{k=1}^{n} x_{rik} w_{ik}$$

$$||y_1|| = \sqrt{y_{r1}^2 + y_{i1}^2}$$

El proceso de activación neuronal en las redes complejas es diferente al de las redes neuronales reales; la neurona recibe y produce una variable del tipo compleja. En la Figura 29 muestra una forma apropiada de realizar la activación de la neurona usando números complejos. Obsérvese que la magnitud de y determina el nivel de activación de la neurona. Note además que el nivel de activación afecta la salida real e imaginaria de la neurona.

Figura 29. Activación de una neurona con entrada compleja.



6.2. La trasformada de Fourier

Una red neuronal puede diseñarse para trabajar con señales en el dominio del tiempo o con señales en el dominio de la frecuencia. La transformada de Fourier

permite cambiar una señal de un dominio a otro. Dependiendo de la aplicación puede ser más conveniente entrenar la red con señales en el dominio del tiempo o de la frecuencia.

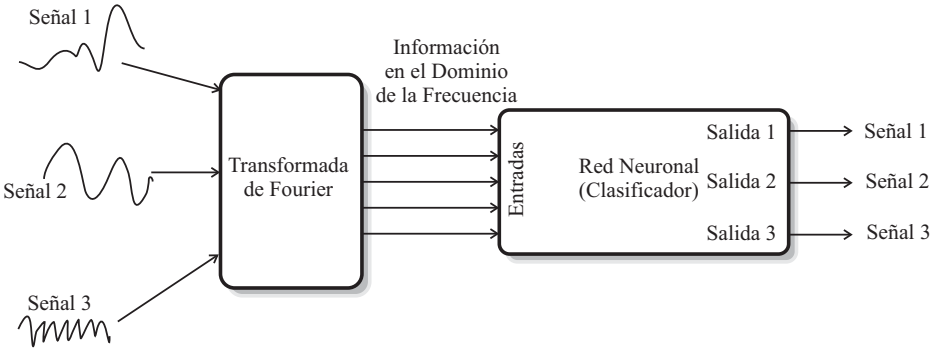
Si una señal en el dominio del tiempo es transformada al dominio de la frecuencia, la nueva señal contendrá dos componentes, parte real y parte imaginaria. Para procesar y analizar este tipo de señales se pueden usar redes neuronales complejas.

6.3. Usando redes neuronales con números complejos

Aunque las redes neuronales complejas son usadas para el proceso y análisis de señales complejas, también es posible usar este tipo de redes en donde existen dos variables que se encuentran relacionadas de algún modo.

En algunos casos la señal en el dominio del tiempo no exhibe en forma apropiada las características que deseamos enseñar a una red neuronal. En estos casos, es posible usar la Trasformada de Fourier para convertir la señal al dominio de la frecuencia antes de realizar el entrenamiento como se muestra en la Figura 30.

Figura 30. Aplicación de la transformada de Fourier en redes neuronales.



7. Ejercicios propuestos

Problema 25. Encuentre la derivada $f'(y)$ de la función de activación $f(y) = tanh(ay)$.

Problema 26. Encuentre la salida, z_1, de la red neuronal de la Figura 2 cuando la entrada es x y los coeficientes de la red son W, suponga que $f(y) = tanh(1.5y)$.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -5 \\ 19 \\ -10 \end{bmatrix} \tag{15}$$

$$W = \begin{bmatrix} w_{11}\ w_{12}\ w_{13}\ w_{1B} \end{bmatrix} = \begin{bmatrix} 2\ 14\ -5\ 1 \end{bmatrix} \tag{16}$$

Problema 27. Encuentre la salida de la red con capa escondida de la Figura 6 cuando la entrada está definida por u, y los coeficientes de la red por H y W como se muestra. Use la función de activación $f(y) = tanh(1.5y)$ para ambas capas.

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 1.1 \\ \text{-}1.5 \end{bmatrix}$$

$$H = \begin{bmatrix} \text{-}1.8 & 2.6 & 0.6 \\ 5.5 & \text{-}0.2 & 3.0 \end{bmatrix}$$

$$W = \begin{bmatrix} 1.0 & 1.0 & \text{-}0.2 \end{bmatrix}$$

Problema 28. Cree una red neuronal de una sola neurona para simular la compuerta lógica AND usando MATLAB con los coeficientes mostrados en la Figura 7.

Problema 29. Diseñe una red neuronal en MATLAB para que ésta aprenda la compuerta lógica AND usando el método del gradiente conjugado (primero, diseñe un conjunto de datos de entrenamiento apropiado.)

Problema 30. Diseñe una red del tipo mapeador con una entrada y una salida para aprender las función $F(x) = sinc(x)$ usando MATLAB. Las neuronas de la red deben usar la función de activación $z = logsig(y)$ y debe operar para valores de x en el rango de -10 a 10.

Problema 31. Usando Microsoft Excel diseñe una hoja de cálculo para resolver el Problema 8.

Problema 32. Usando Microsoft Excel diseñe una hoja de cálculo para realizar dos iteraciones del método de backpropagation con los datos del Problema 9 mostrando el mse después de cada iteración.

Problema 33. Diseñe una red neuronal en MATLAB para clasificar un número como primo, divisible entre 12 o divisible entre 20 en el rango de 1 a 511.

Referencias

1. Jones, M. T.: AI Application Programming, Charles River Media, 2nd edition, (2005) 49–67
2. Johnson, D. and McGeoch, L.: The Traveling Salesman Problem: A Case Study in Local Optimization, in Local Search in Combinatorial Optimization, E. H. Aarts and J. K. Lenstra (eds.), Wiley and Sons.
3. Luke, B. T.: Simulated Annealing Cooling Schedules, available online at http://members.aol.com/btluke/simanf1.htm, accessed June 1, 2007.
4. Masters, T.: Practical Neural Network Recipes in C++. Academic Press, Inc., (1993) 118–134
5. Masters, T.: Signal and Image Processing With Neural Networks. John Wiley & Sons Inc., (1994)
6. Masters, T.: Advanced Algorithms for Neural Networks. John Wiley & Sons Inc., (1995) 135-156
7. Masters, T.: Neural, Novel & Hybrid Algorithms for Time Series Prediction. John Wiley & Sons Inc., (1995)
8. Metropolis, N., Rosenbluth, A., Rosenbluth, M. and Teller, E.: Journal of Chemical Physics, vol. 21, (1953) 1087–1092
9. Nilsson, N. J.: Artificial Intelligence: A New Synthesis, Morgan Kaufmann Publishers, Inc., (1998) 37–58
10. Reed, R. D., Marks II, R. J.: Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks, The MIT Press, (1999) 97–112
11. Russel, S. J. and Norvig, P.: Artificial Intelligence: A Modern Approach, Prentice Hall, 2nd edition, (2002)

Capítulo 6.

Computación evolutiva

1. Introducción

En este capítulo presentamos una introducción al área de Computación Evolutiva. Computación evolutiva, al igual que otras áreas de inteligencia artificial se conocen como técnicas bio-inspiradas. Dentro de estas técnicas se encuentran Redes Neuronales Artificiales, Enjambres de Partículas, Colonias de Hormigas, etc. Estas técnicas han estudiado los diferentes principios que subyacen sus análogos biológicos y los utilizan para desarrollar e implementar algoritmos y representaciones que imiten o copien (partes de) estos mecanismos para resolver problemas particulares.

Computación evolutiva se basa en los principios darwininanos de evolución y de supervivencia del más fuerte. En las diferentes especies que encontramos en la naturaleza, los individuos forman poblaciones; dentro de estas comunidades o poblaciones, los individuos mas aptos tienen myor probabilidad de sobrevivencia. En las especies biológicas, mas apto significa, en la mayoría de los casos, mas fuerte, mas alto, etc.

Las características genéticas de las especies se codifican en los cromosomas. Los cromosomas están a su vez divididos en genes, donde cada gen (o gene) representa una característica del individuo. En el estudio genético de las especies se distingue la codificación de las características genéticas, de las características mismas. De esta manera, a la codificación de las características en un cromosoma se le conoce como genotipo y a la interpretación de éstas se les conoce como fenotipo.

La información genética de un individuo (de una especie, por ende) se encuentra codificada en el cromosoma. La naturaleza le proporciona a las especies dos mecanismos genéticos: mediante la herencia, las características genéticas se transmiten de generación en generación; mediante las mutaciones, las especies adquieren diversidad genética.

El hecho de que los individuos mas aptos tengan mayor probabilidad de cruzarse y por ende reproducirse, asegura que los genes de los mejores individuos

pasen a la siguiente generación. Los individuos débiles morirán y/o sus genes no serán transmitidos a generaciones futuras.

Mediante mutaciones, los genes cambian de manera espontánea y no determinista. Estos cambios generan diversidad genética; ésto es, hace que características que no existían en los padres, de manera espontánea aparezcan en los hijos. Estas variaciones genéticas, pueden llegar a generar individuos mas aptos que sus padres; estos individuos con nuevas características, al ser dominantes, tendrán mas posibilidades de apareamiento y sus genes pasarán a generaciones posteriores.

Estos mecanismos, combinados, logran que las especies vayan mejorando de generación en generación, sobreviviendo y adaptándose al medio ambiente, muchas veces cambiante.

El área de computación evolutiva toma estos conceptos y los plasma en algoritmos y representaciones utilizables mediante computadoras digitales. Estos algoritmos y representaciones llevan poblaciones, mediante generaciones sucesivas, a tener individuos muy buenos con respecto a un criterio determinado conocido como función de aptitud. Computación evolutia es un mecanismo que nos permite optimizar una función (la función de aptitud), utilizando los conceptos que la naturaleza incluye en el desarrollo de especies, tal y como los entendemos a la fecha.

Dentro del área de computación evolutiva, se pueden distinguir varias vertientes, siendo las dos mas importantes Algoritmos Genéticos y Programación Genética. En este capítulo estudiaremos estas dos variantes, incluyendo algunos casos de estudio prácticos, de manera que el lector se de una idea del tipo de aplicaciones que esta área de Inteligencia Artificial resuelve. Dado que el área trata acerca de optimización, comenzaremos con un recuento de varias técnicas alternativas que resuelven este tipo de problemas. Al final del capítulo se incluye una sección con referencias de varios tipos, de manera que el lector interesado pueda acceder a ellas.

2. Optimización

Podemos definir optimización, como la búsqueda de la mejor manera de realizar una actividad. El hombre en busca de mejorar el desempeño de las tareas que realiza ha estudiado y creado un área de investigación a la cual denominó optimización y ésta puede ser definida como la ciencia que intenta buscar la manera de realizar una tarea, con el máximo beneficio. De aquí se desprende una pregunta que resulta crucial para poder realizar la optimización de cualquier actividad, ¿Cómo mido el beneficio/costo?, ¿Qué variables influyen en el beneficio? ¿Es posible establecer una función matemática que permita de manera inequívoca modelar nuestra tarea o actividad?. Tal vez esta sea la parte más difícil de la optimización, ya que una vez establecida la función de beneficio/costo podemos hacer uso de las herramientas y métodos existentes.

De manera formal podemos definir una función objetivo $f(x) : \mathcal{R}^n \rightarrow \mathcal{R}$ y queremos dar solución al siguiente problema:

$$\underset{x \in \Omega}{\text{máx(mín)}}\ f(x) \tag{1}$$

donde $x = [x_1, x_2, \cdots, x_n]^T$ es un vector que representa las variables de decisión de la función objetivo $f(x)$, la cual mide el beneficio de una combinación de variables de decisión y Ω es un espacio de búqueda infinito o que puede estar limitado por restricciones. Una manera de resolver el problema planteado por la ecuación (1) es calcular valores aleatorios de la variables de decisión y en el momento que obtengamos una solución lo suficientemente buena, parar. Este es el método búsqueda estocástico más simple, sin embargo, garantizar que tenemos el valor óptimo solo se puede logar cuando el proceso se repite un número de veces lo suficientemente grande; si el espacio de búsqueda es infinito las iteraciones necesarias serán infinitas.

2.1. Optimización analítica

Otra forma de solucionar el problema planteado por la ecuación (1) es utilizar el teorema de máximos y mínimos del cálculo diferencial e integral. Aplicando este teorema tenemos que solucionar el sistema de ecuaciones dado por la ecuación (2); en el mejor de los casos, el sistem puede ser lineal, o bien no-lineal. Con este enfoque el número de iteraciones necesarios para resolver un problema resulta ser finito.

$$\nabla f(x) = \left[\frac{\partial f(x)}{\partial x_0}\ \frac{\partial f(x)}{\partial x_1} \cdots \frac{\partial f(x)}{\partial x_n} \right]^T = 0 \tag{2}$$

Basado en la ecuación (2) existen varios métodos para calcular el óptimo; de ahí que estos métodos son llamados métodos basados en gradiente. Algunas de estas técnicas se describen a continuación.

2.2. Métodos de optimización basados en gradiente

El más simple de los métodos de optimización basados en gradiente es el método de Descenso de Gradiente (DG). Dada una función continua f, el mínimo es buscado utilizado la siguiente sucesión

$$x^{t+1} = x^t + \alpha p^t \tag{3}$$

donde α es un escalar denominado tamaño de paso y es común que sea un número pequeño y constante durante todo del proceso de minimización; p es una dirección de búsqueda. En el método DG, la dirección de máximo descenso es la dirección opuesta al gradiente. Esto es, $p^t = -\nabla f\left(x^t\right)$.

Una mejora que podemos hacer al método de descenso de gradiente es tratar de calcular un tamaño de paso α^t variable en cada interación, tal que $f\left(x^t + \alpha^t p^t\right) <$

$f(x^t)$. Sin embargo el tamaño de paso es constante en todas las direcciónes de minimización.

El método de Newton, hace una aproximación quadrática de la función $f(x)$, utilizando la serie de Taylor. Será indispensable que para la función existan la primera y segunda derivadas. La aproximación cuadrática en el punto x^t, está dada por la ecuación (4)

$$q(x^{t+1}) = f(x^t) + \left(x^{t+1} - x^t\right) \nabla f(x^t) + \frac{1}{2} \left(x^{t+1} - x^t\right)^T \nabla^2 f(x^t) \left(x^{t+1} - x^t\right)^T \tag{4}$$

donde $\nabla^2 f$ es llamada la matriz Hessiana y contiene información de las segundas derivadas de la función $f(x)$. Para calcular el mínimo de la ecuación (4) hacemos $\nabla q(x^{t+1}) = 0$, lo cual da lugar a un sistema lineal de ecuaciones que es el resultado en cada iteración mediante la sucesión dada por la ecuación (5)

$$x^{t+1} = x^t - \left[\nabla^2 f(x^t)\right]^{-1} \nabla f(x^t) \tag{5}$$

Podemos notar que el método de Newton utiliza un tamaño de paso variable en cada dirección, el cual es función del Hessiano; la dirección de búsqueda es el gradiente en ambos casos. Cuando se utiliza la dirección de gradiente, las direcciones son ortogonales y el algoritmo avanza con pasos en zigzag, tal como se muestra en la figura 1, ya que $\left[\nabla f(x^{t+1})\right]^T \nabla f(x^t) = 0$.

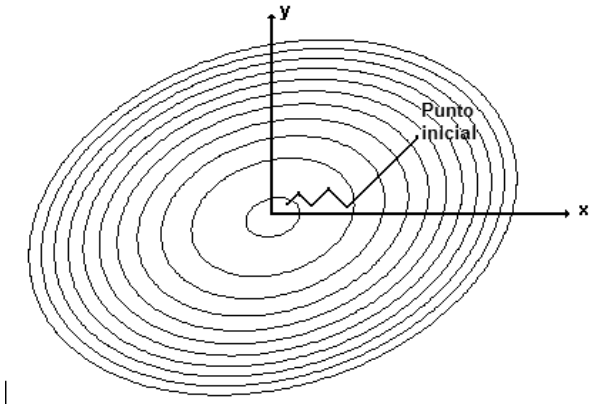


Figura 1. Ejemplo del comportamiento de DG

Un método que resuelve el efecto de zigzag, es el método de Gradiente Conjugado (GC), donde se pide como requisito que las direcciones sean conjugadas. Esto es, $p_i^T A p_j$ $(\forall i \neq j)$. Asumiremos que minimizaremos una función

cuadrática de la forma $q(x) = \frac{1}{2}x^T Ax + bx + c$, donde cada término está relacionado con la ecuación (4). La idea del método es construir progresivamente direcciones $p^0, p^1, \ldots, p^t$, las cuales son conjugadas respecto a la matriz A. En cada estado t la dirección p^t es obtenida por la combinación lineal del gradiente ($r^t = Ax^t - b$) en cada posición x^t y las direcciones previas $p^0, p^1, \ldots, p^{t-1}$. La sucesión para calcular las direcciones conjugadas es $p^{t+1} = -r^{t+1} + \beta^{t+1}p^t$, con un parámetro $\beta^{t+1} = \frac{\left[r^{t+1}\right]^T r^{t+1}}{[r^t]^T r^t}$. Las actualizaciones son llevadas a cabo utilizando la ecuación (3) con un tamaño de paso $\alpha^t = -\frac{\left[r^{t+1}\right]^T r^t}{[p^t]^T Ap^t}$. Si la función a minimizar es cuadrática, el algoritmo de GC garantiza la convergencia en n iteraciones (donde n es el número de variables). El comportamiento de este método lo podemos ver en la figura 2. En el caso de no tener una función cuadrática, el método es llamado GC no lineal. La diferencia principal del método de GC con el método de Newton es que para el primero no es necesario calcular la inversa de un Hessiano de tamaño $n \times n$, lo cual mejora el desempeño en caso de tener un sistema de ecuaciones disperso.

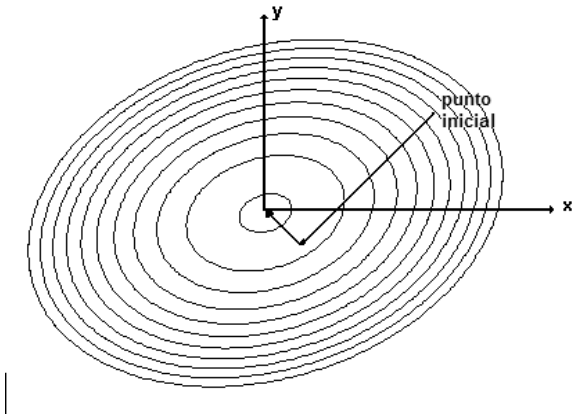


Figura 2. Comportamiento de GC

Los algoritmos basados en gradiente solamente garantizan alcanzar un mínimo local y en el caso de una función multimodal, el mínimo local será alcanzado si y solo si el punto inicial x_0 se encuentra lo suficientemente cerca al óptimo. Adicionalmente, si las derivadas no pueden ser calculadas para la función objetivo, la búsqueda no podrá llevarse a cabo. Otra alternativa la presentan los algoritmos estocásticos como búsqueda aleatoria, recocido simulado, algoritmos genéticos, etc.

3. Búsqueda aleatoria

El método de Búsqueda Aleatoria (BA), evalúa en forma repetida la función mediante la selección aleatoria de valores de la variable independiente. Si un número suficiente de muestras se lleva a cabo, el óptimo será eventualmente alcanzado.

Dada una función $f(x)$ en el dominio acotado $x_i \in [x_i^{\text{mín}}, x_i^{\text{máx}}]$, se genera un número aleatorio $\gamma \in [0, 1]$, los valores de x_i se calculan como:

$$x_i = x_i^{\text{mín}} + (x_i^{\text{máx}} - x_i^{\text{mín}}) * \alpha \tag{6}$$

Así por ejemplo si deseamos encontrar el máximo de la función $f(x) = x_2 - x_1 - 2x_1^2 - 2x_1x_2 - x_2^2$ en el dominio acotado $x \in [-2, 2]$ y $y \in [1, 3]$, los resultados al aplicar este método para la función dada se presentan en el cuadro 1.

t	x_1	x_2	$f(x, y)$
0	0.00000	0.00000	0.00000
1	-0.84276	1.19252	1.20270
2	-0.84276	1.19252	1.20270
3	-0.84276	1.19252	1.20270
4	-0.84276	1.19252	1.20270
5	-0.84276	1.19252	1.20270
6	-0.84276	1.19252	1.20270
7	-0.84276	1.19252	1.20270
8	-0.84276	1.19252	1.20270
9	-0.84276	1.19252	1.20270
167	-0.92622	1.28268	1.22395
183	-1.11212	1.49897	1.22463
465	-1.04210	1.66243	1.23375
524	-0.96660	1.36514	1.23859
626	-0.97539	1.48462	1.24931
999	-0.97539	1.48462	1.24931

Cuadro 1. Mil iteraciones del método de BA para la función $f(x) = x_2 - x_1 - 2x_1^2 - 2x_1x_2 - x_2^2$

La implementación de este algoritmo de BA en java se muestra en la figura 3.

```
void BusquedaAleatoria() {
  int i;
  double r, x, y, max=0, maxx=0, maxy=0, fx;
  double xl=-2, xu=2, yl=1, yu=3;
  Random azar=new Random();

  for(i=0; i < 1000; i++) {
     r=azar.nextDouble();
     x=xl + (xu- xl)*r;
     r=azar.nextDouble();
     y=yl + (yu- yl)*r;
     fx=f(x,y);

     if(fx > max) {
        max=fx;
        maxx=x;
        maxy=y;
     }
     System.out.println(i+maxx+maxy+max);
  }

  double f(double x, double y) {
     return y-x-2*x*x-2*x*y-y*y;
  }
```

Figura 3. Implementación de búsqueda aleatoria en java

4. Recocido simulado

Las técnicas de Recocido Simulado (RS) (en inglés, *simulated annealing*), tienen su origen en los procesos heurísticos que simulan el comportamiento de un grupo de átomos inicialmente en equilibrio a una temperatura T y que son expuestos a enfriamiento. Un enfriamiento rápido bloquea el sistema a un estado de alta energía que corresponde a un desorden, mientras que un enfriamiento lento o recocido, lleva al sistema a un estado ordenado y con baja energía. Tal comportamiento se reproduce por los programas de RS, los cuales intentan simular el comportamiento que tiene un sistema de átomos en movimiento con energía E y temperatura T. En el sistema se elige un átomo de manera aleatoria y se le aplica un desplazamiento aleatorio. Considerando el cambio de energía dE, si dE fuera negativo entonces se acepta el desplazamiento y a la energía E se le resta la diferencia dE. Si, en cambio, dE fuera positivo, se evalúa, según la ley de Boltzmann, la probabilidad de que el cambio dE tenga lugar: $Prob(dE) = e^{-dE/\kappa T}$, donde κ es la constante de Boltzmann y T es la temperatura del sistema. La probabilidad $Prob(dE)$ se compara con el valor de una variable aleatoria γ uniformemente distribuida en el intervalo $[0, 1]$. El desplazamiento se acepta si y sólo si γ no supera a $Prob(dE)$. Esta iteración básica se repite tantas veces co-

mo queda establecido por la ley de Boltzmann hasta que el sistema alcance un estado más estable a una temperatura T. En ese entonces T se multiplica por una constante de decaimiento en el intervalo $[0, 1]$ y la sucesión de iteraciones se repite. El valor mínimo de la energía se decrementa con T así como la probabilidad de aceptar los desplazamientos. Cuando T es suficientemente pequeño, no se aceptan más desplazamientos aleatorios y el sistema se congela en un estado casi estable que corresponde a un mínimo local de E. Esta heurística para problemas de optimización tiene las ventajas siguientes:

1. La heurística no se detiene al encontrar el primer óptimo local, sino que con cierta probabilidad positiva, procede a subsecuentes exploraciones del espacio de búsqueda.
2. La heurística exhibe a altas temperaturas un comportamiento similar a la técnica de divide y vencerás.
3. Bajo aseveraciones razonables, la heurística converge (quizá muy lentamente) a un óptimo global.

La figura 4 presenta el código en java correspondiente a este algoritmo.

```
1. i=i0  //Valor inicial
2. T=T0  //Temperatura inicial
3. K=K0  //Máximo de Iteraciones
4. while (condición de Paro)
5.  while (k<K && a<A)
6.      generar j en N(i)    //Vecindad de i
7.      if (f(j)-f(i)<0)
8.          i=j
9.          a=a+1
10.     else
11.         generar un número r al azar (pseudo-aleatorio)
12.         if (r<exp [(f(i)-f(j))/T])
13.             i=j
14.             a=a+1
15.     k=k+1
16. if (a=A)
17.     T=alfa*T
18. else (k=K)
19.     T=nu*T
20. K=ro*K
21. k=0
22. a=0
23. mostrar i, c(i)
```

Figura 4. Implementación de recocido simulado en java

5. Evolución

Se denomina evolución a cualquier proceso de cambio en el tiempo. En el contexto de las ciencias de la vida, la evolución es un cambio en el perfil genético de una población de individuos. Este cambio puede llevar a la aparición de nuevas especies, a la adaptación a distintos ambientes, o a la aparición de novedades evolutivas. Esta definición es el punto de partida para la creación de los Algoritmos Genéticos (AG), los cuales crean un mecanismos de evolución de una población donde cada individuo representa el vector x de variables de decisión.

5.1. Darwin, adaptabilidad y evolución

El concepto clásico de selección natural afirma que las condiciones de un medio ambiente (o "naturaleza") favorecen o dificultan (seleccionan) la supervivencia o reproducción de los organismos vivos según sus características. La selección natural fue propuesta por Darwin como medio para explicar la evolución biológica. Esta explicación parte de dos premisas, la primera de ellas afirma que entre los descendientes de un organismo hay una variación aleatoria (no determinista), que es en parte heredable. La segunda premisa sostiene que esta variabilidad puede dar lugar a diferencias de supervivencia y de éxito reproductor, haciendo que algunas características de nueva aparición se puedan extender en la población. La acumulación de estos cambios a lo largo de las generaciones produciría todos los fenómenos evolutivos.

La selección natural puede ser expresada como la siguiente ley general (tomada de la conclusión de "El origen de las especies"):

1. Si existen organismos que se reproducen,
2. si la progenie hereda características de sus progenitores,
3. si existen variaciones de características y
4. si el medio ambiente no admite a todos los miembros de una población en crecimiento

entonces aquellos miembros de la población con características menos adaptadas (según lo determine su medio ambiente) morirán con mayor probabilidad y aquellos miembros con características mejor adaptadas sobrevivirán más probablemente. El resultado de la repetición de este esquema a lo largo del tiempo es la evolución de las especies.

Podemos llevar esta representación para una función a optimizar donde seguiremos los pasos mencionados. Los organismos a reproducir serán el conjunto de variables de decisión y la función objetivo es el mecanismo que nos permitirá evaluar cuales son los individuos mejor y peor adaptados.

6. Algoritmos genéticos

Un Algoritmo Genético (AG) es una heurística que imita el mecanismo de evolución biológica con el propósito de maximizar el beneficio de una tarea. Las

bases de los AG fueron establecidos en 1962 por John Holland. Holland fue el primero en proponer explícitamente el cruzamiento y otros operadores de recombinación. Sin embargo, el trabajo fundamental en el campo de los AG apareció en 1975, con la publicación "Adaptación en sistemas naturales y artificiales".

Los AG toman la teoría de la selección natural darwiniana y los pasos que deben repetirse son:

1. Inicializar aleatoriamente una población de soluciones a un problema, representadas por una estructura de datos adecuada.
2. Evaluar cada una de las soluciones y asignarle una puntuación o aptitud,
3. Escoger de la población la parte que tenga una puntuación mayor,
4. Recombinar (recombinación y cruza)
5. Mutar (cambiar) y
6. Repetir un número determinado de veces, o hasta que se haya encontrado la solución deseada.

Dadas estas simple reglas solo resta definir como haremos la representación de un individuo y realizaremos cada una de las operaciones.

6.1. Cromosomas

Un cromosoma representará al conjunto de variables de decisión, dado por el vector de decisión x. A cada elemento de x_i, le llamaremos gen. En la figura 5 se muestra un punto de coordenadas [2,2] en el plano xy, el cual es representado por un individuo de la población (codificado en un cromosoma) y cuya altura corresponde a la evaluación de la función de aptitud.

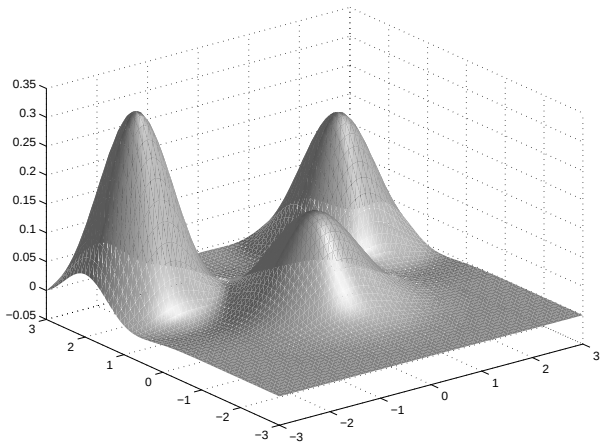


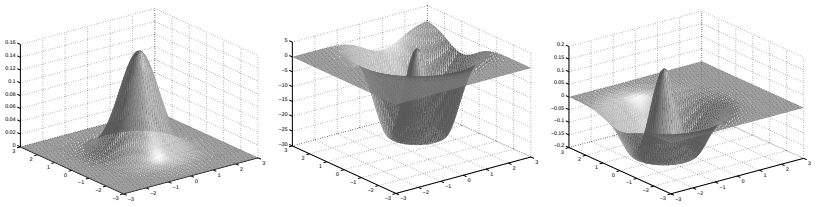
Figura 5. Representación cromosómica de una función en tres dimensiones

Para generar la población inicial debemos comenzar por definir un individuo. Un individuo de nuestra población estará representado por el conjunto de variables de decisión x de nuestra función objetivo $C_k = x_1, x_2, \cdots, x_i, \cdots, x_n$ y a ésto le denominaremos cromosoma del $k - esimo$ individo. El cromosoma está compuesto por genes y un gen representara la $i-$ésima variable de decisión. Así, en el caso de tener una función de dos variables nuestro comosoma tendrá solamente dos genes.

Por ejemplo, consideremos la función

$$f(x,y) = \frac{1}{2\pi\sigma} e^{\left(-(x-x_0)^2-(y-y_0)^2\right)} \left(\frac{2}{\sigma} - \frac{(x-x_0)^2+(y-y_0)^2}{\sigma^4}\right) \qquad (7)$$

podemos crear toda una familia de funciones cambiando los valores de x_0, y_0 y σ. En forma de cromosoma tenemos que estos parámetros son $C_k = x_0, y_0, \sigma$. En la figura 6 se muestra ejemplos de la función con diferentes cromosomas.



(a) Cromosoma {0, 0, 5}

(b) Cromosoma {1, 1, 0.1}

(c) Cromosoma {-1, 0,0.5}

Figura 6. Diferentes cromosomas para la función de la ecuación 7

7. Algoritmos genéticos con codificación binaria

En este caso el algoritmo genético trabaja con un conjunto finito de valores y son ideales para minimizar funciones que toman valores en un espacio discreto.

7.1. Representación de parámetros

Si el parámetro es continuo, entonces debe ser cuantizado. Las formulas para hacer la codificación binaria y la decodificación de $i-$ésimo parámetro, x_i, son dados como codificación: Para este paso se hace un escalamiento del intervalo $[x^{\text{mín}}, x^{\text{máx}}]$ al intervalo $[0, 1]$, utilizando la ecuación (7.1)

$$\overline{x}_i = \frac{x_i - x_{\text{mín}}}{x_{\text{máx}} - x_{\text{mín}}}$$

Posteriormente se hace una codificación binaria del número resultante. La figura 7 ilustra este método, expresado en java.

```
static public int[] codificar(double P, int k) {
  double Pnorm=(P- Rangos[k][0])/(Rangos[k][1]- Rangos[k][0]);
  double suma;
  int gene[]=new int[Ngene];
  int i, aux;

  for(i=0; i<Ngene; i++) {
    Pnorm*=2.0;
    aux=(int) Pnorm;
    gene[i]=aux;
    Pnorm-=aux;
  }
  return gene;
}
```

Figura 7. Implementación del método de codificación en java

Para la decodificación, primeramente hacemos una conversión de binario a decimal y el valor del parámetro lo calculamos utilizando la ecuación (8)

$$x_i = \overline{x}_i(x^{\text{máx}} - x^{\text{mín}}) + x^{\text{mín}} \tag{8}$$

El método de decodificación, escrito en java se ilustra en la figura 8

```
static public double decodificar(int gene[], int k) {
  int i, n=gene.length;
  double suma=0, fac=0.5, valor;

  for(i=0; i<n; i++) {
    suma+=gene[i]*fac;
    fac /=2.0;
  }

  valor=suma*(Rangos[k][1]-Rangos[k][0])+Rangos[k][0];
  return valor;
}
```

Figura 8. Implementación del método de decodificación en java

Una vez realizada la codificación, un ejemplo de un cromosoma con $N_{gene} = 3$ y un número de parámetros $N_{par} = 4$ se ilustra en la figura 7.1.

001	111	101	010

7.2. Población inicial

El algoritmo genético comienza con una población inicial de tamaño N_{ipop}, la cual es una matriz de tamaño $N_{ipop} \times N_{bits}$, donde N_{bits} es el total de bits para representar los N_{par} de tamaño N_{gene}. El método para generar la población inicial en java se ilustra en la figura 9

```
static public void generaPoblacion() {
  int i, j;
  int k=0;

  for (i=0; i<Npop; i++) {
    k=0;
    for (j=0; j < Npar; j++) {
      GeneraCromosoma(i, j);
    }
    Costos[i]=funcion(Cromosomas[i]);
  }
  ordena();
  Npop/=2.0;
}

static public void GeneraCromosoma(int i, int j) {
  int k;

  for(k=0; k<Ngene; k++)
    Cromosomas[i][j][k]=(int) (r.nextDouble()+0.5);
}
```

Figura 9. Generación de la Población Inicial en java

7.3. Selección natural

Normalmente la población inicial es muy larga y es deseable reducirla eliminando algunos elementos de la población. Para ello se aplica una rutina de ordenamiento que nos permitirá seleccionar a los mas aptos (una vez evaluada una función de costo). Así para una población inicial, reduciremos la población a la mitad de la población inicial $N_{pop} = N_{ipop}/2$. La población mantendrá este tamaño durante todo el proceso iterativo. De esta población de tamaño N_{pop}, definiremos una parte que serán los mejores elementos N_{good} y los peores elementos N_{bad}. Los primeros, los N_{good} serán los candidatos a formar parejas y

tener descendencia, los segundos serán reemplazados por la descendencia de los N_{good} elementos.

Consideremos como ejemplo la función continua $f(x) = x_1 seno(4x_1)+1{,}1x_2 seno(2x_2)$, para la cual deseamos calcular una población inicial en el rango $[0, 10]$ para ambas variables utilizando un cromosoma de 20 bits y una población inicial de 24 individuos, la cual se presenta en el cuadro 2. En la figura 10, se muestra la función utilizada en este ejemplo y en la figura 11 la población inicial.

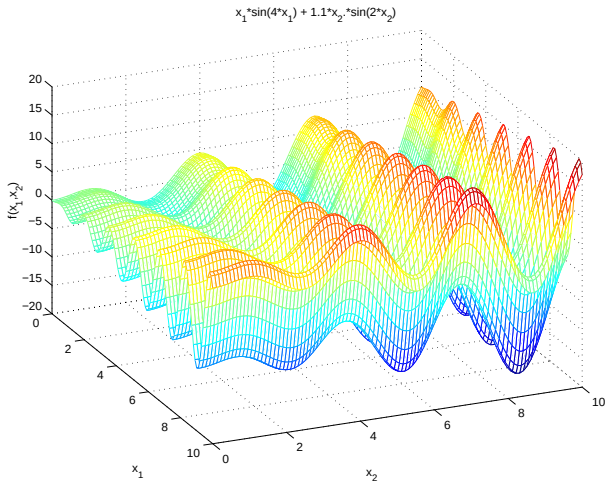


Figura 10. Función $f(x) = x_1 seno(4x_1) + 1{,}1x_2 seno(2x_2)$

7.4. Apareamiento

Existen diferentes formas de seleccionar a las parejas que tendrán descendencia. En el proceso de apareamiento es importante que las parejas se seleccionen de manera aleatoria. Algunas alternativas son:

1. Apareamiento de arriba a abajo. En este esquema se hacen parejas con los $Ngood$ individuos de la siguiente forma $[1, 2]$, $[3, 4]$, $[5, 6]$, ... $[N_{good}-1, N_{good}]$.
2. Apareamiento aleatorio. En este caso se generan parejas aleatoria en el rango $[1, N_{good}]$.
3. Apareamiento aleatorio pesado. En este caso se hace un aparemiento donde la probabilidad de apareamiento de cada uno de los individuos no es uniforme. Esta probabilidad dependerá del costo asociado a cada individuo, así el mas apto será el que tenga mayor probabilidad de apareamiento. Esta se calcula utilizando la ecuación 9

$$p_i = \frac{c_i}{\sum_{i=1}^{Ngood} c_i} \tag{9}$$

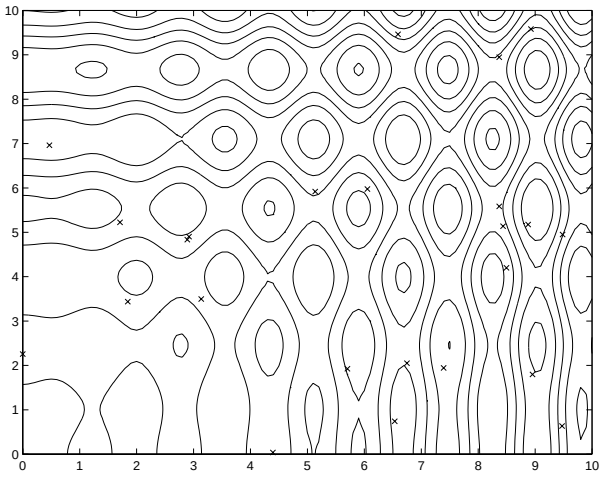


Figura 11. Población inicial para la función $f(x) = x_1 seno(4x_1) + 1{,}1x_2 seno(2x_2)$

donde $c_i = Costo_i - Costo_{Ngood+1}$.

El mejor esquema de apareamiento es el aleatorio pesado. Para nuestro ejemplo, las probabilidades pesadas se muestran en el cuadro 3.

Para calcular las parejas se genera un número aleatorio $\alpha \in [0, 1]$. Se elige al $i-$ésimo individuo tal que $\alpha \leq \pi_i$.

7.5. Descendencia

Una vez calculadas las parejas, pasamos al proceso de generar descendencia. Para cada una de las parejas se selecciona un bit en el cual se llevará a cabo el proceso de cruzamiento. Esta posición se calcula de manera aleatoria.

Supongamos que aleatoriamente se generaron las parejas $[(3, 2), (0, 3), (2, 0)]$ y un cruzamiento aleatorio en el bit 7, 6 y 10, respectivamente se tienen las siguientes parejas e hijos resultantes los cuales se muestran en el cuadro 4.

7.6. Mutación

Este procedimiento se realiza totalmente de manera aleatoria. Normalmente el número de individuos a mutar va del 1 al 20 %, aunque este valor pudiera cambiar. Es deseable no mutar al mejor individuo de la población, por lo cual, se generan números aleatorios en el rango $[2, N_{good}]$; una vez elegido el individuo a mutar, se debe decidirr cual es el bit que se va a cambiar.

Vamos a suponer que se mutan el bit 18 del cromosoma 4 y el bit 3 del cromosoma 2. Esta operación se muestra en el cuadro 4.

Ind	$Gene_1$	$Gene_2$	x_1	x_2	$f(x_1, x_2)$
0	1110001101	1000010010	8.876953125	5.17578125	-11.775107946435504
1	1110010101	0010111000	8.955078125	1.796875	-9.397234102008252
2	1001101100	1001100100	6.0546875	5.9765625	-8.57861714158948
3	1011110101	0011000111	7.392578125	1.943359375	-8.564664578124413
4	1001001000	0011000101	5.703125	1.923828125	-5.548160097898145
5	1110010010	1111010101	8.92578125	9.580078125	-4.910272831483166
6	0100101011	0111110110	2.919921875	4.90234375	-4.262645021922337
7	0111000010	0000000100	4.39453125	0.0390625	-4.195725821467517
8	0010101111	1000010111	1.708984375	5.224609375	-4.013162754305918
9	0100101000	0111101111	2.890625	4.833984375	-3.7188337466840142
10	0000000000	0011100111	0.0	2.255859375	-2.4316506543930565
11	1101011001	1110010100	8.369140625	8.9453125	-0.6696799445115991
12	1111001011	0111111011	9.482421875	4.951171875	-0.3361225990105128
13	1000001110	1001011110	5.13671875	5.91796875	0.7523936350592972
14	1101011001	1000111100	8.369140625	5.5859375	1.3357129488105146
15	1101100000	1000001110	8.4375	5.13671875	1.8566966237160756
16	0101000001	0101100110	3.134765625	3.49609375	2.4182539844896622
17	1111001010	0001000001	9.47265625	0.634765625	2.4698636738114845
18	0010111101	0101100000	1.845703125	3.4375	3.754030385201311
19	1010110011	0011010010	6.748046875	2.05078125	4.620931822460583
20	1010011101	0001001100	6.533203125	0.7421875	6.311111856904211
21	1010100011	1111001001	6.591796875	9.462890625	7.015104449242521
22	0000110000	1011001001	0.46875	6.962890625	7.935922596234852
23	1101100110	0110101110	8.49609375	4.19921875	8.55848738648271

Cuadro 2. Población inicial para el problema de la figura 10

i	$Costo_i$	$Costo_i - Costo_{N_{good}+1}$	p_i	$\pi_i = \sum p_i$
0	-11.7751	-7.5125	0.3238	0.3238
1	-9.3972	-5.1346	0.2213	0.5452
2	-8.5786	-4.3160	0.1860	0.7312
3	-8.5647	-4.3020	0.1854	0.9167
4	-5.5482	-1.2855	0.0554	0.9721
5	-4.9103	-0.6476	0.0279	1.0000

Cuadro 3. Ejemplo de apareamiento aleatorio pesado

	Ind	$Cromosoma$	x_1	x_2	$f(x_1, x_2)$
$padre$	3	10111101 010011000111	7.392578125	1.943359375	-8.564664578124413
$madre$	2	10011011 001001100100	6.0546875	5.9765625	-8.57861714158948
$hijo_1$	6	10111101 001001100100	7.3828125	5.9765625	-10.805733576930972
$hijo_2$	7	10011011010011000111	6.064453125	1.943359375	-6.103949823195019
$padre$	0	1110001 1011000010010	8.876953125	5.17578125	-11.775107946435504
$madre$	3	1011110 1010011000111	7.392578125	1.943359375	-8.564664578124413
$hijo_1$	8	1110001 1010011000111	8.876953125	1.943359375	-8.67166082376904
$hijo_2$	9	1011110101100001001O	7.392578125	5.17578125	-11.668111700790877
$padre$	2	1001101100 1001100100	6.0546875	5.9765625	-8.57861714158948
$madre$	0	1110001101 1000010010	8.876953125	5.17578125	-11.775107946435504
$hijo_1$	10	1001101100 1000010010	6.0546875	5.17578125	-9.347934290467709
$hijo_2$	11	1110001101100110010O	8.876953125	5.9765625	-11.005790797557275

Cuadro 4. Ejemplo de cruzamiento

Este procedimiento se repite hasta obtener convergencia. Para el ejemplo de la función $f(x) = x_1 seno(4x_1) + 1{,}1x_2 seno(2x_2)$, la figura12, ilustra como

Ind	$Cromosoma$	x_1	x_2	$f(x_1, x_2)$
0	11100011011000010010	8.876953125	5.17578125	-11.775107946435504
1	11100101010010111000	8.955078125	1.796875	-9.397234102008252
2	100 $\overset{\frown}{0}$ 1011001001100100	5.4296875	5.9765625	-2.3228094178933736
3	10111101010011000111	7.392578125	1.943359375	-8.564664578124413
4	100100100000110001 $\overset{\frown}{1}$ 1	5.703125	1.943359375	-5.6245656618150175
5	11100100101111010101	8.92578125	9.580078125	-4.910272831483166
6	10111101001001100100	7.3828125	5.9765625	-10.805733576930972
7	10011011010011000111	6.064453125	1.943359375	-6.103949823195019
8	11100011010011000111	8.876953125	1.943359375	-8.67166082376904
9	10111101011000010010	7.392578125	5.17578125	-11.668111700790877
10	10001011001000010010	5.4296875	5.17578125	-3.0921265667716025
11	11100011011001100100	8.876953125	5.9765625	-11.005790797557275

Cuadro 5. Ejemplo de mutación

el AG converge. La figura13, muestra la población en la última generación del algoritmo.

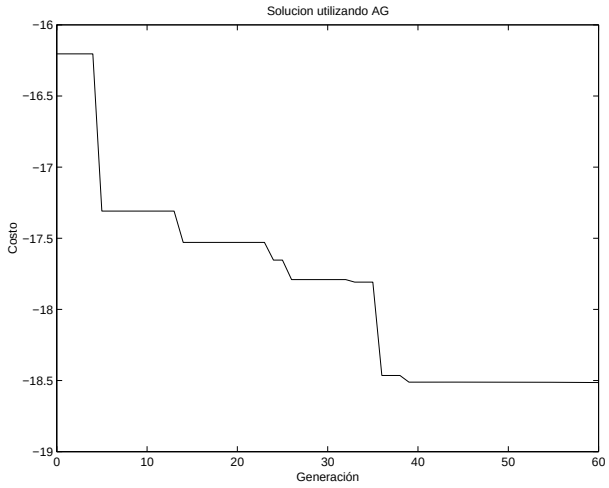


Figura 12. Solución usando AG para la Función $f(x) = x_1 seno(4x_1) + 1{,}1x_2 seno(2x_2)$

8. Algoritmos genéticos con codificación real

Dada una función $f : \mathcal{R}^n$ la solución encontrada por los algoritmos Genéticos con codificación binaria, no darán una solución con la precisión necesaria, ya que esta limitada al conjunto de valores que el proceso de cuantización puede ver. Esto se puede mejorar a medida que se aumenta el número de bits, sin embargo el tiempo se ejecución se incrementa.

Los pasos que seguiremos serán los mismos, la diferencia radica en como determinar los valores.

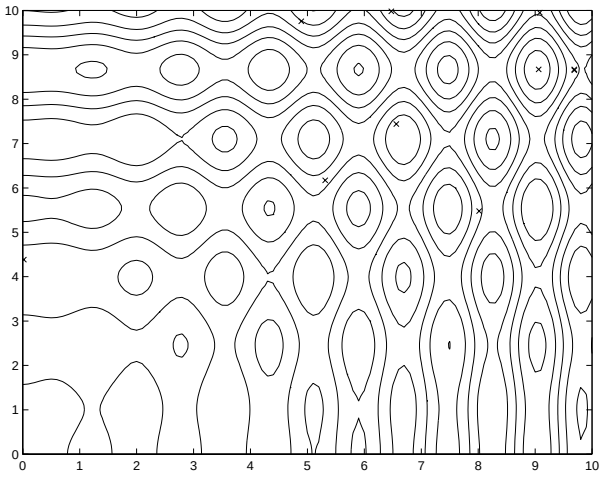


Figura 13. Población Final para la Función $f(x) = x_1 seno(4x_1) + 1{,}1x_2 seno(2x_2)$

8.1. Población inicial

Para la población inicial generaremos aleatoriamente una población de tamaño N_{ipop} donde cada cromosoma tiene la forma $C = \{c_1, c_2, \cdots, c_j, \cdots, c_{N_{par}-1}, c_{N_{par}}\}$, y los parámetros se calculan por medio de $c_j = (c_{\text{máx}} - c_{\text{mín}})\alpha + c_{\text{mín}}$, donde $\alpha \in [0, 1]$ es un número aleatorio. Utilisando estas fórmulas se tendrán números aleatorios en el rango de cada uno de los parámetros que representan la función.

Una vez generada la población inicial, se seleccionan los mejores individuos que formen el 50 % de éstos. Esta será la población inicial y el tamaño se conservará durante el proceso de convergencia del algoritmo.

8.2. Apareamiento

Para el proceso de apareamiento consideraremos los mismos casos que en los AG con codificación binaria. Esto se debe a que para la selección de las parejas interviene solamente el costo y no la representación de los cromosomas.

8.3. Cruzamiento

El procedimiento de cruzamiento es llevado a cabo después de seleccionar aleatoriamente a las parejas. Dado un par de padres p y m, el procedimiento de apareo genera dos descendientes usando una pareja de cromosomas de acuerdo con

$$C^{(m)} = \boxed{c_1^{(m)}}\boxed{c_2^{(m)}}\boxed{\cdots}\boxed{c_i^{(m)}}\boxed{\cdots}\boxed{c_{N_{par}}^{(m)}}$$

$$C^{(p)} = \boxed{c_1^{(p)}}\boxed{c_2^{(p)}}\boxed{\cdots}\boxed{c_i^{(p)}}\boxed{\cdots}\boxed{c_{N_{par}}^{(p)}}$$

Entonces un punto de cruzamiento j es seleccionado aleatoriamente en el intervalo $[1, N_{gen}]$ y los parámetros alrededor del gene j son intercambiados. Los hijos generados muestran las ecuaciones (10) y (11).

$$hijo^{(1)} = \boxed{c_1^{(m)}}\boxed{c_2^{(m)}}\boxed{\cdots}\boxed{c_j^{(nuevo_1)}}\boxed{\cdots}\boxed{c_{N_{par}-1}^{(p)}}\boxed{c_{N_{par}}^{(p)}} \tag{10}$$

$$hijo^{(2)} = \boxed{c_1^{(p)}}\boxed{c_2^{(p)}}\boxed{\cdots}\boxed{c_j^{(nuevo_2)}}\boxed{\cdots}\boxed{c_{N_{par}-1}^{(m)}}\boxed{c_{N_{par}}^{(m)}} \tag{11}$$

El gene en la posición j es combinado usando la ecuación (13).

$$\begin{aligned} c_j^{(nuevo_1)} &= c_j^{(m)} - \beta(c_j^{(m)} - c_j^{(p)}) \\ c_j^{(nuevo_2)} &= c_j^{(p)} + \beta(c_j^{(m)} - c_j^{(p)}) \end{aligned} \tag{12}$$

donde $\beta \in [0, 1]$ es un número aleatorio.

8.4. Mutación

Para este procedimiento es necesario especificar una probabilidad de mutación. Recuerde que la mutación es un procedimiento que permite diversificar la búsqueda en regiones no exploradas.

Para el proceso de mutación seleccionaremos aleatoriamente un número entre 2 y $Npop$ (nunca seleccionaremos al individuo mejor adaptado para mutar). Posteriormente se selecciona aleatoriamente el parámetro del cromosoma que se alterará (entre 1 y $Npar$) y se calcula utilizando:

$$c_j = (c_{\text{máx}} - c_{\text{mín}})\alpha + c_{\text{mín}}$$

En la figura 14 podemos ver el desempeño del algoritmo genético para la función $f(x) = x_1 seno(4x_1) + 1{,}1x_2 seno(2x_2)$ planteada en el ejemplo. Note el mejor desempeño en este caso de la representación real (ver la figura 12).

9. Caso de estudio: registro de imágenes

Un problema del procesamiento digital de señales es el Registro de Imágenes (RI). El RI es definido como la tarea de empatar una imagen origen I_1 con una images destino I_2, para lo cual debemos calcular la transformación lineal que fue aplicada a la imagen origen. Esta transformación lineal puede ser una transformación afín la cual permite realizar traslaciones, rotaciones, escalamiento, cizallamiento y combinaciones de éstas. La Transformación Afín (TA) está dada por la ecuación (13)

$$\begin{bmatrix} \widehat{r}_i(x) \\ \widehat{s}_i(x) \end{bmatrix} = \begin{bmatrix} x_0 & x_1 & x_2 \\ x_3 & x_4 & x_5 \end{bmatrix} \begin{bmatrix} r_i \\ s_i \\ 1 \end{bmatrix} \tag{13}$$

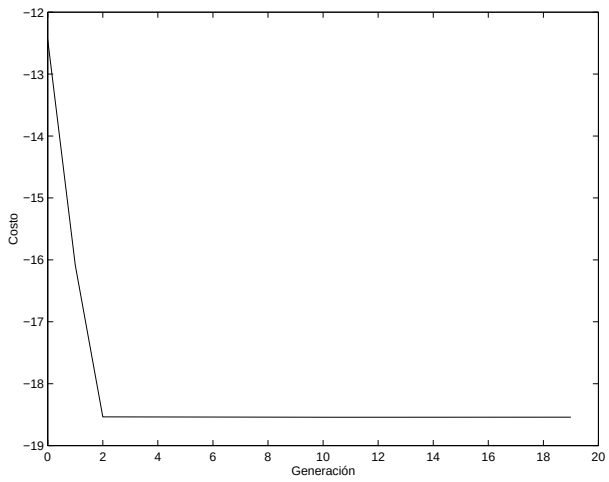


Figura 14. Solución usando AG para la función $f(x) = x_1 seno(4x_1) + 1{,}1 x_2 seno(2x_2)$

donde $[r_i, s_i]$ son las coordenadas espaciales y nuestro cromosoma está dado por el vector $x = [x_0 x_1 x_2 x_3 x_4 x_5]^T$. Con este vector podemos calcular una traslación si $x_T = [1, 0, Tx, 0, 1, Ty]^T$, un escalamiento isotrópico con $x_s = [s, 0, 0, 0, s, 0]^T$, una rotación $x_\phi = [\cos(\phi), -seno(\phi), 0,\ seno(\phi),\ cos(\phi), 0]^T$, etc. Dada una transformación y una imagen en tono de gris $I(r, s)$, la imagen transformada la podemos calcular como $In(r, s) = I(\widehat{r}\,(x)\,, \widehat{s}\,(x))$.

La función de aptitud la podemos plantear como

$$\min_{\Theta} f(x) = \sum_{i=1}^{N} \left[I_1(\widehat{r}_i\,(x)\,, \widehat{y}_i\,(x)) - I_2(x_i, y_i) \right]^2 \tag{14}$$

donde $[\widehat{x}_i\,(x)\,, \widehat{y}_i\,(x)]$ es calculado utilizando la TA de la ecuación (13) e interpolación lineal sobre la imagen origen I_1. En la figura 15, se presentan algunos ejemplos de translación, escalamiento y rotación.

Para este ejemplo se utilizó una transformación correspondiente a una rotación de 20^0 y una traslación de 3 pixeles en cada dirección, lo cual da el vector de transformación $x_0 = [0{,}9396, -0{,}3420, 3, 0{,}3420, 0{,}9396, 3]^T$. Las imágenes origen y destino utilizadas se muestran en la figuras 16(a) y 16(b).

El intervalo de búsqueda está dado en el cuadro 6. En la figura 17(a) se muestra la solución obtenida por el AG y en la figura 17(b) la diferencia de tono de gris entre la imagen calculada y la imagen destino (figuras 17(a) y 16(b). Note la calidad del registro llevado a cabo utilizando esta técnica.

(a) Translación (b) Escalamiento (c) Rotación

Figura 15. Diferentes transformaciones afines

(a) Origen (b) Destino

Figura 16. Imágenes utilizadas para realizar el registro con AG

g	Mín	Máx
x_0	0.5	1.5
x_1	-0.5	0.5
x_2	-10.0	10.0
x_3	-0.5	0.5
x_4	0.5	1.5
x_5	-10.0	10.0

Cuadro 6. Rangos de búsqueda para el problema de registro

10. Programación genética

Cuando usamos algoritmos genéticos para evolucionar un modelo, nos encontramos explorando el espacio de búsqueda de todos los modelos de la forma

(a) Imagen obtenida con el AG

(b) Diferencia entre las imágenes final y destino

Figura 17. Solución obtenida al aplicar AG

especificada. En realidad la búsqueda se lleva a cabo en el espacio de parámetros de ese tipo de modelos. Si los parámetros del modelo son $(a_1, a_2, \ldots, a_p)$, el cromosoma que codifica al modelo puede verse como en la figura 18.

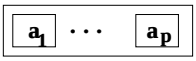


Figura 18. Cromosoma en algoritmos genéticos

El uso de algoritmos genéticos nos permite determinar los parámetros $(a_1, a_2, \ldots, a_p)$ de ese tipo de modelo, que mejor replica el comportamiento de la planta a modelar.

Esta representación no nos permite efectuar una búsqueda a través de varios tipos de modelos, con forma y conjuntos de parámetros diferentes. Sería muy conveniente contar con un procedimiento que no solo determine los parámetros de cierto modelo, sino que nos permita también optimizar la forma del modelo. Programación Genética (PG) es un mecanismo que nos ofrece esa característica. Un cromosoma de PG no solo codifica los parámetros del modelo, sino la forma del mismo. En PG, el cromosoma puede estar codificado en diversas formas. Sin embargo, la mas frecuente es un árbol de expresiones. Ver figura 19.

Un cromosoma en algoritmos genéticos es un vector de parámetros, donde todos sus componentes son números. Un cromosoma en programación genética codifica tanto estructura como parámetros. Para lograr esta representación, programación genética hace uso de dos conjuntos (generalmente definidos por el usuario): $T-$, un conjunto de terminales (v.g. constantes, números, etc.) y $F-$,

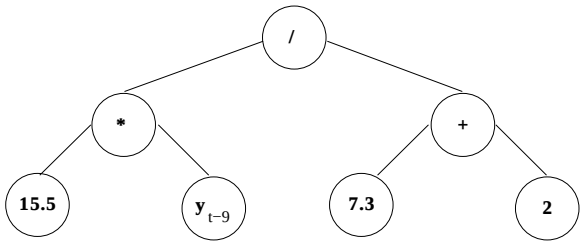


Figura 19. Cromosoma en programación genética

un conjunto de funciones. Las funciones representan nodos internos del árbol y las terminales son usadas en las hojas del mismo.

Si lo que se desea es que el cromosoma represente una expresión, función, un conjunto de funciones, o un programa en un lenguaje de alto nivel, esta representación es la adecuada. Para desarrollar programas en lenguaje ensamblador es mejor considerar el programa como una lista de instrucciones. Para otras aplicaciones se deben considerar estructuras de datos de tipo grafo. Existen otras representaciones, las cuales caen fuera del alcance del presente libro, en donde un cromosoma se representa como una cadena de caracteres. Esta cadena de caracteres (el genotipo) es interpretada posteriormente para extraer las características del individuo que codifica (fenotipo).

10.1. Cruza en programación genética

En la subseccion anterior se presentó la representación de árbol el cromosoma en programación genética. Esta representación del cromosoma tiene ventajas y desventajas. Una ventaja es que es una representación muy popular, con semántica bien conocida.

Una de las desventajas, expuesta por Candida Ferreira en su trabajo *Gene Expression Programming*, es que tanto el genotipo como el fenotipo están incluidos en la misma estructura. Otra desventaja es que la implementación de los operadores genéticos no resulta tan directa como en otras representaciones.

Para realizar la cruza de dos cromosomas (supongamos cruza de un solo punto), debemos elegir un nodo de cada árbol e intercambiarlos en los padres cruzados. Para elegir un nodo aleatorio, se puede etiquetar cada nodo con el número de nodos del sub-árbol del cual es raíz. Este etiquetamiento puede ser generado a la hora de formar un nodo y propagado en las operaciones que combinan o modifican nodos (v.g. cruza y mutación).

Si se cuenta con el tamaño del sub-árbol que encabeza cada nodo, el nodo raíz nos indica el tamaño total del árbol. Para elegir un nodo aleatoriamente, generamos un número aleatorio entre 1 y el tamaño del árbol. Con la información del tamaño de cada sub-arbol y el número aleatorio podemos saber en que sub-

árbol se encuentra el nodo elegido. La implementación de estos procedimientos quedan como ejercicio para el lector.

Para realizar la cruza de un punto entre dos cromosomas representados por medio de un árbol, elegimos un nodo aleatorio de cada uno de ellos. Los nodos elegidos son intercambiados, como se ilustra en las figuras 20 y 21.

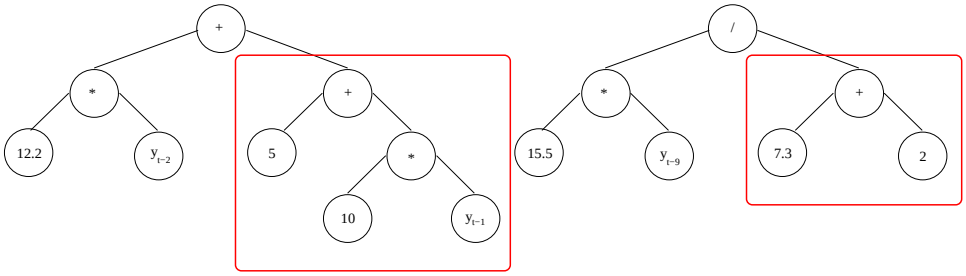


Figura 20. Selección de padres en la cruza

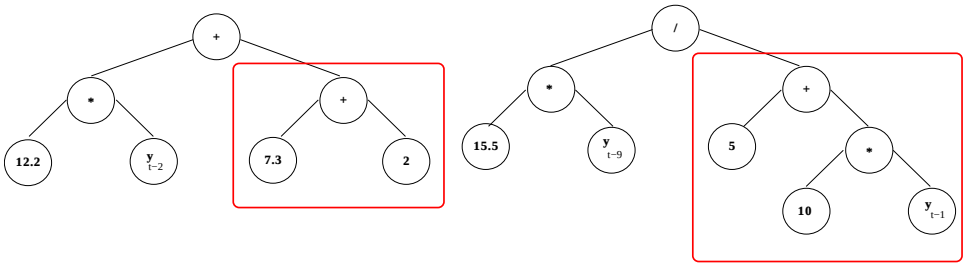


Figura 21. Hijos producidos por la cruza

10.2. Mutación en programación genética

Para implementar el operador genético de mutación se selecciona un nodo al azar y se cambia su valor aleatoriamente. Si el nodo representa un valor terminal, se elige otro terminal. Si el nodo es un nodo interno (función), podemos pensar en reemplazar la función por otra de la misma cardinalidad y dejar los hijos como están, o bien, reemplazarlo por cualquier función y arreglar su cardinalidad. Esto es, borrar los hijos que ya no sean necesarios, en el caso de que la cardinalidad disminuya, o generar un árbol aleatorio para cada uno de los hijos faltantes si la cardinalidad aumenta.

Las figuras 22 y 23 ilustran la aplicación del operador genético de mutación a un árbol de ejemplo.

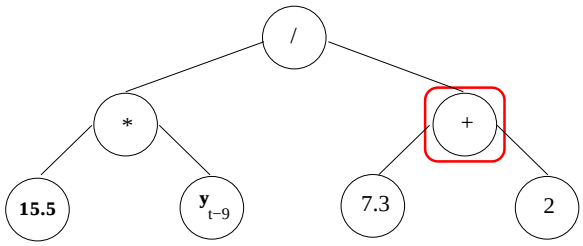


Figura 22. Nodo seleccionado para mutación

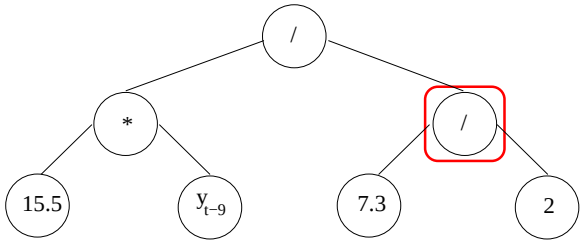


Figura 23. Nodo después de la mutación

10.3. Generación de la población inicial

Para poder efectuar una implementación de programación genética necesitamos definir como se genera la población inicial. Al igual que en algoritmos genéticos, la población inicial es generada aleatoriamente. El tamaño de la población inicial es un parámetro del proceso, el cual debe ser elegido por la persona que está ejecutando los experimentos de búsqueda.

Resta otro detalle que debe ser especificado, las características de los árboles (individuos) a generar aleatoriamente. Para esto existen varias formas de generar árboles aleatorios.

Usando F y T se crea la poblacion inicial. Para crearla, Koza propone dos métodos: el primero es crear árboles balanceados y el segundo es crear árboles desbalanceados. Koza tambien propone que la altura del árbol sea variable. Estas formas de crear la población inicial hace que exista diversidad morfológica en la población inicial.

Para crear un árbol balanceado se siguen los siguientes pasos: Se selecciona aleatoriamente un elemento del conjunto de funciones F siempre y cuando no se halla llegado a la altura del árbol. Cuando se llega a la altura del árbol se seleccionan un elemento del conjunto de terminales T.

El procedimiento para crear un árbol desbalanceado es el siguiente: en la altura 0 se selecciona una función del conjunto F; mientras no llegue a la altura final, se hace la unión de los conjuntos F y T. Se selecciona un elemento de este conjunto; si se trata de una función, se siguen añadiendo elementos a esa rama, de lo contrario se deja la rama como está. Si se llega a la altura máxima del árbol, se seleccionan solo elementos de T.

10.4. El proceso evolutivo

Habiendo definido los componentes principales de cualquier algoritmo evolutivo, el proceso de evolución se lleva a cabo de la misma forma que en algoritmos genéticos. Se genera aleatoriamente un conjunto de individuos (población inicial), los cuales son evaluados mediante la función de aptitud. En cada generación se seleccionan individuos para cruzamiento y se cruzan, generando así los hijos de la siguiente generación. Se seleccionan aleatoriamente individuos para ser mutados.

De todos los individuos de la generación anterior, mas los generados usando los operadores genéticos, se selecciona cuales de ellos formarán la siguiente generación. Para ésto existen varias estrategias: podemos formarlos por aptitud y descartar los más débiles (menos aptos); los hijos pueden reemplazar a los padres; una estrategia elitista solo garantiza que un número de los mejores pasará a la siguiente generación.

Se ejecua el ciclo hasta llegar al número límite de generaciones o lograr algun criterio de convergencia. El mejor individuo se reporta como solución, o como mejor solución encontrada por el proceso.

10.5. Caso de estudio: modelado de empresas

En esta subsección se presenta un ejemplo de aplicación de programación genética. El modelado del comportamiento financiero de una empresa es muy importante en la planeación y el control financiero de la misma. Si contamos con un modelo que nos permita predecir el futuro próximo de la empresa, podremos evitar que ocurran sucesos indeseables, o al menos disminuir sus efectos en la compañía. Por otra parte, podremos tratar que los eventos deseados se presenten con mayor frecuencia o intensidad.

Es conveniente remarcar que el modelado solamente nos permite predecir el comportamiento de la empresa; el resto debe ser realizado por un analista financiero, basado en los datos que el modelo le proporciona.

Una compañía exhibe un comportamiento complejo, el cual puede depender de muchos factores. Algunos de estos factores son internos y otros pueden ser

externos. En esta subsección analizamos la compañía como una entidad aislada, sin tomar en cuenta cualquier interacción con factores externos.

Como una entidad aislada, consideramos a una compañía como un sistema dinámico. Como tal, observamos algunas cantidades que consideramos importantes (de acuerdo a los principios de contabilidad financiera). Estas cantidades o variables constituirán nuestras variables de estado. Esto es, los valores que toman las variables de estado en un instante de tiempo t, representan el estado de la compañía en ese instante.

En el proceso de modelado, se observa una compañía (se registran los valores de las variables de estado) por un periodo de tiempo, produciendo un conjunto de series de tiempo. Basados en esas series de tiempo, la meta es producir modelos que nos permitan predecir el comportamiento de la compañía. En los experimentos realizados hemos registrado la actividad de una compañia comercializadora de cerveza. Se cuenta con datos en un periodo de tiempo de tres años, de los cuales hemos usado dos años como conjunto de entrenamiento y uno como conjunto de validación para los modelos obtenidos.

10.6. Conceptos de contabilidad financiera

Un punto importante en la generación de modelos para una compañía es que cantidades involucrar en los mismos. En contabilidad financiera, dos reportes definen el estado de una compañía: El balance y el estado de resultados. Los cuadros 7 y 8 muestran un ejemplo de estos reportes.

Sucursal No. 1 Estado de resultados Mes de marzo de 200X		
Ventas (ingresos)		120,000.00
Gastos:		
Costo de ventas	100,000.00	
Renta	1,000.00	
Total de gastos		101,000.00
Ingreso neto		19,000.00

Cuadro 7. Estado de resultados

Ambos reportes incluyen varias variables importantes para la empresa. El problema es cuales de esas variables son necesarias y suficientes para definir el estado de una compañía. La respuesta a este problema depende del tipo de compañía en cuestión e incluso del criterio del contador.

De las variables incluídas en estos reportes, podríamos seleccionar variables simples (por ejemplo, activo circulante). Sin embargo, estas variables pueden no contener suficiente información para reflejar de manera precisa la situación de la empresa. Esto da lugar a la introducción de las conocidas razones financieras,

Sucursal No. 1			
Balance			
Marzo 31 de 200X			
Activo		Pasivo y capital	
Caja	42,000.00	Cuentas por pagar	25,000.00
Cuentas por cobrar	90,000.00	Capital	100,000.00
Inventario	10,000.00		
Renta prepagada	2,000.00	Ingresos retenidos	19,000.00
Total activos:	144,000.00	Total pasivo + Capital:	144,000.00

Cuadro 8. Balance

las cuales nos proporcionan medias comparativas. Por ejemplo, cuanto tiene la empresa en relación de cuanto debe.

Para el modelado de esta empresa en particular, hemos decidido usar las siguiente cinco razones financieras[1]:

C = Activo/Pasivo
DTA = Deudas/Activo
ID = Costo de ventas/Inventario promedio
NI = Ingreso neto/Ventas netas
RE = Ingreso neto/Capital promedio

Estas razones son indicadores de la salud de una empresa y serán las variables de estado para conformar nuestros modelos. No existe una fórmula para determinar las variables a usar al modelar una compañía. Para empresas diferentes, o aún un contador diferente, podría aconsejar un conjunto diferente.

En cualquier instante de tiempo, el estado de la compañía será representado por la 5-tupla (C, DTA, ID, NI, RE).

Las series de tiempo obtenidas para realizar el modelado de esta comppañía se muestran en la figura 24

10.7. Modelos ARIMA

Una vez sleccionadas las variables a modelar, necesitamos determinar que tipo de modelos deseamos producir. Basados en la experiencia en modelar otras series de tiempo [9], hemos decidido usar modelos autorregresivos.

Los modelos autorregresivos están compuestos de los valores pasados de la serie de tiempo. Esto es, la predicción de una variablese basa en los valores históricos de la misma. El número de observaciones pasadas usadas en la construcción del modelo se llama tamaño de ventana, k, del modelo. La figura 25 ilustra este concepto.

[1] las siglas vienen del Inglés: C=Current Ratio, DTA=Debts to Total Assets Ratio, ID=Inventory Days Ratio, NI=Net Income Ratio y RE=Return on Equity Ratio

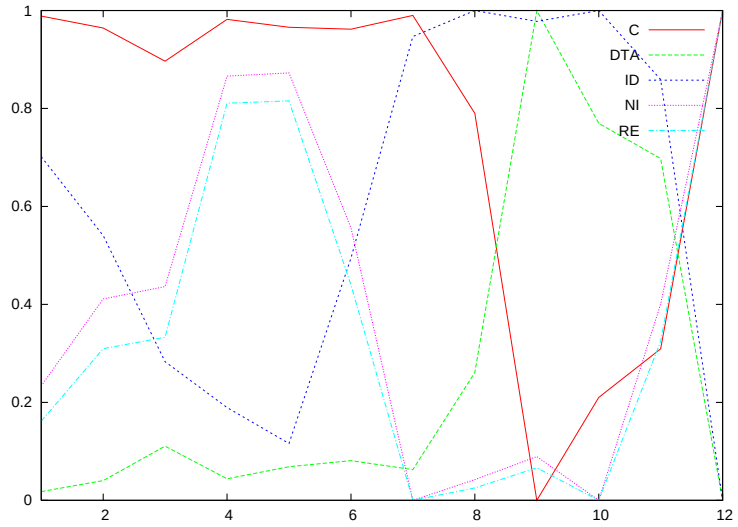


Figura 24. Datos de la compañía a modelar

Básicamente, usaremos dos tipos de modelos. En el primero, el cual llamaremos modelo univariable, una variable es modelada usando solo la serie de tiempo para esa variable. Esto es, cada variable se considera independiente del resto. La ecuación 15 muestra esta estructura.

$$y_{i,n} = f\left(y_{i,n-1}, y_{i,n-2}, \ldots, y_{i,n-k}\right) \tag{15}$$

donde k es el tamaño de la ventana.

La ecuación 15 representa la forma del modelo para predecir la variable $i-$ésima en el instante de tiempo $t = n$. El modelo es una función de los valores pasados de esa misma variable. Note que no estamos restringiendo la forma de tal función. De hecho, la forma de dicha función será determinada por el conjunto de operadores disponibles.

En el segundo tipo de modelos incorporamos la posibilidad de incluir todas las variables en todos los modelos. Este hecho introduce un concepto de interdependencia, donde cada variable puede formar parte del modelo de cualquier otra variable. La ecuación 16 muestra la estructura de un modelo multivariable.

$$\begin{aligned} y_{i,n} = f(&y_{1,n-1}, y_{1,n-2}, \ldots, y_{1,n-k}, \\ &y_{2,n-1}, y_{2,n-2}, \ldots, y_{2,n-k}, \\ &\ldots \\ &y_{m,n-1}, y_{m,n-2}, \ldots, y_{m,n-k}) \end{aligned} \tag{16}$$

donde k es el tamaño de la ventana, $i\epsilon[1, m]$, y m es el número de variables en el modelo.

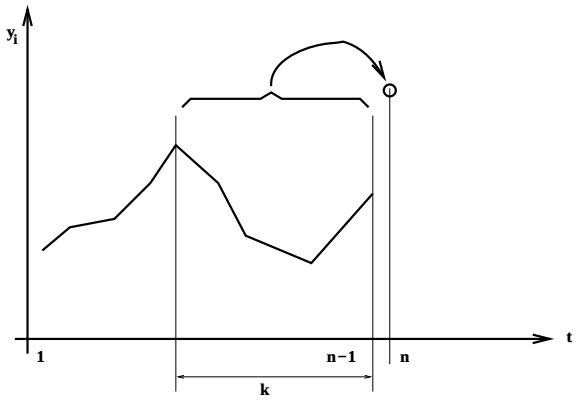


Figura 25. Modelos Arima

10.8. Resultados: modelos univariables

Una vez que se han determinado las variables de estado de la compañía, se procede a colectar los datos. En nuestro ejercicio, se colectaron 24 meses de datos. Estos datos se descompusieron en 20 meses para entrenamiento y 4 para validación. La función de aptitud aplicada a un modelo propuesto será la suma de los cuadrados de os errores. Dado que las magnitudes de las variables de estado son muy diversas, la suma de los cuadrados de los errores también serán muy disímiles, haciendo una comparación entre modelos para diferentes variables muy difícil. Dados esto, decidimos calcular el factor de correlación, el cual estandariza la medida de que tan bien un modelo se ajusta a los datos, produciendo una medida en el intervalo [-1, +1]. Los parámetros utilizados para los experimentos genéticos, se muestran en el cuadro 9.

Parameter	Value
Mutación	0.2
Cruza	0.8
Operadores	$+, -, *$
Terminales	$Real, f(i,j)$
Número de nodos	20
Generaciones	100
Población	100
Número de corridas	10

Cuadro 9. Parámetros para programación genética

Los experimentos fueron ejecutados de acuerdo a las descripciones del párrafo anterior, resultando en los modelos descritos en las ecuaciones 17 to 21.

$$\begin{aligned} C(n) = (0{,}962525217243 - ((C(n-8)C(n-8))(C(n-1) \\ - (0{,}962525217243 - (C(n-10)((C(n-1) - C(n-5)) \\ (C(n-7) + C(n-6)))))))) \end{aligned} \tag{17}$$

$$\begin{aligned} DTA(n) = 4DTA(n-5)(4DTA(n-5) - DTA(n-11)) \\ + DTA(n-1) \end{aligned} \tag{18}$$

$$\begin{aligned} ID(n) = (ID(n-12) + (ID(n-6) * ((ID(n-11) - (ID(n-9) \\ * ID(n-10))) + (ID(n-12) * ((ID(n-11) * ID(n-6)) \\ + (ID(n-6) * ID(n-5))))))) \end{aligned} \tag{19}$$

$$\begin{aligned} NI(n) = (((NI(n-2) * (((NI(n-10) + NI(n-5)) * (NI(n-5) \\ * NI(n-10))) - ((NI(n-3) * NI(n-3)) - NI(n-6)))) \\ + NI(n-11)) * NI(n-12)) \end{aligned} \tag{20}$$

$$\begin{aligned} RE(n) = ((RE(n-12) - (RE(n-7) * (RE(n-3) * RE(n-4)))) \\ (((RE(n-12) + RE(n-2)) * RE(n-11)) + RE(n-6))) \end{aligned} \tag{21}$$

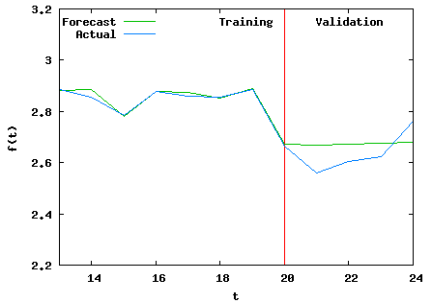
Los coeficientes de correlación se muestran en el cuadro 10. La figura 26 muestra las series de tiempo y los resultados de los modelos univariables evolucionados mediante programacion genética. En lo que resta del capítulo, usaremos la notación $C(j)$ para denotar la función $C(n-j)$. Es decir, la $j-$ésima observación anterior al punto n.

El cuadro 10 muestra los coeficientes de correlación obtenidos en la aplicación de los modelos univariables a las diferentes series de tiempo. En este cuadro (y en las sucesivas), r representa el coeficiente de correlación para el periodo de entrenamiento y r_v el coeficiente de correlación para el periodo de validación.

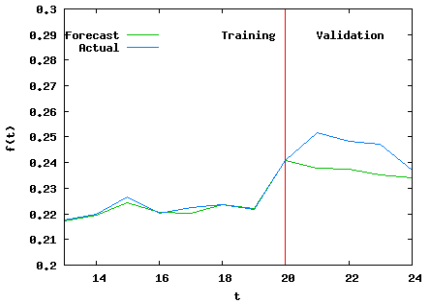
Variable	Entrenamiento r	Validación r_v
C	0,970	−0,844
DTA	0,966	−0,887
ID	0,997	0,809
NI	0,995	0,394
RE	0,995	0,791

Cuadro 10. Coeficientes de correlación para modelos univariables

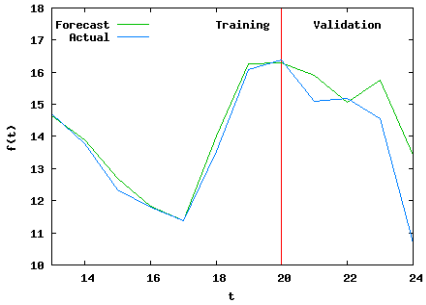
Del cuadro anterior se observa que los modelos se ajustan muy bien a los datos en el periodo de entrenamiento, pero no son muy buenos en el periodo de predicción (validación). Se observa, por los coeficientes de correlación, que las



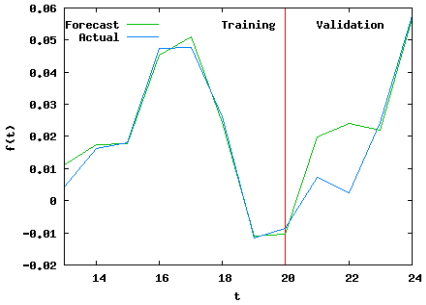
(a) C



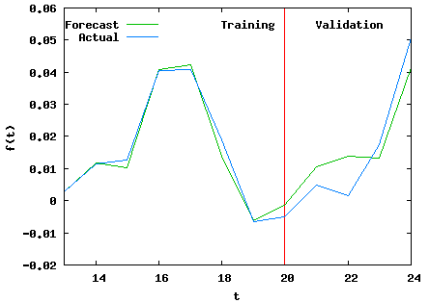
(b) DTA



(c) ID



(d) NI



(e) RE

Figura 26. Modelos univariables para las demás variables de estado

variables C y DTA son más difíciles de modelar que las otras tres. Pese a que sus modelos son mas complejos, sus coeficientes de correlación son mas bajos.

10.9. Resultados: modelos multivariables

Dados los resultados de la sección anterior, se decidió realizar experimentos con modelos multivariables. Los modelos obtenidos para las cinco variables de estado se muestran en las ecuaciones 22 a 26.

$$\begin{aligned} C(n) = {} & (C(n-4) - C(n-1))ID(n-10)RE(n-5) \\ & + C(n-4)C(n-7)(ID(n-5) - ID(n-10))ID(n-10) \\ & + C(n-1) \end{aligned} \tag{22}$$

$$\begin{aligned} DTA(n) = {} & (((RE(n-4) * NI(n-12)) * (NI(n-9) \\ & - (DTA(n-11) - C(n-10)))) * C(n-12)) \end{aligned} \tag{23}$$

$$\begin{aligned} ID(n) = {} & (C(n-4) - ((((RE(n-12) - DTA(n-5)) \\ & NI(n-1)) * ID(n-7)) + (RE(n-12) * ID(n-7)))) \end{aligned} \tag{24}$$

$$\begin{aligned} NI(n) = {} & ((((((RE(n-11) + DTA(n-7)) - (DTA(n-9) \\ & * NI(n-10))) + DTA(n-7)) - (DTA(n-10) * NI(n-10))) \\ & * C(n-3)) + DTA(n-7)) \end{aligned} \tag{25}$$

$$\begin{aligned} RE(n) = {} & (((ID(n-8) * RE(n-12)) * ID(n-3)) + (RE(n-12) \\ & * ID(n-7))) \end{aligned} \tag{26}$$

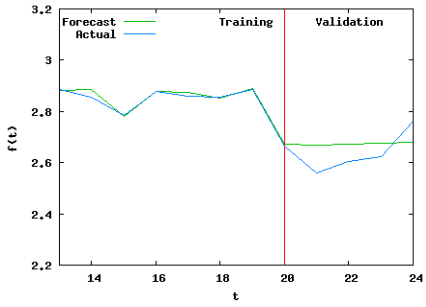
Los coeficientes de correlación obtenidos con tales modelos se muestran en el cuadro 11. La figura 27 muestra las variables de estado con sus observaciones y las aproximaciones logradas por los modelos multivariables.

El cuadro 11 muestra los coeficientes de correlación obtenidos en la aplicación de los modelos multivariables a las diferentes series de tiempo.

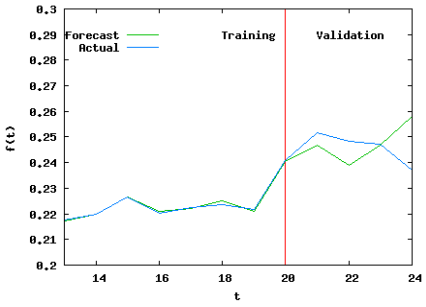
Variable	Entrenamiento r	Validación r_v
C	0,998	0,889
DTA	0,836	0,666
ID	0,926	0,922
NI	0,986	−0,555
RE	0,953	−0,235

Cuadro 11. Coeficientes de correlación para modelos multivariables

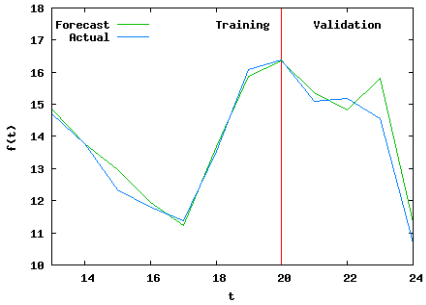
De los resultados de este experimento se puede observar que los modelos aun son complejos y los ajustes nos son los deseados. Una de las razones por las que



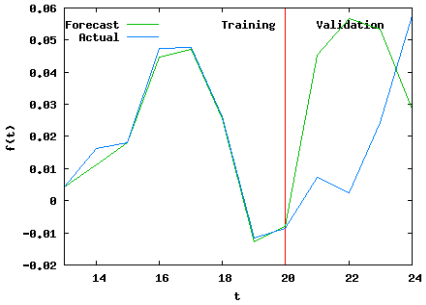
(a) C



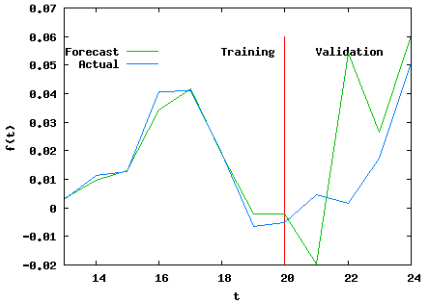
(b) DTA



(c) ID



(d) NI



(e) RE

Figura 27. Modelos multivariables para las variables de estado

creemos que los modelos no son aceptables es el sobreentrenamiento. Este efecto se presenta cuando los modelos se depuran tanto en los datos de entrenamiento y llegan a modelarlos a tal detalle, que pierden toda posibilidad de generalización y, por consiguiente, de pronóstico.

10.10. Reducción del conjunto de funciones

Una manera de evitar este sobreentrenamiento es reducir el conjunto de terminales. Para nuestros experimentos, las terminales son las referencias a datos previos de las diferentes variables. La idea es ejecutar una corrida preliminar, digamos de 10 generaciones, en la cual se cuentan las apariciones de las terminales y se eliminan todas aquellas terminales que no aparezcan de manera consistente en modelos aceptables. Esta idea es similar a lo que los estadistas hacen para seleccionar observaciones estadísticamente significativas, usando el coeficiente de autocorrelación. El umbral de discriminación usado fue la media aritmética. Si una terminal aparece menos veces que la media, se descarta. La figura 28 muestra el perfil de uso de funciones, junto con la media de uso. Para este caso, solo las funciones $C(n), C(n-1)$ y $C(n-4)$ serán usadas en el proceso evolutivo.

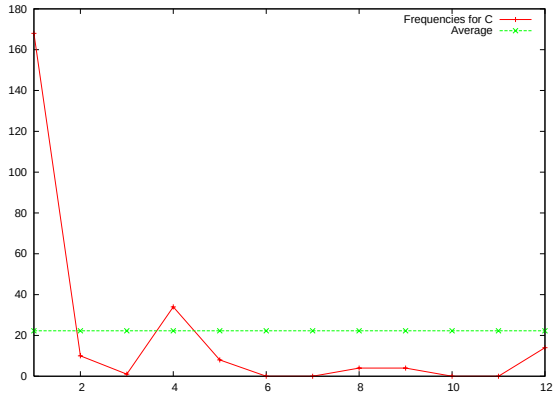


Figura 28. Frecuencias de uso de funciones

Al reducir el conjunto de terminales, se simplifican los modelos, se reduce el espacio de búsqueda y se evita el sobreentrenamiento. Las ecuaciones 27 a 31 muestran los modelos producidos usando la estrategia de reducción de funciones.

$$C(n) = ID(4) + NI(11) \tag{27}$$

$$\begin{aligned} DTA(n) = (DTA(9) + ((CA(9) - DTA(6)) * (CA(3) - NI(6)))) \\ * ((CA(10) * RE(5)) * ((DTA(6) + CA(10)) - DTA(7))) \end{aligned} \tag{28}$$

$$ID(n) = CA(5) - (((DTA(11) + RE(5)) * NI(4)) * (RE(5) + ((DTA(1) + ID(7)) * (RE(5) + DTA(11)))))) \quad (29)$$

$$NI(n) = NI(11) - ((ID(10) * ID(5)) + -0{,}08576300044) \quad (30)$$

$$RE(n) = ((RE(11) + (NI(10) - CA(9))) + (NI(10) * RE(1))) * CA(9) \quad (31)$$

El cuadro 12 muestra los coeficientes de correlación para estos modelos. La figura 29 muestra las variables de estado con sus observaciones y las aproximaciones logradas por los modelos multivariables usando la estrategia de reducción del conjunto de terminales.

Variable	Single-variable		multivariable	
	r	r_v	r	r_v
C	0,900	−0,555	0,795	0,976
DTA	0,838	0,954	0,768	0,604
ID	0,993	0,966	0,946	−0,683
NI	0,979	0,956	0,994	0,939
RE	0,993	0,448	0,981	0,821

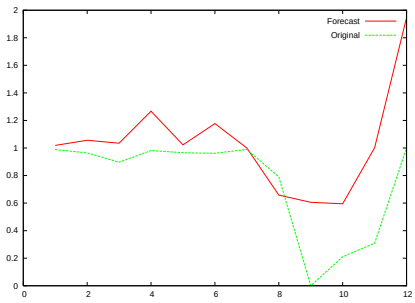
Cuadro 12. Coeficientes de correlación para modelos multivariables usando la estrategia de reducción del conjunto de terminales

10.11. Comparación final

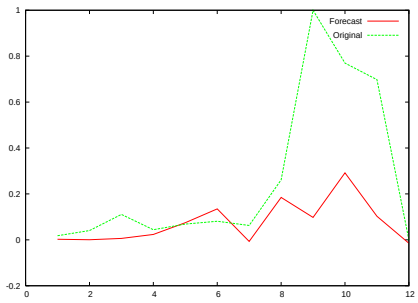
En los experimentos realizados con ambos tipos de modelos, se notó una falta de capacidad para generalizar y predecir en todos ellos. Aún cuando los coeficientes de correlación eran por encima de 0.99 en el conjunto de entrenamiento. Esta era una indicación clara de sobreentrenamiento.

Var.	Uni				Multi			
	SRF		CRF		SRF		CRF	
	r	r_v	r	r_v	r	r_v	r	r_v
C	0,970	−0,844	0,900	−0,555	0,998	0,889	0,795	0,976
DTA	0,966	−0,887	0,838	0,954	0,836	0,666	0,768	0,604
ID	0,997	0,809	0,993	0,996	0,926	0,922	0,946	−0,683
NI	0,995	0,394	0,979	0,956	0,986	−0,555	0,994	0,939
RE	0,995	0,791	0,993	0,448	0,953	−0,235	0,981	0,821

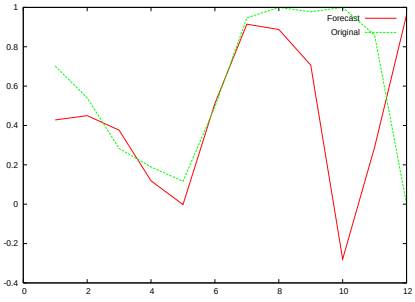
Cuadro 13. Comparación de los coeficientes de correlación obtenidos para los diferentes modelos evolucionados



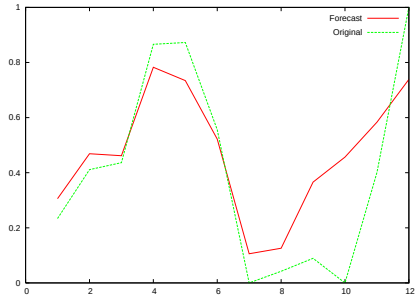
(a) C



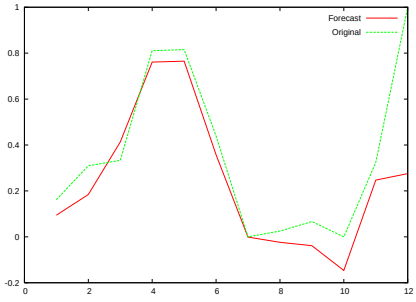
(b) DTA



(c) ID



(d) NI



(e) RE

Figura 29. Series de tiempo para modelos multivariables usando la estrategia de reducción del conjunto de terminales

Para reducir el sobreentrenamiento, la complejidad de los modelos y el espacio de búsqueda, se presenta una estrategia que hemos llamado *reducción del conjunto de terminales*. En esta estrategia, el conjunto de terminales se reduce a aquellas que aparecen frecuentemente en modelos buenos en una corrida preliminar. La estrategia propuesta dio como resultado la producción de mejores modelos, en el sentido de que son mejores predictores.

Aún utilizando la estrategia de reducción del conjunto de terminales, hubo casos en donde el coeficiente de correlación era aún negativo. En estos casos, el conjunto de validación exhibía características no presentes en el conjunto de entrenamiento. Este fenómeno es equivalente a cambios repentinos o inesperados en la compañía. Estos cambios pueden ser resultado de cambios en la economía del país o alguna otra perturbación externa impredecible. Los autores del presente trabajo creemos que con series de tiempo suficientemente grandes, que exhiban características representativas del comportamiento de la compañía, los coeficientes de correlación serían positivos en todos los casos.

11. Conclusiones

En este capítulo se ha presentado un tutorial breve de computación evolutiva. La computación evolutiva, al igual que muchas otras áreas de Inteligencia Artificial, se les conoce como técnicas bio-inspiradas. Computación evolutiva toma sus raíces del principio de evolución de Darwin y de la supervivencia del más fuerte. Mediante estos principios, la computación evolutiva determina el individuo más apto del espacio de búsqueda. La aptitud de un individuo cualesquiera está definida por una función, conocida como función de aptitud o desempeño. Por medio de este proceso, al determinar el individuo con mejor función de aptitud (o una muy buena aproximación), se está llevando a cabo una optimización de la función de aptitud. Las técnicas de computación evolutiva son, entonces, optimizadores.

Los algoritmos evolutivos se pueden dividir, a groso modo, en dos vertientes principales: algoritmos genéticos y programación genética.

En los algoritmos genéticos, todos los individuos tienen la misma estructura. Un individuo se codifica en un cromosoma parametrizando sus características. Esta codificación puede tomar varias formas, dependiendo de las características de la estructura de los individuos a optimizar. Si consideramos el cromosoma que representa un individuo, como un vector de tamaño n, entonces el proceso de optimización se llevará a cabo en un espacio $n-$dimensional.

En programación genética la codificación de los individuos incluye su estructura. Es decir, no suponemos uniformidad en la estructura, sino que damos la libertad de que la estructura misma evolucione dentro del proceso de optimización. Con ésto se le da una capacidad mucho mayor al sistema; sin embargo, como es bien sabido, nada es gratis. El precio a pagar es un espacio de búsqueda mucho mayor, con la consiguiente degradación de la eficiencia.

Existe un número muy grande de aplicaciones para esta área, ya sea con algoritmos genéticos o programación genética. Las aplicaciones van desde el problema del agente viajero, modelado para control en ingeniería eléctrica, determinación de transformaciones en procesamiento de señales, modelado de sistemas dinámicos, etc. La computación evolutiva ha sido aplicada incluso a ayudar otras áreas de Inteligencia Artificial; por ejemplo, ha sido aplicada para la optimización de la estructura y parámetros de redes neuronales artificiales.

12. Ejercicios propuestos

1. Dada una variable x en una sola dimensión con valores en el rango $[0, 10]$ ¿Cuál sería el error de codificación binaria de $x = 2{,}13$ si se utilizan 2, 4, 8 y 16 bits? ¿Cuál sería el error de codificación real?
2. Dado el conjunto de tres puntos $[x_i, y_i] = [0{,}9, 1{,}1], [2, 1{,}9], [3{,}1, 3{,}5]$ hay una linea recta que pasa por estos puntos dada como $y_i = mx_i + b$ donde m es la pendiente y b el cruce por cero de la recta. Programar un algoritmo genético con codificación real para calcular estos dos parámetros. Utilice como intervalo de búsqueda $m = [0{,}5, 2]$ y $b = [-1, 1]$. La función de aptitud a evaluar es $f(m, b) = \sum_{i=1}^{3} [mx_i + b - y_i]^2$
3. Repita el ejemplo anterior utilizando solamente búsqueda aleatoria. Cuente las veces que evalúa la función de aptitud. ¿Cuál algoritmo hace más evaluaciones?
4. Considere el conjunto de datos siguiente:
   [(-10., 287.3), (-9.33333, 249.967), (-8.66667, 215.3), (-8., 183.3),
   (-7.33333, 153.967), (-6.66667, 127.3), (-6., 103.3), (-5.33333, 81.9667),
   (-4.66667, 63.3), (-4., 47.3), (-3.33333, 33.9667), (-2.66667, 23.3),
   (-2., 15.3), (-1.33333, 9.96667), (-0.666667, 7.3), (0., 7.3),
   (0.666667, 9.96667), (1.33333, 15.3), (2., 23.3), (2.66667, 33.9667),
   (3.33333, 47.3), (4., 63.3), (4.66667, 81.9667), (5.33333, 103.3),
   (6., 127.3), (6.66667, 153.967), (7.33333, 183.3), (8., 215.3),
   (8.66667, 249.967), (9.33333, 287.3), (10., 327.3)]
   Estos datos corresponden a pares de coordenadas, en un sistema cartesiano; es decir, parejas (x, y).
   Al proceso de determinar una función matemática que describa el comportamiento de un conjunto de datos se le llama modelado de datos. Modelando un conjunto de datos podemos interpolar, extrapolar, y en general predecir el comportamiento del fenómeno observado.
   En este ejercicio se le pide al alumno modelar el conjunto de datos anterior. Para que sea más claro, el proceso de modelado se puede llevar a cabo siguiendo los pasos descritos a continuación:
   a) Grafique los datos.
   b) Usando sus conocimientos matemáticos, trate de identificar a que función corresponden.

   c) Una vez determinada la forma de la función, diseñe la codificación de un cromosoma que represente a un individuo. Ésto es, un cromosma representará los parámetos de una función que modele los datos.
   d) Ejecute un algoritmo genético que determine la función óptima (la que más se apegue a los datos proporcionados).
   e) Una función de aptitud a usar en el algoritmo genético puede ser la suma de los cuadrados de los errores.

5. Observe que, usando algoritmos genéticos, el proceso de modelado involucra al analista (estadista, ingeniero, o científico). Usando programación genética, el analista solamente le proporciona al sistema el conjunto de funciones a utilizar y el sistema realiza la identificación cualitativa (la forma de la función) y cuantitativa (determinación de los valores óptimos de los parámetros).
   Repita el ejercicio anterior usando programación genética. Se sugiere usar el siguiente conjunto de funciones: +, -, *, /, Sin, Cos, Tan, Exp
   Compare los resultados con los del ejercicio anterior.
6. Para los dos ejercicios anteriores, produzca gráficas de convergencia del proceso. Una gráfica de convergencia presenta el valor de aptitud (ordenadas) del mejor individuo de cada generación (abscisas).
7. Los datos siguientes representan las mismas mediciones anteriores, pero con una cierta cantidad de ruido adicionada a ellos:
   [(-10., 282.597), (-9.33333, 247.426), (-8.66667, 206.939), (-8., 179.601), (-7.33333, 162.051), (-6.66667, 126.241), (-6., 99.8905), (-5.33333, 87.0212), (-4.66667, 68.4079), (-4., 38.7809), (-3.33333, 29.3201), (-2.66667, 26.1631), (-2., 17.5221), (-1.33333, 17.4443), (-0.666667, 1.46924), (0., 7.63643), (0.666667, 7.59827), (1.33333, 7.65406), (2., 25.9058), (2.66667, 38.127), (3.33333, 41.4962), (4., 62.6206), (4.66667, 87.8123), (5.33333, 104.952), (6., 136.2), (6.66667, 145.828), (7.33333, 187.507), (8., 210.651), (8.66667, 240.782), (9.33333, 290.22), (10., 324.916)]
   Repita los ejercicios 4, 5 y 6 con ellos y conteste las siguientes preguntas:
   a) ¿Qué método es mejor, AGs o PG en cada uno de los casos?
   b) ¿Por qué?
   c) ¿Hace la adición de ruido que cambien los resultados?
8. Los siguientes datos representan una función tridimensional; representan tripletas de la forma (x, y, z). Repita los ejercicios 4, 5 y 6 con ellos.
   [(0., 0., 0.), (0., 1., 1.00023), (0., 2., -1.66497), (0., 3., -0.922071), (0., 4., 4.35318), (0., 5., -2.99212), (0., 6., -3.54138), (0., 7., 7.62768), (0., 8., -2.53355), (0., 9., -7.43477), (0., 10., 10.0424), (1., 0., -0.756802), (1., 1., 0.243425), (1., 2., -2.42177), (1., 3., -1.67887), (1., 4., 3.59637), (1., 5., -3.74892), (1., 6., -4.29818), (1., 7., 6.87087), (1., 8., -3.29035), (1., 9., -8.19158), (1., 10., 9.2856), (2., 0., 1.97872), (2., 1., 2.97894), (2., 2., 0.313751), (2., 3., 1.05665), (2., 4., 6.33189), (2., 5., -1.0134), (2., 6., -1.56266), (2., 7., 9.60639), (2., 8., -0.554833), (2., 9., -5.45606), (2., 10., 12.0211), (3., 0., -1.60972), (3., 1., -0.609492), (3., 2., -3.27468), (3., 3., -2.53179), (3., 4., 2.74346), (3., 5., -4.60183), (3., 6., -5.1511),

(3., 7., 6.01796), (3., 8., -4.14327), (3., 9., -9.04449), (3., 10., 8.43268), (4., 0., -1.15161), (4., 1., -0.151386), (4., 2., -2.81658), (4., 3., -2.07368), (4., 4., 3.20156), (4., 5., -4.14373), (4., 6., -4.69299), (4., 7., 6.47606), (4., 8., -3.68516), (4., 9., -8.58639), (4., 10., 8.89078), (5., 0., 4.56473), (5., 1., 5.56495), (5., 2., 2.89976), (5., 3., 3.64266), (5., 4., 8.9179), (5., 5., 1.57261), (5., 6., 1.02334), (5., 7., 12.1924), (5., 8., 2.03118), (5., 9., -2.87005), (5., 10., 14.6071), (6., 0., -5.43347), (6., 1., -4.43324), (6., 2., -7.09844), (6., 3., -6.35554), (6., 4., -1.08029), (6., 5., -8.42559), (6., 6., -8.97485), (6., 7., 2.19421), (6., 8., -7.96702), (6., 9., -12.8682), (6., 10., 4.60893), (7., 0., 1.89634), (7., 1., 2.89657), (7., 2., 0.231375), (7., 3., 0.974269), (7., 4., 6.24952), (7., 5., -1.09578), (7., 6., -1.64504), (7., 7., 9.52402), (7., 8., -0.637209), (7., 9., -5.53843), (7., 10., 11.9387), (8., 0., 4.41141), (8., 1., 5.41164), (8., 2., 2.74645), (8., 3., 3.48934), (8., 4., 8.76459), (8., 5., 1.4193), (8., 6., 0.870032), (8., 7., 12.0391), (8., 8., 1.87786), (8., 9., -3.02336), (8., 10., 14.4538), (9., 0., -8.92601), (9., 1., -7.92578), (9., 2., -10.591), (9., 3., -9.84808), (9., 4., -4.57283), (9., 5., -11.9181), (9., 6., -12.4674), (9., 7., -1.29833), (9., 8., -11.4596), (9., 9., -16.3608), (9., 10., 1.11639), (10., 0., 7.45113), (10., 1., 8.45136), (10., 2., 5.78617), (10., 3., 6.52906), (10., 4., 11.8043), (10., 5., 4.45902), (10., 6., 3.90975), (10., 7., 15.0788), (10., 8., 4.91758), (10., 9., 0.0163579), (10., 10., 17.4935)]

Para los ejercicios anteriores, pueden conseguir sistemas, tanto de algoritmos genéticos como de programación genética, basadas en java en:
http://jgap.sourceforge.net/
http://sourceforge.net/projects/jrgp
http://jgprog.sourceforge.net

13. Referencias

En esta sección presentamos un conjunto de recursos que pueden llegar a ser útiles al lector y/o estudiantes: libros, revistas, conferencias y sitios Web. Tenga en cuenta que los sitios de web son volátiles; algunos desaparecen, otros aparecen y otros cambian de dirección. Si no encuentra alguno de ellos, utilice su buscador preferido de internet para encontrar palabras o frases relacionadas. De forma parecida, para acceder a los sitios de las conferencias, busque los sitios del año en curso (o del anterior, dado que algunas conferencias son bienales). Fuentes son ordenadas alfabéticamente.

13.1. Libros

- Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. Genetic Programming: An Introduction. On the Automatic Evolution of Computer Programs and Its Applications. Morgan Kaufmann Series in Artificial Intelligence. 1997

- J. E. Dennis and Robert B. Schnabel. Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Society for Industrial and Applied Mathematics. SIAM. Philadelphia. 1996.
- David E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley. 1989.
- Richard Hartley and Andrew Zisserman. Multiple View Geometry in Computer Vision. Cambridge University Press. University of Oxford, UK. Second Edition. 2003.
- Randy L. Haupt and Sue Ellen Haupt. Practical Genetic Algorithms. Wiley Interscience. 2004.
- Christian Jacob. Illustrating Evolutionary Computation with Mathematica. Morgan Kaufmann Series in Artificial Intelligence. 2001.
- Bernd Jahne. Digital Image Processing. Springer. New York, N.Y. 2002.
- John R. Koza. Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992.
- John R. Koza. Genetic Programming II: Automatic Discovery of Reusable Programs, MIT Press, May 1994.
- Alan V Oppenheim. Seales y Sistemas. Pearson Education. Mxico. 1997.
- J. R. Parker. Algorithms for Image Processing and Computer Vision. Wiley. USA. 1997.
- Bernard Sklar. Digital Comunication. Prentice Hall. 2000.
- H P Sue. Analisis de Fourier. Addison Wesley Iberoamericana. USA. 1987.
- Gilbert Strang. Linear Algebra and Its Applications. Thomson Learning. Tercera Edición. 1988.
- Ronald E. Walpole and R H Myers. Probabilidad y Estadistica para Ingenieros. Editorial Interamericana. Mxico D.F. 1987.
- Paul F Whelan and Derek Molloy. Machine Vision Algorithms in java. Springer. Great Britain. 2001.

13.2. Revistas

- Evolutionary Computation. MIT Press.
- Genetic Programming and Evolvable Machines. Springer.
- IEEE Transactions on Evolutionary Computation. IEEE.

13.3. Conferencias

- European Conference on Genetic Programming (EuroGP 2007). Valencia, Spain, 11th April 2007, 3 days, Deadline: November 10, 2006.
- Evo* 2007. Valencia, España. 11-13 de Abril, 2007.
- Foundations of Genetic Algorithms (FOGA 2007). Mexico City, Mexico, 07th January 2007.
- Genetic and Evolutionary Computation Conference (GECCO 2007). London, UK, 07th July 2007, 5 days, Deadline: 17th January 2007.
- IEEE Congress on Evolutionary Computation (CEC 2007). Singapore, 25th September 2007, 4 days, Deadline: 15th March 2007.

13.4. Sitios Web

- Bibliografía de programación genética.
  http://liinwww.ira.uka.de/bibliography/Ai/genetic.programming.html
- EC Conferences.
  http://www.genetic-programming.org/gpotherconfs.html
- Evolutionary Computation Conferences.
  http://www-users.york.ac.uk/~mal503/conference.htm
- Evolvica.
  http://pages.cpsc.ucalgary.ca/ jacob/Evolvica/
- Genetic Programming Engine.
  http://gpe.sourceforge.net/
- GPLAB.
  http://gplab.sourceforge.net/
- International Society for Genetic and Evolutionary Computation.
  http://www.isgec.org/
- Sitio oficial de programación genética.
  http://www.genetic-programming.org/

Capítulo 7.

Paradigmas emergentes en algoritmos bio-inspirados

1. Introducción a las heurísticas

Los problemas de búsqueda se encuentran inmersos en muchas áreas de las actividades científicas y tecnológicas contemporáneas. Los problemas del mundo real plantean modelos cuya complejidad, generada por diferentes factores, deriva en espacios de búsqueda muy grandes y difíciles de recorrer, siempre con la intención de encontrar aquella solución que cumpla con los requerimientos de quien trata de resolver el problema.

1.1. Diferentes problemas por resolver

El diseño de algoritmos de búsqueda es tan diverso como lo es el conjunto de problemas que requieren un proceso de este tipo. Para efectos de este capítulo se centrará la discusión de uno de ellos, donde los algoritmos bio-inspirados han sido aplicados ampliamente. A continuación se presenta una definición un problema de búsqueda, así como el concepto específico de problema de optimización.

Búsqueda general

En este problema, se busca una solución que satisfaga un cierto conjunto de condiciones pre-establecidas. Un ejemplo concreto es el problema de satisfacción de restricciones. Una instancia encontrada en el mundo real es la definición de horarios para grupos en una escuela. Se desea encontrar una solución que asigne a cada grupo sus clases del periodo (trimestre, semestre), así como salones y profesores, de manera que no exista ningún empalme en las asignaciones, por ejemplo, que no haya dos grupos tomando una materia en el mismo salón a la misma hora.

Optimización

Puede verse como un caso particular del problema de búsqueda general. Es aquel donde se pretende encontrar una solución o conjunto de soluciones que maximice o minimice una cierta medida de calidad, conocida como función objetivo. Muchas veces, además de que la solución sea la que obtenga el valor óptimo para la función objetivo, debe también satisfacer un conjunto de restricciones asociadas al problema. Un ejemplo puede ser el diseño de una pieza mecánica, de la cual se quiere encontrar su costo mínimo de construcción, el cual depende de variables como el largo y el ancho de la pieza, así como el grosor del material. Además, pueden existir restricciones de espacio, de manera que la pieza no puede ser ni muy grande ni muy chica, pues debe incorporarse a un mecanismo más grande.

A su vez, los problemas de optimización se pueden dividir en dos grandes clases:

- **Problemas de optimización numérica**: Se busca un conjunto de valores para las variables del problema de manera que al sustituirse en la función objetivo se maximice o minimice el valor de esta función. Un ejemplo de este problema puede ser el diseño de la pieza mecánica descrito anteriormente. Aquí se busca el conjunto de valores para el largo, el ancho y el grosor del material de la pieza mecánica, de manera que satisfaga las restricciones de espacio y se minimice el costo de construcción.
- **Problemas de optimización combinatoria**: Se busca encontrar el orden de un conjunto de elementos de manera que se maximice o minimice el valor de la función objetivo. Un ejemplo de este tipo de problemas es el del agente viajero, que debe recorrer un conjunto de ciudades, pasando por ellas sólo una vez, de manera que regrese a su punto de salida y se minimice el costo del viaje. Aquí se desea encontrar el orden óptimo de recorrido de las ciudades.

Para efectos de este capítulo, la discusión se centrará en el problema de optimización numérica.

1.2. Métodos clásicos

Existe un área dentro de las matemáticas, dedicada al estudio, entre otros temas, de técnicas para la resolución de problemas de optimización y se conoce como Investigación de Operaciones (IO). La IO plantea la determinación del mejor curso de acción de un problema de decisión con la restricción de productos limitados [1]. Para el problema de optimización numérica se tiene un conjunto de técnicas, las cuales se pueden clasificar de la siguiente manera [2]:

1. Métodos tradicionales
   - Técnicas de variable simple: Se subdividen en métodos directos (utilizan la función a optimizar para guiar la búsqueda) y de gradiente (utilizan información del gradiente para guiar la búsqueda).

- Técnicas multivariable: También se subdividen en métodos directos y de gradiente y realizan búsquedas en múltiples dimensiones, valiéndose, en algunos casos, de técnicas de variable simple.
- Técnicas para problemas con restricciones: Realizan búsquedas en espacios restringidos, usualmente utilizando técnicas multivariable y/o de variable simple de manera iterativa.
- Técnicas especializadas: Métodos para problemas particulares como programación entera (variables que sólo toman valores enteros).

2. Métodos no tradicionales. Métodos que incorporan conceptos heurísticos para mejorar la búsqueda.

Este capítulo se centrará precisamente en dos propuestas novedosas que forman parte de los métodos no tradicionales para resolver problemas de optimización numérica.

1.3. ¿Por qué utilizar heurísticas?

Como se ha detallado previamente en este capítulo, existe una gama de técnicas para resolver problemas de búsqueda y/o optimización. Surge de manera natural la pregunta ¿Qué novedad o conveniencia ofrecen las heurísticas para resolver problemas de optimización? Esta pregunta tiene sentido, pues teniendo un arsenal de métodos de solución, pareciese que no se tiene cabida para métodos "alternativos". Sin embargo, los problemas del mundo real suelen tener características tan particulares que hacen difícil (y a veces imposible) el aplicar los métodos tradicionales en su resolución. Por otro lado, existen problemas donde estos métodos si pueden ser aplicados, sin embargo los resultados y/o el tiempo requerido para obtener una solución no son los esperados por quien resuelve el problema. Michalewicz y Fogel [3] distinguen las siguientes características que hacen difícil de resolver un problema de búsqueda del mundo real:

- El número de posibles soluciones (espacio de búsqueda) es demasiado grande.
- El problema es tan complicado que, con la intención de obtener alguna solución, se deben utilizar modelos simplificados del mismo y por ende, la solución es poco útil.
- La función de evaluación que describe la calidad de cada solución en el espacio de búsqueda varía con el tiempo y/o tiene ruido.
- Las soluciones posibles están altamente restringidas, lo cual dificulta incluso la generación de al menos una solución factible (es decir, que cumpla con las restricciones del problema).

Las técnicas tradicionales pueden garantizar la convergencia al óptimo global (la mejor de todas las posibles soluciones) para algunos problemas que cumplen con requerimientos muy particulares del método, sin embargo, el costo computacional puede ser muy alto o bien pueden ser sensibles a quedarse estancadas en soluciones óptimas locales (soluciones buenas en una cierta zona del espacio

de búsqueda, pero que no son la mejores que el óptimo global). Las técnicas heurísticas (no tradicionales según la clasificación incluida en este capítulo), no pueden garantizar el encontrar la mejor solución de un problema, sin embargo, encontrarán una solución aceptable en un tiempo razonable. De ahí la importancia de su estudio actualmente. De hecho, existe en la literatura especializada una gran cantidad de publicaciones donde se destacan casos de éxito del uso de técnicas no tradicionales en la solución de problemas complejos [4,5,6].

1.4. Heurísticas no bio-inspiradas

Este capítulo se centra en heurísticas inspiradas en procesos encontrados en la naturaleza. Sin embargo, existen también técnicas que se basan en otro tipo de mecanismos que les permiten evadir las soluciones óptimas locales, que tienen un costo computacional razonable y que no requieren que el problema se ajuste a requerimientos por parte de la heurística. Una de ellas es el recocido simulado [7], que basa su funcionamiento en el proceso físico de tratamiento térmico de materiales, donde primero el material es calentado a altas temperaturas y después, mediante una cuidadosa variación de temperatura, es enfriado. La intención es que el material obtenga características particulares en el proceso. Esta idea ha sido utilizada para resolver problemas de optimización (numérica sobre todo) agregando una temperatura inicial y un horario de enfriamiento que permite disminuirla de manera gradual con la intención de permitir al algoritmo de búsqueda el seleccionar una nueva solución, comparada con una solución previa, aunque no mejore el valor de calidad de esta última. Con ello, se agrega un aspecto aleatorio al proceso de búsqueda al permitir al proceso el seleccionar una nueva solución a pesar de que no mejora a la solución previamente encontrada. Este comportamiento ayuda a evadir óptimos locales. Otra heurística, que en su propuesta original no agrega aspectos aleatorios, pero que pueden ser incluidos, es la Búsqueda Tabú [8], la cual utiliza una "memoria" para almacenar movimientos previamente realizados o soluciones ya visitadas para evitar realizarlos o recorrerlas de nuevo, respectivamente. Las estructuras de memoria pueden ser de corto, mediando o largo plazo, dependiendo de la complejidad del problema a resolver. Por ende, la implementación de la Búsqueda Tabú puede requerir de algoritmos adicionales de búsqueda y estructuras de datos adecuadas para realizar un recorrido eficaz y eficiente de la memoria y evitar que este manejo sea aún más costoso que la optimización misma. La búsqueda Tabú se ha aplicado principalmente en problemas de optimización combinatoria.

1.5. Heurísticas bio-inspiradas

Heurística bio-inspirada = algoritmo bio-inspirado

Como su nombre lo dice, una heurística bio-inspirada basa su funcionamiento en fenómenos encontrados en la naturaleza. De ahí que se puede construir un

algoritmo (una serie de pasos detallados) que emula en una computadora los comportamientos realizados por seres vivos.

Motivación biológica

Actualmente, los fenómenos naturales que son tomados para diseñar algoritmos heurísticos se pueden dividir en dos grandes grupos:

- **La evolución natural**: La evolución de los seres vivos en el planeta ha sido visto como un proceso de optimización, donde cada especie ha ido mejorando sus capacidades debido a la transmisión de características de padres a hijos, donde aquellos individuos mejor adaptados son los que tienen mayores posibilidades de reproducirse y sobrevivir. Al área que agrupa al conjunto de heurísticas que basan su comportamiento en la evolución de las especies y la supervivencia del más apto se le conoce como Computación Evolutiva [9].
- **Comportamientos sociales en conjuntos de animales e insectos**: Al estudio del comportamiento de agentes (seres vivos) que interactúan de manera local con su ambiente y que dichas interacciones causan la emergencia de comportamientos colectivos coherentes y útiles (inteligencia colectiva) se le conoce como Inteligencia en Cúmulos (Swarm Intelligence) [10]. Ejemplo de inspiraciones naturales son las siguientes: Parvadas de aves, bancos de peces, cúmulos de insectos, hormigas, termitas, etc.

En este capítulo se analizarán 2 heurísticas propuestas a mediados de los años noventa, una tomada de la Computación Evolutiva, llamada Evolución Diferencial y la otra de la Inteligencia en Cúmulos, llamada Optimización Mediante Cúmulos de Partículas.

1.6. Componentes de un algoritmo bio-inspirado

Los algoritmos evolutivos y de inteligencia en cúmulos, de manera general, trabajan de la siguiente manera (Véase la Figura 1):

1. Generar un conjunto (población) de soluciones (individuos) al problema.
2. Evaluar cada solución en la función objetivo a optimizar.
3. Seleccionar las mejores soluciones de la población con base en su valor en la función objetivo.
4. Generar nuevas soluciones a partir de las mejores soluciones utilizando operadores de variación.
5. Evaluar las nuevas soluciones.
6. Escoger las soluciones que formarán parte de la siguiente iteración (generación).

Aunque existen diferencias específicas entre los diversos algoritmos evolutivos y también entre aquellos de inteligencia en cúmulos, tienen componentes comunes, los cuales se enumeran a continuación:

- **Representación de soluciones**: Las soluciones se pueden representar de diferentes formas (cadenas binarias, números enteros). Para el caso de la optimización numérica, la representación más adecuada es utilizar números reales, puesto que las variables de un problema de optimización numérica suelen tomar precisamente este tipo de valores. Por ende, cada solución consistirá de un vector de números reales.
- **Mecanismo de selección**: La guía de la búsqueda se realiza en este mecanismo o en el reemplazo (explicado más adelante). Estos mecanismos seleccionan un subconjunto de soluciones de la población para generar nuevas soluciones. El criterio para escoger depende normalmente del valor de la función objetivo de cada solución.
- **Operadores de variación**: Los operadores de variación generan nuevas soluciones a partir de las soluciones existentes (aquellas escogidas con los mecanismos de selección). Existen diversas maneras, entre las que destacan el generar una nueva solución a partir de una sola solución, o generar una o varias nuevas soluciones a partir de dos o más soluciones existentes.
- **Mecanismos de reemplazo**: Una vez que se tiene el conjunto de soluciones actual, además de las recién generadas, se utilizan mecanismos para escoger aquellas que formarán parte de la población para la siguiente generación. Este proceso puede estar basado en la calidad de cada solución (valor de la función objetivo), en la "edad" de las soluciones o tomando en cuenta aspectos aleatorios.

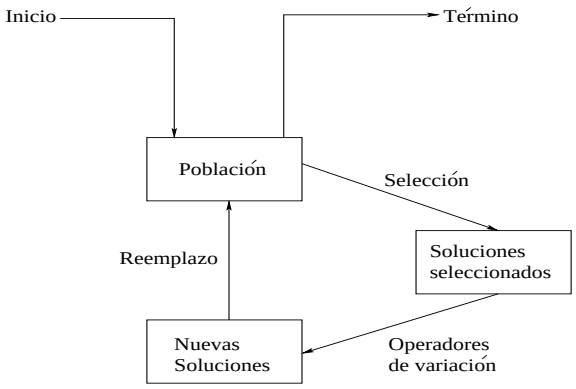


Figura 1. Algoritmo bio-inspirado

En las siguientes 2 secciones se presentarán el algoritmo evolutivo conocido como Evolución Diferencial y el algoritmo de inteligencia en cúmulos llamado Optimización Mediante Cúmulos de Partículas, detallando cada uno de sus componentes y mostrando un sencillo ejemplo de aplicación.

2. Evolución Diferencial

La Evolución Diferencial (ED) es un método relativamente nuevo, que se ha convertido rápidamente en uno de los más empleados dentro de los algoritmos evolutivos, debido a que se entiende fácilmente y su implementación es sencilla.

2.1. Motivación

ED fue propuesto por Storn y Price en 1995, tiene sus orígenes en el algoritmo genético de Recocido Simulado (*Genetic Annealing algorithm*, [7]) empleado para resolver el problema de ajuste polinominal de Chebysev. La implementación de éste algoritmo resultó satisfactoria, no obstante tenía algunas deficiencias, siendo las principales la convergencia y el ajuste de parámetros. Posteriormente, Price realizó ciertas modificaciones que fortalecieron sus bases [11].

ED es un método de búsqueda directa (no requiere el cálculo del gradiente de la función a optimizar) que puede emplearse para encontrar el mínimo de una función multimodal (que tiene muchos óptimos locales). Su primera publicación apareció como un reporte técnico en 1995 [12]. Es un método que converge en la mayoría de los problemas y es viable para la paralelización. Principalmente se utiliza para resolver problemas en espacios de búsqueda contínuos, es decir, problemas donde las variables toman valores de los números reales.

ED se considera un algoritmo evolutivo porque maneja una población inicial de soluciones al problema (llamadas vectores) que se genera de forma aleatoria con una distribución uniforme. Además, utiliza operadores de mutación y cruza (típico en los algoritmos evolutivos) para generar un nuevo vector a partir de cada uno de los ya existentes. Posteriormente, compara los resultados de la evaluación en la función objetivo del vector padre y el vector hijo para determinar al mejor de ellos y conservarlo para la siguiente generación (mecanismo de reemplazo).

Debido a que éste método es multipunto (a diferencia de los métodos tradicionales que usualmente manejan sólo un punto o solución a la vez), se debe especificar el número de vectores para formar la población, número de generaciones que va a ejecutarse, además de los valores que se utilizan en el proceso de mutación y cruza para generar nuevos vectores. ED tiene pocos parámetros, los cuales mantienen fijos sus valores durante el proceso completo de optimización [13] y se muestran en la Tabla 1.

2.2. Elementos de la Evolución Diferencial

Representación de soluciones

Cuadro 1. Parámetros de la ED.

Nombre	Símbolo	Descripción
Número de vectores	NP	Número de soluciones en la población
Generaciones	MAX_GEN	Número de generaciones (iteraciones) que se ejecutará la ED
Factor de escala	F	Multiplica al vector diferencia que se utiliza para calcular el vector de mutación. Es un valor real entre (0,1+]
Porcentaje de cruza	CR	Determina la aportación del vector de mutación (con respecto al vector padre) en la generación del vector hijo

La representación de las soluciones se lleva a cabo mediante vectores de D-dimensiones (donde D es el número de variables del problema). Para representar una población es necesario tener NP vectores (individuos) y se numeran de 1 a NP. La representación gráfica se muestra en la Figura 2 y la representación vectorial del individuo i en la generación g es la siguiente:

$$\boldsymbol{x}_{i,g}, i = 1, \ldots, NP, g = 1, \ldots, MAX\_GEN$$

2.1	2.9	10.1	7.0

Figura 2. Representación de un individuo de 4-dimensiones con representación real para la ED.

Cada una de las variables que forma parte de la solución está asociada a un rango ($l_{inf} \leq x_i \leq l_{sup}$), el cual se debe de tomar en cuenta al momento de generar de manera aleatoria, con una distribución uniforme, las soluciones iniciales.

Mecanismo de selección

En ED no existe una selección de padres como tal, pues cada vector en la población, invariablemente, genera siempre un vector hijo valiéndose de los operadores de variación propios de la ED. Esta es una gran diferencia con respecto a otros algoritmos evolutivos como los algoritmos genéticos [14] o las estrategias evolutivas [15], donde cada individuo puede dar origen a ninguno, uno, dos o más nuevas soluciones.

Operadores de variación

ED es un método que emplea la recombinación (cruza) y mutación para generar nuevas posibles soluciones al problema a resolver a partir de cada una de las soluciones existentes. Primeramente, se calcula el llamado vector de mutación, que se crea mediante una combinación lineal de una diferencia entre 2 vectores sumada a un tercer vector (usualmente seleccionados al azar). Después, el vector hijo se genera al recombinar (cruzar) el vector de mutación con el vector padre.

Mutación: Consiste en calcular el vector de mutación para cada vector en la población. Esto quiere decir que si se tienen NP vectores, se deberán generar NP vectores de mutación, uno por cada vector (llamado vector padre) Este cálculo se hace en 2 fases: (1) Cálculo del vector diferencia (que representa la dirección de búsqueda) y (2) suma de la diferencia (dirección de búsqueda) al vector base. Para calcular el vector diferencia, se utiliza un par de vectores "r_1" y "r_2", seleccionados de la población actual de manera aleatoria y los cuales simplemente se restan para definir una dirección de búsqueda (es decir, una orientación hacia donde se generará una nueva solución). Esta diferencia se multiplica por el factor de escala F. Una vez calculada, este vector diferencia se suma a un tercer vector (llamado vector base o donante y etiquetado como "r_3"). El resultado será el vector de mutación.

Es importante destacar que "r_1", "r_2" y "r_3" deben ser diferentes entre si y diferentes al vector padre. En la Figura 3 se representa gráficamente este proceso, y la Ecuación 1 muestra el cálculo del vector de mutación.

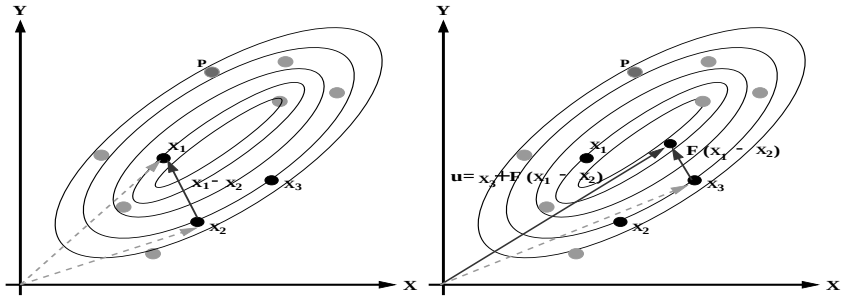


Figura 3. (a) El vector diferencia se calcula entre los vectores $\boldsymbol{x}_1$ y $\boldsymbol{x}_2$. (b) El vector de diferencia escalado ($F \cdot (\boldsymbol{x}_1 - \boldsymbol{x}_2)$) se suma al vector base $\boldsymbol{x}_3$ para generar el vector de mutación $\boldsymbol{v}$.

$$\boldsymbol{v}_{i,g} = \boldsymbol{x}_{r3,g} + F \cdot (\boldsymbol{x}_{r1,g} - \boldsymbol{x}_{r2,g}) \tag{1}$$

Recombinación: Esta operación se lleva a cabo entre el vector padre $\boldsymbol{x}_{i,g-1}$ y el vector de mutación $\boldsymbol{v}_{i,g}$. La recombinación que se realiza es discreta (el vector hijo $\mathbf{u}_{i,g}$ hereda directamente los valores de alguno de sus progenitores, en este caso, el vector padre y el vector de mutación). La ecuación 2 realiza estos cálculos.

$$\mathbf{u}_{i,g} = \boldsymbol{u}_{j,i,g} = \begin{cases} \boldsymbol{v}_{j,i,g} & \text{si } (rand_j(0,1) \leq CR \text{ o } j = j_{rand}) \\ \boldsymbol{x}_{j,i,g-1} & \text{de otra manera.} \end{cases} \tag{2}$$

El valor (número real) de CR es definido por el usuario, se encuentra en un rango de [0,1] y define qué tan parecido será el vector hijo con respecto al vector de mutación o al vector padre. Si CR es cercano a 1, el vector hijo se parecerá mucho al vector de mutación, si CR es cercano a 0, el vector hijo será muy similar al vector padre. Cabe mencionar que, aunque el valor de CR sea igual a 0 (lo que implicaría que el vector hijo sea exactamente igual al vector padre), al menos una posición del vector hijo siempre será distinta, debido a la condición $j = j_{rand}$, donde j representa una variable del vector que se está calculando y j_{rand} es una posición del vector generada de manera aleatoria en la cual el vector padre y el vector hijo diferirán.

ED maneja principalmente 2 tipos de recombinación discreta: exponencial y binomial. Ambas utilizan el valor del parámetro CR para determinar el parecido del vector hijo con respecto al vector de mutación o al vector padre [16] y un valor j_{rand} que representa una posición aleatoria del vector. Si se utiliza la cruza exponencial (*exp*) se inicia en la posición j_{rand} y el vector se considera como una cola circular donde al llegar a la posición final se continúa con las primeras posiciones de éste si aún no han sido consideradas (si j_{rand} es una posición media del vector). Se genera un valor aleatorio entre 0 y 1 y los elementos del vector de mutación se copian al vector hijo hasta que un valor aleatorio sea más grande que CR, una vez que se cumple la condición, los restantes valores se copian del vector padre (véase la Figura 4 (a)). Por otro lado, la cruza binomial (*bin*) funciona de la siguiente manera: Para cada posición del vector, se genera un número aleatorio entre 0 y 1, si este valor es más pequeño que CR (Ecuación 2), se copia el valor del vector de mutación; en caso contrario, se copia del vector padre. La posición j_{rand} se tomará del vector de mutación independientemente del valor de CR (véase la Figura 4 (b)).

Por lo tanto, en la recombinación exponencial, siempre quedarán elementos del vector de mutación en un extremo del vector hijo y los elementos del vector padre en el otro extremo (similar a una cruza de 1 punto) cuando j_{rand} es igual a 1, o bien el resultado será similar a una cruza de 2 puntos si el valor de j_{rand} es intermedio. En cambio, en la recombinación binomial, los elementos del vector de mutación y del vector padre quedan intercalados en el vector hijo. Para valores intermedios de CR la recombinación binomial introduce más elementos del vector de mutación que su contraparte exponencial.

Mecanismo de reemplazo

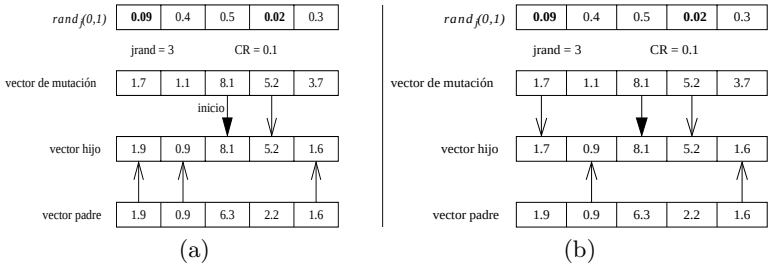


Figura 4. (a) Cruza exponencial, el vector $rand_j(0,1)$ contiene valores $\epsilon(0,1)$, el valor en negrita es menor que CR, por lo tanto esa posición se copia del vector de mutación y el resto del vector padre, se inicia en la posición j_{rand}. (b) Representación de la cruza binomial, el vector $rand_j(0,1)$ contiene valores ϵ $(0,1)$, los valores en negrita son menores que CR y son las posiciones que se copian del vector de mutación en el vector hijo. Las restantes se copian del vector padre. Los valores de la posición j_{rand} en ambos casos toman el valor del vector de mutación.

Una vez que cada vector padre ha generado un vector hijo utilizando los operadores de variación, el tamaño de la población se habrá duplicado $(2NP)$. El mecanismo de reemplazo tiene la función de mantener la población en su tamaño original (NP) prefiriendo a los mejores vectores. La comparación se realiza de manera directa entre el vector padre $\mathbf{x}_{i,g}$ y el vector hijo $\mathbf{u}_{i,g}$; el que tenga el mejor valor de la función objetivo permanecerá para la siguiente generación y el peor de ambos será eliminado (véase la ecuación 3 donde se asume que se está resolviendo un problema de minimización). Esta comparación directa entre individuos similares (padres con hijos) hace que ED incorpore la recombinación y selección más que cualquier otro algoritmo evolutivo [11].

$$\mathbf{x}_{i,g} = \begin{cases} \mathbf{u}_{i,g} \text{ si } f(\mathbf{u}_{i,g}) \leq f(\mathbf{x}_{i,g-1}) \\ \mathbf{x}_{i,g} \text{ de otra manera.} \end{cases} \tag{3}$$

Una vez que la nueva población es generada, el proceso de mutación, recombinación y reemplazo son repetidos hasta que se cumpla una condición de paro (usualmente definida por un número de generaciones).

El proceso explicado se muestra como algoritmo en la Figura 5, para la variante $ED/rand/1/bin$.

2.3. Variantes

La variante hasta ahora descrita en este capítulo es la más popular, conocida como $ED/rand/1/bin$. Esta nomenclatura se explica de la siguiente manera: ED significa Evolución Diferencial, *rand* indica que el vector base se escoge de manera aleatoria, 1 es el número de diferencias que se calculan (por cada difer-

```
1  Begin
2      G=0
3      Crear una población inicial aleatoria x_{i,G} ∀ i, i = 1,...,NP
4      Evaluar f(x_{i,G}) ∀ i, i = 1,...,NP
5      For G=1 to MAX_GEN Do
6          For i=1 to NP Do
7              Seleccionar aleatoriamente r1 ≠ r2 ≠ r3
8              j_rand = randint(1,D)
9              For j=1 to D Do
10                 If (rand_j[0,1) < CR or j = j_rand) Then
11                     u_{i,j,G} = x_{r3,j,G-1} + F(x_{r1,j,G-1} − x_{r2,j,G-1})
12                 Else
13                     u_{i,j,G} = x_{i,j,G-1}
14                 End If
15             End For
16             If (f(u_{i,G}) ≤ f(x_{i,G-1})) Then
17                 x_{i,G} = u_{i,G}
18             Else
19                 x_{i,G} = x_{i,G-1}
20             End If
21         End For
22     End For
23 End
```

Figura 5. Algoritmo de ED/*rand*/1/*bin*. *randint* es una función que devuelve un número aleatorio entero entre 1 y D. *rand* es una función que devuelve un número aleatorio real entre 0 y 1. CR, F, NP y MAX_GEN son parámetros proporcionados por el usuario, D es el número de variables del problema.

encia se requieren 2 vectores seleccionados de manera aleatoria), y finalmente *bin* indica el tipo de recombinación, que es la binomial en este caso.

Las principales variantes [17,13] que existen son las siguientes:

1. ED/*rand*/*p*/*bin*.
2. ED/*rand*/*p*/*exp*.
3. ED/*best*/*p*/*bin*.
4. ED/*best*/*p*/*exp*.
5. ED/*current-to-rand*/1.
6. ED/*current-to-best*/1.

La interpretación de cada una de las variantes es la siguiente:

- *best, rand*. El vector base se escoge de 2 formas distintas: (1) El mejor de la población o (2) de manera aleatoria.
- *current-to-best, current-to-rand*. A diferencia de la cruza discreta (discutida en este capítulo), en estas variantes se realiza cruza aritmética, utilizando al

vector base como: (1) el mejor de la población o (2) un individuo escogido de manera aleatoria.

- p. Representa el número de diferencias consideradas en el cálculo del vector diferencia. Usualmente se utiliza un valor de 1 pero el número de pares puede ser mayor. El efecto esperado al incrementar el número de diferencias es permitir que el algoritmo explore con mayor detalle el espacio de búsqueda, pero en consecuencia, tardará más en regresar una buena solución [11].
- *bin o exp*. Indica el tipo de recombinación discreta que se va a emplear. ED utiliza dos tipos de recombinación discreta: exponencial o binomial, véase sección 2.2.

La Tabla 2 muestra las fórmulas empleadas para el vector de mutación en cada una de las variantes, así como la condición que se aplica para la recombinación exponencial y la binomial. Este es el único cambio entre las diferentes variantes de la ED. Los demás pasos se mantienen sin cambios.

Cuadro 2. Variantes principales de la ED.

Nomenclatura	Operador de recombinación y mutación
rand/p/bin	$u_{i,j} = \begin{cases} x_{r3,j} + F \cdot \sum_{k=1}^{p}(x_{r_1^p,j} - x_{r_2^p,j}) & \text{si } U_j(0,1) < CR \text{ o } j = j_{rand} \\ x_{i,j} & \text{de otra manera} \end{cases}$
rand/p/exp	$u_{i,j} = \begin{cases} x_{r3,j} + F \cdot \sum_{k=1}^{p}(x_{r_1^p,j} - x_{r_2^p,j}) & \text{mientras } U_j(0,1) < CR \text{ o } j = j_{rand} \\ x_{i,j} & \text{de otra manera} \end{cases}$
best/p/bin	$u_{i,j} = \begin{cases} x_{best,j} + F \cdot \sum_{k=1}^{p}(x_{r_1^p,j} - x_{r_2^p,j}) & \text{si } U_j(0,1) < CR \text{ o } j = j_{rand} \\ x_{i,j} & \text{de otra manera} \end{cases}$
best/p/exp	$u_{i,j} = \begin{cases} x_{best,j} + F \cdot \sum_{k=1}^{p}(x_{r_1^p,j} - x_{r_2^p,j}) & \text{mientras } U_j(0,1) < CR \text{ o } j = j_{rand} \\ x_{i,j} & \text{de otra manera} \end{cases}$
current-to-rand/1	$u_i = x_i + K \cdot (x_{r3} - x_i) + F \cdot \sum_{k=1}^{p}(x_{r_1^p} - x_{r_2^p})$
current-to-best/1	$u_i = x_i + K \cdot (x_{best} - x_i) + F \cdot \sum_{k=1}^{p}(x_{r_1^p} - x_{r_2^p})$

2.4. Ejemplo de funcionamiento

A continuación se presenta un ejemplo de la variante ED/*rand/1/bin* aplicado a una función muy sencilla conocida como la esfera:

$$\text{Minimizar: } f(x_1, x_2) = x_1^2 + x_2^2$$

Los intervalos para las variables son:

$$-5 \leq x_1 \leq 5$$
$$-5 \leq x_2 \leq 5$$
$$x_1, x_2 \in \mathbb{R}$$

Los valores de los parámetros de la ED se muestran en la Tabla 3.

Cuadro 3. Parámetros de ED empleados en la función de la esfera.

Parámetro	Valor
NP	5
MAX_GEN	5
F	0.7
CR	0.0001

Siguiendo el algoritmo mostrado en la Figura 5, primero se inicializa el contador de generaciones (G) con el valor de 0, posteriormente se crea población inicial de manera aleatoria (la cual está dentro del rango especificado) y se muestra en la Tabla 4.

Cuadro 4. Población inicial, el valor de la función para cada individuo se encuentra en la cuarta columna.

Individuo	x_1	x_2	$f(\boldsymbol{x}_i)$
1 ($\boldsymbol{x}_{1,0}$)	0.5	0.3	0.34
2 ($\boldsymbol{x}_{2,0}$)	-0.2	0.1	0.05
3 ($\boldsymbol{x}_{3,0}$)	-0.4	-0.7	0.65
4 ($\boldsymbol{x}_{4,0}$)	0.2	0.05	0.0425
5 ($\boldsymbol{x}_{5,0}$)	0.1	0.9	0.82

Posteriormente se tiene que evaluar cada uno de los individuos en la función objetivo, estos valores se muestran en la cuarta columna de la Tabla 4.

En el paso 5 empieza el ciclo del número de generaciones y en el 6 el ciclo para llevar a cabo las operaciones de ED en cada uno de los individuos.

Para el individuo 1 ($\boldsymbol{x}_{1,0}$), supongamos que los vectores para el cálculo del vector diferencia son, $r_1 = \boldsymbol{x}_{3,0}$ y $r_2 = \boldsymbol{x}_{5,0}$. Además, el vector base, escogido de manera aleatoria (pues la variante es *rand*) $r_3 = \boldsymbol{x}_{2,0}$. Supóngase también $j_{rand} = 1$. Los valores aleatorios que se emplean para la generación del vector de mutación son:

$rand_1$ 0.05
$rand_2$ 2.5e-5

al comparar $rand_1$ con CR (porcentaje de cruza) se tiene que $rand_1 \geq CR$, por lo que ese valor debería tomarse del vector padre. Sin embargo, la posición del vector que se está calculando es la misma que j_{rand}, por lo tanto, la posición del vector se calcula con la ecuación 1 (renglón 11 en la Figura 5). Por otro lado, para la segunda variable (x_2) el valor de $rand_2$ es menor que CR, por lo que se emplea nuevamente la ecuación 1. Como puede notarse, en este caso los cálculos para el vector hijo se tomarán, ambos, del vector de mutación y son los siguientes:

$$u_{1,1,1} = [\text{-}0.2 + 0.7 \cdot (\text{-}0.4 - 0.1)] = \text{-}0.2 + (\text{-}0.35) = \text{-}0.55$$
$$u_{1,2,1} = [0.1 + 0.7 \cdot (\text{-}0.7 - 0.9)] = 0.1 + (\text{-}1.12) = \text{-}1.02$$

quedando el vector hijo de la siguiente manera:

$$\boldsymbol{u}_1 = [\text{-}0.55, \text{-}1.02]$$

Aplicando el mecanismo de reemplazo para determinar quién sobrevive para la siguiente generación se toma en cuenta el valor en la función objetivo del padre ($\boldsymbol{x}_{1,0}$) y del recién generado vector hijo ($\boldsymbol{u}_{1,1}$), renglón 16 en la Figura 5 (Ecuación 3).

$$f(\boldsymbol{x}_{1,0}) = 0.34$$
$$f(\boldsymbol{u}_{1,1}) = 1.3429$$

por lo cual el individuo que pasa a la siguiente generación es el vector padre por tener un valor más pequeño (recuérdese que se está minimizando la función objetivo).

$$\boldsymbol{x}_{1,1} = [0.5, 0.3]$$

Para el individuo 2 ($\boldsymbol{x}_{2,0}$) se realiza el mismo proceso, el cual se describe a continuación: Los vectores para calcular el vector de mutación son: $r_1 = \boldsymbol{x}_{4,0}$, $r_2 = \boldsymbol{x}_{1,0}$ y $r_3 = \boldsymbol{x}_{3,0}$. Además, el valor para $j_{rand} = 1$. Los valores aleatorios que se emplean para la generación del vector de mutación son:

$$rand_1\text{=}0.887$$
$$rand_2\text{=}0.039$$

Estos valores son mayores que CR, pero cuando $j = 1$ (asociada a x_1, la primera variable del vector) se aplica la ecuación 1 (renglón 11 de la Figura 5). Para $j = 2$ (x_2), el valor del vector de mutación para esa posición es $x_{2,2,1}$ (renglón 13 de la Figura 5). Este caso particular ejemplifica que, a pesar de que $CR = 0.0001$, lo que implica que el vector hijo sea muy parecido a su padre, el descendiente tendrá al menos una posición diferente con respecto al vector padre (Ecuación 1 y renglón 11 en la Figura 5). Los cálculos para el vector de mutación quedan de la siguiente manera:

$$u_{2,1,1} = [\text{-}0.4 + 0.7 \cdot (0.2 - 0.5)] = \text{-}0.4 + (\text{-}0.21) = \text{-}0.61$$
$$u_{2,2,1} = 0.1$$

Quedando el vector hijo de la siguiente manera:

$$\boldsymbol{u}_2 = [\text{-}0.61, 0.1]$$

De nuevo, el mecanismo de reemplazo se lleva a cabo con base en el valor de de la función objetivo del vector padre ($\boldsymbol{x}_{2,0}$) y del nuevo hijo ($\boldsymbol{u}_{2,1}$) (renglón 16 de la Figura 5):

$$f(\boldsymbol{x}_{2,0}) = 0.05$$
$$f(\boldsymbol{u}_{2,1}) = 0.3821$$

Por lo tanto el individuo que pasa a la siguiente generación es el vector padre, pues tiene un valor menor al del vector hijo:

$$\boldsymbol{x}_{2,1} = [\text{-}0.2, 0.1]$$

Los cálculos para el individuo 3 se muestran a continuación:

Vectores	$r_1 = \boldsymbol{x}_{1,0}$, $r_2 = \boldsymbol{x}_{4,0}$, $r_3 = \boldsymbol{x}_{2,0}$
j_{rand}	2
	$rand_1 = 0.048$, $rand_2 = 1.5e\text{-}2$
	$u_{3,1,1} = \text{-}0.4$
	$u_{3,2,1} = [0.1 + 0.7 \cdot (0.3 - 0.05)] = 0.1 + 0.175 = 0.275$
$f(\boldsymbol{u}_{3,1})$=0.236	$f(\boldsymbol{x}_{3,0})$=0.65

Como puede observarse el valor de la función del vector hijo es menor que el valor de $\boldsymbol{x}_{3,0}$, por lo tanto el vector que pasa a la siguiente generación es:

$$\boldsymbol{x}_{3,1} = [\text{-}0.4, 0.275]$$

Los cálculos para el individuo 4 se presentan a continuación

Vectores	$r_1 = \boldsymbol{x}_{3,0}$, $r_2 = \boldsymbol{x}_{5,0}$, $r_3 = \boldsymbol{x}_{2,0}$
j_{rand}	1
	$rand_1 = 21e\text{-}8$, $rand_2 = 0.48$
	$u_{4,1,1} = [\text{-}0.2 + 0.7 \cdot (\text{-}0.4 - 0.1)] = \text{-}0.2 + (\text{-}0.35) = \text{-}0.55$
	$u_{4,2,1} = 0.05$
$f(\boldsymbol{u}_{4,1})$=0.305	$f(\boldsymbol{x}_{4,0})$=0.0425
$\boldsymbol{x}_{4,1} = [0.2, 0.05]$	

Como se puede observar, el vector padre genera un mejor resultado que el vector hijo, así que este se conserva para la siguiente generación.
Los valores que produce ED al aplicarlos al individuo 5 se muestran a continuación:

Vectores	$r_1 = \boldsymbol{x}_{2,0}$, $r_2 = \boldsymbol{x}_{1,0}$, $r_3 = \boldsymbol{x}_{4,0}$
j_{rand}	2
	$rand_1 = 9e\text{-}4$, $rand_2 = 0.49$
	$u_{5,1,1} = 0.1$
	$u_{5,2,1} = [0.05 + 0.7 \cdot (0.1 - 0.3)] = 0.05 + (\text{-}0.14) = -0.09$
$f(\boldsymbol{u}_{5,1})$=0.0181	$f(\boldsymbol{x}_{5,0})$=0.82
$\boldsymbol{x}_{5,1}$=[0.1, -0.09]	

Como se puede observar el vector hijo genera un mejor resultado que el vector padre, por lo tanto el vector hijo pasa a la siguiente generación.

Este proceso se repite 4 veces más para completar las 5 generaciones. La población final, y su valor de aptitud se muestran en la Tabla 5.

Cuadro 5. Población final y su valor de aptitud de cada individuo. El mejor individuo obtenido se muestra en negrita.

Individuo	x_1	x_2	$f(\boldsymbol{x})$
1 ($x_{1,5}$)	-0.151	-0.028	0.024
2 ($x_{2,5}$)	0.17	0.1	0.039
3 ($\mathbf{x_{3,5}}$)	**-0.109**	**0.046**	**0.014**
4 ($x_{4,5}$)	0.151	0.025	0.023
5 ($x_{5,5}$)	0.1	-0.09	0.018

La solución a reportar será la partícula 4, con $x_1 = -0{,}109$, $x_2 = 0{,}046$ y $f(\boldsymbol{x}) = 0{,}014$

Como dato adicional, el mínimo global de esta función se encuentra en el punto [0,0], en donde $f(\boldsymbol{x}) = 0$. Se invita al lector a implementar el algoritmo propuesto en la Figura 5 y ejecutarlo con los siguientes valores: $NP = 90$, $MAX\_GEN = 100$, $F = 0{,}5$ y $CR = 0{,}9$.

3. PSO

La Optimización mediante Cúmulos de Partículas es una heurística que fue propuesta para resolver problemas de optimización, denominada PSO por sus siglas en inglés (Particle Swarm Optimization). PSO es una técnica fácil de implementar y de funcionamiento sencillo. Por su naturaleza representa un paradigma diferente a los algoritmos evolutivos, catalogándose dentro del área de la Inteligencia en Cúmulos, pues no se simula el proceso de evolución, sino más bien el comportamiento individual y social de agentes sencillos de los que emerge un comportamiento grupal complejo [18].

3.1. Motivación

PSO fue propuesto por Kennedy y Eberhart en 1995 [19] y fue concebido como un algoritmo de Inteligencia en Cúmulos por tener sus orígenes en la idea básica del comportamiento social de las bandadas de pájaros o bancos de peces. En estos grupos animales se establecen relaciones sociales entre los individuos del grupo, definiéndose jerarquías de acuerdo a las características de cada uno de ellos, existiendo un líder que es reconocido y seguido por los individuos de su grupo. El líder es un individuo que tiene características físicas o habilidades

Cuadro 6. Parámetros de PSO.

Nombre	Símbolo	Descripción
Número de partículas	NP	Número de Partículas en el cúmulo
Generaciones	MAX_GEN	Número de generaciones (iteraciones) que se ejecutará PSO
Peso de inercia	w	Regula la influencia de la velocidad actual al calcular el nuevo valor de velocidad
Coeficiente de aceleración c1	c1	Regula la influencia de la memoria de la partícula (mejor posición recordada) al calcular el nuevo valor de velocidad
coeficiente de aceleración c2	c2	Regula la influencia del líder al calcular el nuevo valor de velocidad

que le permiten mantener un control sobre los demás en el grupo, y ser el guía para desarrollar actividades como buscar comida o moverse a otro lugar cuando la comida se ha acabado. Los individuos confían en la capacidad del líder para dirigirlos, pero en estos grupos sociales se suele cambiar de líder si surge un individuo con mejores capacidades que el viejo líder, sustituyéndolo en la guía del grupo. Cada individuo ve influenciado su comportamiento por dos factores: El primero es el conocimiento y habilidades propias adquiridas durante su vida o de manera innata, y el segundo es la influencia del líder. Cuando el grupo se organiza y decide, en el caso de aves, emprender el vuelo para ir en busca de comida, el líder guía al grupo por donde tiene la sospecha encontrarán alimento; sin embargo, todos los individuos, también durante su vuelo, tratan de avistar comida y avisar al grupo de la nueva dirección que deben tomar con el objeto de tener éxito en su misión, ya que en esa nueva dirección pueden encontrar alimento. Durante el vuelo se debe tener noción de la posición propia y de la de los demás, a fin de evitar colisiones y el perder la dirección de búsqueda del líder. Este comportamiento social de las bandadas de pájaros es la base de PSO, donde cada individuo del grupo es representado por una partícula en el cúmulo y esa partícula tiene una posición actual, la cual representa una solución al problema de optimización.

La Optimización Mediante Cúmulos de Partículas es utilizada para resolver problemas de optimización, tiene, al igual que en el caso de la ED, un espacio de búsqueda definido por la dimensionalidad del problema a resolver y por el rango de las variables. PSO es una heurística que utiliza una población de soluciones representadas por las partículas del cúmulo, la cual no varía su tamaño a través de las generaciones, ya que no existe un mecanismo de generación de hijos ni reemplazo como en la ED y otros algoritmos evolutivos. La Tabla 6 muestra los parámetros de esta heurística. El conocimiento cognitivo de cada partícula se refiere a su experiencia propia en la búsqueda y es representado por su mejor posición alcanzada (y su correspondiente valor de la función objetivo) la cual es

recordada. Además, se almacena la posición actual de la partícula con su valor propio de la función objetivo. El conocimiento social se refiere a los valores de la mejor partícula de todo el cúmulo (líder) que será tomada como referencia para guiar a las demás partículas. En cada generación las partículas actualizan su posición utilizando como referencia tanto el conocimiento cognitivo (su memoria) como el social (el líder).

3.2. Elementos de la Optimización Mediante Cúmulos de Partículas

Representación de soluciones

Al igual que en la ED, las soluciones (partículas) se representan como vectores D-dimensionales $\boldsymbol{x} = (x_{i1}, x_{i2}, ..., x_{iD})$, donde los valores de cada posición del vector se encuentran acotados por límites inferior y superior. Inicialmente se genera un cúmulo de NP partículas con valores aleatorios utilizando una distribución uniforme dentro del rango de las variables. Cada partícula representa un punto en el espacio de búsqueda. Mediante el vector de valores para cada variable se puede obtener el valor de la función objetivo que nos da una medida de la calidad de la solución representada por la partícula.

Mecanismo de selección

En PSO el mecanismo de selección consiste en identificar al líder del cúmulo (p_{gd} ó p_{global}). Este proceso se realiza mediante la búsqueda de aquella partícula que tenga el mejor valor de la función objetivo de entre todas las partículas del cúmulo.

Memoria de las partículas

Cada partícula tiene la capacidad de recordar la mejor posición por donde ha pasado mediante una memoria propia, que es representada de como $pBest(\boldsymbol{x})$, que forma parte de su conocimiento cognitivo. Para cada partícula recordar su mejor posición por donde ha pasado es de utilidad, puesto que reduce la influencia del conocimiento social (representado por el líder del cúmulo) al actualizar su velocidad, y permite a la partícula explorar el área de búsqueda cercana a donde encontró su mejor posición hasta el momento.

Función de vuelo

Dada su naturaleza, cada partícula tiene asociada un vector de velocidad denotado como $\boldsymbol{v}_i = (v_{i1}, v_{i2}, ..., v_{iD})$ de la misma dimensión que el vector de soluciones.

Este vector es actualizado en cada generación, previo a realizar el movimiento de la partícula. A este movimiento se le conoce como "vuelo".

El vector de velocidad, en sí, representa una dirección de búsqueda (hacia dónde se dirigirá la partícula en su siguiente movimiento). Esta dirección equivale a las diferencias que se calculan para el algoritmo de la ED. La Figura 6 nos muestra los factores que influencian la dirección de búsqueda de una partícula. La fórmula para calcular la velocidad de la partícula es la siguiente:

$$v_{id}(t+1) = w * v_{id}(t) + c_1 * rand() * (p_{id} - x_{id}) + c_2 * Rand() * (p_{gd} - x_{id}) \quad (4)$$

Donde $d = 1, 2, \ldots, D$. La nueva velocidad ($\boldsymbol{v}_i(t+1)$) de la partícula es determinada en su forma general por los siguientes parámetros:

- Peso de inercia w. Determina qué tanto se tomará en cuenta la velocidad anterior $v_{id}(t)$ en el cálculo de su nuevo valor $v_{id}(t+1)$. El factor de inercia promueve, bajo ciertas condiciones, la convergencia del cúmulo, es decir, que todas las partículas se agrupen en un sólo punto, el cual se espera sea la solución óptima al problema de optimización.
- $\boldsymbol{v}_i(t)$. Representa la velocidad actual de la partícula que fue calculada en la iteración previa. Este valor representa la dirección de vuelo que tiene la partícula. En un inicio todas la partículas tienen una velocidad de 0, lo que indica que toda partícula parte de un estado de reposo.
- Coeficientes de aceleración ($c_1 * rand()$ *y* $c_2 * Rand()$). Rand() y rand() representan un número real aleatorio con distribución uniforme entre 0 y 1. El coeficiente c1 regula la influencia del conocimiento cognitivo de la partícula y el coeficiente c2 regula la influencia del conocimiento social. Esto quiere decir que c1 regula la influencia que tiene la mejor posición alcanzada por la partícula ($\boldsymbol{p}_i$) para guiar su nueva dirección y c2 regula la influencia del líder del cúmulo ($\boldsymbol{p}_g$) en el cálculo de la dirección de búsqueda de la partícula. $c_1 * rand()$ y $c_2 * Rand()$ pueden ser representados como ϕ_1 y ϕ_2 respectivamente. Si $\phi_1 > 0$ y $\phi_2 = 0$, únicamente el conocimiento cognitivo influye para actualizar la velocidad. Si $\phi_1 = 0$ y $\phi_2 > 0$, únicamente el conocimiento social influye para actualizar la velocidad. Si $\phi_1 > 0$ y $\phi_2 > 0$ intervienen ambos conocimientos para guiar la nueva dirección de búsqueda.
- $\boldsymbol{p}_i$ *o* $pBest$. Representa la memoria de la partícula, es decir, su mejor posición alcanzada hasta esa generación.
- $\boldsymbol{p}_g$ *o* $gBest$. Representa la mejor posición del cúmulo, es decir, al líder.
- $\boldsymbol{x}_i$. Representa la posición actual de la partícula, que se toma como referencia para calcular la nueva velocidad (dirección de búsqueda).
- $\boldsymbol{v}_{id}(t+1)$. Es la velocidad actualizada de la partícula que determinará la nueva dirección de búsqueda de la partícula en la generación $t+1$.

Una vez calcula la velocidad actualizada $\boldsymbol{v}_i(t+1)$ se realiza el vuelo de la partícula (actualización de su posición) mediante la siguiente fórmula:

$$x_{id} = x_{id} + v_{id} \quad (5)$$

Donde $d = 1, 2, \ldots, D$. La posición actualizada de la partícula no debe exceder el límite inferior y superior del vector de variables, pudiendo establecer una velocidad máxima o mínima para poder evitar salirse del rango permitido para cada una de las variables del problema.

3.3. Variantes

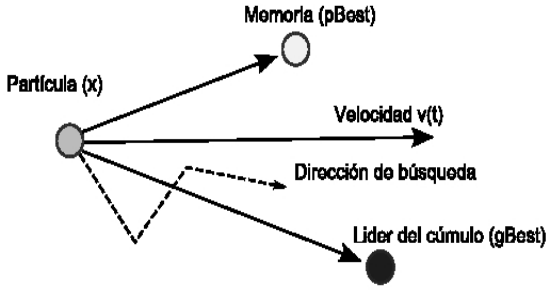


Figura 6. *Esquematización del funcionamiento de PSO*

Existen dos variantes básicas de PSO que se clasifican de acuerdo a la manera en que un individuo se relaciona socialmente con los demás individuos del cúmulo. La primera variante se denomina global best (*gbest*), en donde existe un solo líder en todo el cúmulo, por lo tanto cada partícula se comunica con todas las demás, para poder conocer la posición del líder único (véase la Figura 7a). El algoritmo se detalla en la Figura 8.

La segunda variante se conoce como local best (*lbest*). Para esta variante el cúmulo completo se divide en un numero n de vecindarios con igual número de individuos, y en cada vecindario existe un líder local el cual influencia el vuelo de las partículas dentro de su vecindario. En la Figura 7b se presenta la forma de comunicación entre partículas en esta variante. En este caso, cada partícula se comunica sólo con sus vecinos (2 vecinos en la Figura) y desconoce el comportamiento de las restantes partículas del cúmulo.

3.4. Ejemplo de funcionamiento

Para ilustrar el funcionamiento de PSO se presenta un ejemplo de la variante global best con factor de inercia aplicado a la función de la esfera. Primero se definen los parámetros de PSO en la Tabla 7:

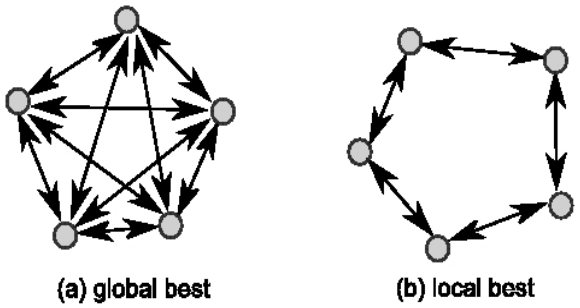


Figura 7. *Tipo de comunicación entre las partículas en sus algoritmos básicos de PSO (global best (a) y local best (b))*

```
1  Begin
2      G=0
3      Crear un cúmulo inicial aleatorio x_i ∀ i, i = 1,...,NP
4      Evaluar f(x_i), ∀ i, i = 1,...,NP
5      Inicializar pBest_i = x_i, ∀ i, i = 1,...,NP
6      Inicializar v(x_i) = 0, ∀ i, i = 1,...,NP
7      For G=1 to MAX_GEN Do
8          Identificar al líder del cúmulo gBest_G
9          con el mejor valor de la función objetivo f(gBest_G)
10         For i=1 to NP Do
11             For d=1 to D Do
12                 v_id(t+1) = w * v_id(t) + c1 * rand() * (pBest_id - x_id)
12                 +c2 * Rand() * (gBest_G - x_id)
13                 x_id = x_id + v_id
14             End For
15             If (f(x_i) ≤ f(pBest_i)) Then
16                 pBest_i = x_i
17             End If
18         End For
19     End For
20 End
```

Figura 8. Algoritmo de PSO global best. *rand()* y *Rand()* son funciones que devuelven un número aleatorio real entre 0 y 1. w, c_1, c_2, NP y MAX_GEN son parámetros proporcionados por el usuario, D es el número de variables del problema.

Cuadro 7. Parámetros del PSO empleados en la función de la esfera

Parámetro	Valor
NP	5
MAX_GEN	5
w	0.7
$c1$	0.7
$c2$	0.5

Ahora, siguiendo el algoritmo global best de la Figura 8 se inicializa el contador de generaciones $G = 0$. Paso siguiente se genera un cúmulo de partículas de tamaño NP que son soluciones generadas aleatoriamente (dentro de los valores del rango especificados para las variables x_1 y x_2) cuyos valores se muestran en la Tabla 8. Para facilitar la comprensión del ejemplo estos valores iniciales son similares a los usados en el ejemplo de la ED. Se agrega también en esta Tabla 8 la evaluación de cada partícula.

Cuadro 8. Cúmulo inicial, valor de las variables y de la función objetivo para cada partícula

Partícula	x_1	x_2	$f(\boldsymbol{x})$
1	0.5	0.3	0.34
2	-0.2	0.1	0.05
3	-0.4	-0.7	0.65
4	0.2	0.05	0.0425
5	0.1	0.9	0.82

Cuadro 9. Cúmulo inicial, valor de las variables y de la función objetivo de la mejor posición de la partícula ($pBest$)

pBest	x_1	x_2	$f(\boldsymbol{x})$
1	0.5	0.3	0.34
2	-0.2	0.1	0.05
3	-0.4	-0.7	0.65
4	0.2	0.05	0.0425
5	0.1	0.9	0.82

Posteriormente se inicializa el $pBest$ de cada partícula con el valor inicial generado aleatoriamente solamente en la primera generación (cuando $G = 0$)

Cuadro 10. Velocidades iniciales, primera generación $v_{id}(t+1)$

Partícula	$v_{i1}(t)$	$v_{i2}(t)$
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0
5	0.0	0.0

mostrados en la Tabla 9 (renglón 5 en la Figura 8). Igualmente el valor inicial de la velocidad para cada partícula es inicializado con 0, ver Tabla 10 (renglón 6 en la Figura 8). Ahora, se procede a elegir al líder del cúmulo (renglones 8 y 9 en la Figura 8). Para ello, se compara el valor de la función objetivo para cada partícula, seleccionando como líder a aquella partícula que tenga el mejor valor en la función objetivo (representado por pgd). El líder seleccionado en la primera generación ($G = 0$) es la partícula 4 (Tabla 8), que tiene el menor valor para la función objetivo (0.0425), y será utilizado para calcular la nueva velocidad de cada partícula en la función de vuelo. Ahora se actualiza la velocidad para la partícula 1, (renglón 12 en la Figura 8). Supóngase que los valores aleatorios generados para $rand()$ es de 0.05 y para $Rand()$ es de 0.137. Entonces, tomando como referencia los valores de los parámetros para la formula de vuelo fijados en la Tabla 7, el calculo para la fórmula de actualización de la velocidad para la partícula 1 se muestra a continuación:

Para x_1, sustituyendo los valores en la fórmula de la Ecuación 4 (renglón 12 en la Figura 8) se tiene:

$$v_{11}(t+1) = (0{,}7 * 0{,}0) + (0{,}7 * 0{,}05 * (0{,}5 - 0{,}5)) + (0{,}5 * 0{,}137 * (0{,}2 - 0{,}5))$$

Se obtiene una velocidad de:

$$v_{11}(t+1) = -0{,}0206$$

Para x_2 sustituyendo los valores

$$v_{12}(t+1) = (0{,}7 * 0{,}0) + (0{,}7 * 0{,}05 * (0{,}3 - 0{,}3)) + (0{,}5 * 0{,}137 * (0{,}05 - 0{,}3))$$

Se obtiene una velocidad de

$$v_{12}(t+1) = -0{,}0171$$

Se sigue el mismo proceso para actualizar la velocidad de las siguientes partículas. Las velocidades actualizadas de las partículas en la primera generación se muestran en la Tabla 12. Los números aleatorios $rand()$ y $Rand()$ utilizados se muestran en la Tabla 11:

Una vez actualizadas las velocidades podemos proceder a actualizar la posición de la partícula mediante la formula $x_{id} = x_{id} + v_{id}$, $d = 1, 2, \ldots, D$ (renglón

Cuadro 11. Números aleatorios utilizados en la primera generación

Generación	$rand()$	$Rand()$
1	0.05	0.137
2	0.1	0.2
3	0.003	0.4
4	0.0987	0.121
5	0.013	0.21

Cuadro 12. Velocidades actualizadas, primera generación

Partícula	$v_{i1}(t+1)$	$v_{i2}(t+1)$
1	-0.0206	-0.0171
2	0.0400	-0.0050
3	0.1200	0.1500
4	0.0000	0.0000
5	0.0105	-0.0893

13 en la Figura 8). Para ejemplificar el proceso se actualiza la posición de la partícula 1:

Para x_1 sustituyendo los valores:

$$x_{11} = 0{,}5 + (-0{,}0206)$$

la nueva posición es:

$$x_{11} = 0{,}4795$$

Para x_2 sustituyendo los valores:

$$x_{12} = 0{,}3 + (-0{,}0171)$$

la nueva posición es:

$$x_{12} = 0{,}2829$$

De igual forma se actualiza la posición de las demás partículas obteniendo nuevos valores para el cúmulo en la siguiente generación. Este proceso representa el vuelo de cada partícula a una nueva posición. Para poder notar el cambio observado en la función objetivo se muestran la posición actualizada de las partículas y su valor para la función en la Tabla 13. Si en un vuelo, el valor de alguna variable se sale de los rangos permitidos, se debe utilizar un mecanismo para que el valor se obtenga dentro de ellos. Una de las opciones más utilizadas es manejar un valor máximo de velocidad (por ejemplo, el valor definido por el rango de la variable) para evitar que en un vuelo, la partícula abandone el espacio de búsqueda. Otra alternativa, es asignar el valor límite más cercano a la variable. Por ejemplo, en el problema de la esfera, si en un vuelo una variable tomara el valor de 10.5,

se modificaría a 5, que es el límite superior del intervalo válido. Es importante mencionar que este mecanismo puede ser válido también para el caso de la ED, pues el operador de mutación y recombinación puede causar que el valor de una variable se salga del rango permitido.

Cuadro 13. Estado del cúmulo después de realizar el vuelo en la primera generación

Partícula	x_1	x_2	$f(\boldsymbol{x})$
1	0.4795	0.2829	0.3099
2	-0.1600	0.0950	0.0346
3	-0.2800	-0.5500	0.3809
4	0.2	0.05	0.0425
5	0.1105	0.8108	0.6695

Cuadro 14. Valor de las variables y de la función objetivo en el valor de su memoria ($pBest$) después de actualizar sus valores con respecto a la posición después del vuelo en la primera generación

pBest	x_1	x_2	$f(\boldsymbol{x})$
1	0.4795	0.2829	0.3099
2	-0.1600	0.0950	0.0346
3	-0.2800	-0.5500	0.3809
4	0.2	0.05	0.0425
5	0.1105	0.8108	0.6695

Siguiendo el algoritmo de la Figura 8, en el renglón 15 y 16 se realiza la actualización de la memoria de cada partícula ($pBest$), comparando los valores de la Tabla 13 (que representan la nueva posición de cada partícula después del vuelo) con los valores actuales de la memoria de cada partícula en la Tabla 9. Si el valor de la función objetivo en la nueva posición es mejor (menor en este caso) que el mejor valor recordado en $pBest$, entonces se actualiza. El resultado de esta actualización se muestra en la Tabla 14.

Como puede notarse, todas las partículas mejoraron el valor de su memoria, excepto el líder, quien no se movió.

Después de la actualización del $pBest$ inicia la siguiente generación, donde se escogerá un nuevo líder, que en este caso, basándonos en los valores de las posiciones actuales de las partículas (Tabla 13) será la partícula 2, que tiene el valor más pequeño de la función objetivo. De aquí en adelante se repite el proceso de manera similar a la descrita hasta ahora.

Una vez transcurridas las 5 generaciones los valores resultantes se muestran en la Tabla 15. La solución a reportar será la partícula 4, con $x_1 = -0{,}0409$, $x_2 = 0{,}0257$ y $f(\boldsymbol{x}) = 0{,}00233$. Se invita al lector a implementar el algoritmo propuesto en la Figura 8 y ejecutarlo con los siguientes valores: $NP = 90$, $MAX\_GEN = 100$, $w = 0{,}5$, $c1 = 0{,}5$ y $c2 = 0{,}7$. ¿Qué tan buenos fueron los resultados con respecto a los obtenidos por la ED? Discuta. Trate de calibrar los parámetros para mejorar el rendimiento de cada uno de los 2 algoritmos.

Cuadro 15. Valor de las variables y de la función objetivo en la memoria de la partícula ($pBest$) después de actualizar sus valores en la última generación. La mejor partícula obtenida se muestra en negrita

pBest	x_1	x_2	$f(x_1, x_2)$
1	0.3707	0.3834	0.28439
2	-0.106	0.1129	0.02397
3	0.3268	0.414	0.27834
4	**0.0409**	**0.0257**	**0.00233**
5	-0.087	0.116	0.02113

4. Resumen y tendencias futuras

En este capítulo se ha presentado una introducción a las heurísticas bio-inspiradas y su papel en la resolución de problemas complejos de búsqueda, específicamente el problema de optimización numérica. Se explicaron 2 heurísticas: (1) un algoritmo evolutivo llamado Evolución Diferencial, que basa su funcionamiento en el cálculo de direcciones de búsqueda usando diferencias entre vectores de la población actual y en un mecanismo de reemplazo basado en la supervivencia del más apto y (2) un algoritmo de inteligencia en cúmulos conocido como Optimización Mediante Cúmulos de Partículas, que modela el comportamiento social de las parvadas de aves para buscar, en conjunto, un objetivo común. Una función de vuelo permite a las partículas el moverse en el espacio de búsqueda, tomando en cuenta el conocimiento propio de cada partícula, conocida como memoria, y además siguiendo la guía de un líder al establecer una comunicación indirecta con las demás partículas en el cúmulo.

El estudio de estas dos heurísticas es tema actual de investigación dentro del área del Cómputo Evolutivo y de la Inteligencia en Cúmulos. Dentro de los temas que actualmente se abordan se encuentran los siguientes:

- Estudio de los parámetros de cada técnica.
- Hibridización de la ED y PSO con métodos tradicionales de optimización.
- Nuevas variantes de los algoritmos originales.

- Mecanismos para evitar que el usuario deba calibrar los parámetros del algoritmo.

Por último, se invita al lector a discutir cómo podrían combinarse la ED y el PSO en un mismo algoritmo. Existen ya algunos intentos reportados en la literatura especializada [20], pero podrían ser mejorados.

5. Ejercicios propuestos

1. Mencione otros ejemplos de problemas de optimización (numérica o combinatoria) que puede encontrar en el mundo real.
2. Escriba el seudocódigo del algoritmo de Evolución Diferencial de la Figura 5 para la variante ED/rand/1/exp.
3. Para el ejemplo de funcionamiento de Evolución Diferencial (sección 2.4) cuáles serían los valores de los individuos después de la primera generación ($x_{1,1}$, $x_{2,1}$, $x_{3,1}$, $x_{4,1}$, $x_{5,1}$) utilizando los datos del ejemplo y $F = 0{,}5$.
4. Utilizando nuevamente el ejemplo de funcionamiento de Evolución Diferencial (sección 2.4) cuáles serían los valores de los individuos después de la primera generación utilizando los datos de la Tabla 16 y la variante *ED/best/1/bin*, nótese que $x_{4,0}$ es el mejor individuo en la población inicial.

Cuadro 16. Parámetros del Problema 4

Indiv.	r1	r2	r3	rand1	rand2	jrand
1	$x_{3,0}$	$x_{5,0}$	$x_{4,0}$	0.05	0.000025	1
2	$x_{3,0}$	$x_{1,0}$	$x_{4,0}$	0.887	0.039	1
3	$x_{1,0}$	$x_{2,0}$	$x_{4,0}$	0.048	0.015	2
4	$x_{3,0}$	$x_{5,0}$	$x_{4,0}$	0.00000021	0.48	1
5	$x_{2,0}$	$x_{1,0}$	$x_{4,0}$	0.0009	0.49	2

5. ¿Cuáles fueron los mejores valores encontrados después de una generación para los problemas 3 y 4?
6. Para el ejemplo de funcionamiento del PSO (sección 3.4) calcular los valores de las partículas después de la primera iteración utilizando los valores del ejemplo y $c2 = 0{,}1$.
7. Utilizando nuevamente el ejemplo de funcionamiento del PSO (sección 3.4) calcular los valores de las partículas después de la primera iteración utilizando los valores del ejemplo y $c2 = 0{,}7$.
8. ¿Cuáles fueron los mejores valores encontrados después de una generación para los problemas 6 y 7?

Referencias

1. Taha, H.A.: Investigación de Operaciones. Second edn. Alfa Omega (1991)
2. Deb, K.: Optimization for Engineering Design Algorithms and Examples. Fourth edn. Prentice Hall of India (2000)
3. Michalewicz, Z., Fogel, D.B.: How to Solve It: Modern Heuristics. Second edn. Springer-Verlag (2004)
4. Mezura-Montes, E., Coello Coello, C.A., Landa-Becerra, R.: Engineering Optimization Using a Simple Evolutionary Algorithm. In: Proceedings of the Fiftheenth International Conference on Tools with Artificial Intelligence (ICTAI'2003), Los Alamitos, CA, Sacramento, California, IEEE Computer Society (2003) 149–156
5. Angantyr, A., Aidanpää, J.O.: A Pareto-Based Genetic Algorithm Search Approach to Handle Damped Natural Frequency Constraints in Turbo Generator Rotor System Design. Journal of Engineering for Gas Turbines and Power **126**(3) (2004) 619–625
6. Alvarez-Gallegos, J., Villar, C.A.C., Flores, E.A.P.: Evolutionary Dynamic Optimization of a Continuously Variable Transmission for Mechanical Efficiency Maximization. In Gelbukh, A., Álvaro de Albornoz, Terashima-Marín, H., eds.: MICAI 2005: Advances in Artificial Intelligence, Monterrey, México, Springer (2005) 1093–1102 Lecture Notes in Artificial Intelligence Vol. 3789,.
7. Kirkpatrick, S., Jr., C.D.G., Vecchi, M.P.: Optimization by Simulated Annealing. Science **220**(4598) (1983) 671–680
8. Glover, F.W., Laguna, M.: Tabu Search. Kluwer Academic Publishers, London, UK (1997)
9. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. First edn. Springer (2003)
10. Engelbrecht, A.P.: Fundamentals of Computational Swarm Ingelligence. John Wiley & Sons, Sussex, England (2005)
11. Price, K.V., Storn, R.M., Lampinen, J.A.: Differential Evolution. First edn. Springer (2005)
12. Price, K., Storn, R.: Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Technical Report TR-95-012, ICSI, Berkeley University (1995) (Disponible en: http://www.icsi.berkeley.edu/~storn/litera.html).
13. Storn, R.: On the Usage of Differential Evolution for Function Optimization. NAFIPS (1996) 519–523
14. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Co., Reading, Massachusetts (1989)
15. Schwefel, H.P.: Evolution and Optimization Seeking. John Wiley & Sons, New York (1995)
16. Price, K.V.: An Introduction to Differential Evolution. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. Mc Graw-Hill, UK (1999) 79–108
17. Mezura-Montes, E., Velázquez-Reyes, J., Coello Coello, C.A.: A Comparative Study of Differential Evolution Variants for Global Optimization. In Keijzer, M., ed.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2006), Seattle, Washington, Morgan Kaufmann Publishers (2006) 485–492
18. Engelbrecht, A.: Fundamentals of Computational Swarm Intelligence. John Wiley & Sons (2006)

19. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. Proceedings of IEEE International Conference on Neural Networks, Piscataway (1995) 1942–1948
20. Munoz-Zavala, A.E., Hernández-Aguirre, A., Villa-Diharce, E.R., Botello-Rionda, S.: PESO+ for Constrained Optimization. In: 2006 IEEE Congress on Evolutionary Computation (CEC'2006), Vancouver, BC, Canada, IEEE (2006) 935–942

Capítulo 8.

Agentes inteligentes y sistemas multiagente

1. Introducción

Durante muchos años la investigación sobre la inteligencia artificial se ha orientado hacia las aplicaciones independientes con un conocimiento determinado y con una meta especifica. Estas aplicaciones se desarrollan en un ambiente estático y sus actividades principales son las siguientes: recopilar información, planear y ejecutar algún plan para lograr su meta. Esta metodología ha resultado insuficiente debido a la inevitable presencia de un número de entidades que cooperan en el mundo real. A finales de los años 1970 aparecen los primeros trabajos en Inteligencia Artificial Distribuida (IAD), aunque la primera reunión temática fue en 1980. Su objeto es el estudio de modelos y técnicas para resolución de problemas en los que la distribución, sea física o funcional, sea inherente.

Los sistemas de IAD se caracterizan por una arquitectura formada por componentes inteligentes y modulares que interactúan de forma coordinada. A estos componentes se les llaman agentes inteligentes. Los agentes de un sistema se conciben como independientes de un problema en concreto, autónomos y capaces de adaptarse a entornos dinámicos y se dotan al sistema de protocolos de comunicación a nivel de conocimiento suficientemente genéricos para lograr la interoperabilidad en sistemas compuestos por agentes que se conocen como Sistemas Multi-Agente (SMA). Las partes más importantes de un comportamiento inteligente incluyen aspectos sociales (que generalmente se han ignorado en la IA clásica) de las actividades en el mundo real. A diferencia del enfoque clásico, el enfoque de agentes desde el principio se ha centrado en estos aspectos y se considera como una vista social a la computación.

En este capítulo, nuestro principal objetivo será introducir los principales conceptos de interacción en SMA e indicar como estos están relacionados con las arquitecturas de agentes inteligentes. Iniciaremos su estudio con el análisis

de las propiedades y tipos de agentes. En la sección 3 presentaremos los modelos de interacción en SMA constituidos por agentes intencionales llamados agentes BDI. Posteriormente, analizaremos otros aspectos de comportamiento social de agentes: modelos de coordinación, negociación, formación de coaliciones y comunicación. En la sección 6 introducimos un modelo de aprendizaje colectivo. Finalmente, presentaremos algunos lenguajes y entornos de desarrollo de SMA que implementen los modelos analizados en desarrollos prácticos de software basado en agentes.

2. Agentes inteligentes

Desde los principios de nuestra época, los seres humanos se han fascinado con la idea de agencias no humanas. Por una parte, los androides, humanoides, robots, cyborgs y otras criaturas de la ciencia ficción han penetrado en nuestra cultura y han creado una base psicológica en la sociedad (pero en la mayoría de los casos negativa). Por otra parte, la tecnología siempre ha tratado de convertir estas ideas en la realidad, empezando por robots. Sin embargo, recientemente ha tenido que cambiar de hardware a software, de los átomos que comprenden un robot mecánico a los bits que integran un agente de software [4].

2.1 Antecedentes históricos

Tal vez debería mencionarse que la idea de un agente originó con John McCarthy[1] en 1950, y el término fue inventado por Oliver G. Selfridge unos pocos años después, cuando ambos estaban en el Instituto Tecnológico de Massachussets (MIT). Ellos tenían proyectado *un sistema que, cuando le dan una meta, podría llevar a cabo los detalles de las operaciones de cómputo apropiadas y podrían pedir y recibir un consejo, ofrecido en términos humanos, cuando este sea introducido.* Sin embargo, por varias décadas la idea de agentes quedó abandonada a favor de otras líneas de investigación en la IA. Fue hasta finales de los años 1970 que la IAD llamó la atención de los investigadores en la IA. La IAD se puede definir de la siguiente manera:

Definición 1: *La IAD es un subcampo de la IA que intenta construir un modelo del mundo poblado por entidades inteligentes que interactúan por medio de la cooperación, la coexistencia o por la competición.*

Como lo observamos en otros capítulos de este libro, inteligencia es un fenómeno con muchas facetas; de allí surgen diferentes perspectivas de su estudio. Un enfoque de mayor aceptación, que a veces se llama el enfoque

[1] Al quien se le atribuye el termino "Inteligencia Artificial".

clásico es el enfoque simbólico que busca la replicación de la inteligencia reconstruyendo el proceso de la cognición y tiene sus raíces en lógica, lingüística, psicología cognoscitiva y filosofía (**Fig. 1**). Según éste enfoque la estructura interna del agente se basa en la representación simbólica del mundo (representación del conocimiento) y en el manejo de esta representación en términos de procesamiento de símbolos.

La Resolución Distribuida de Problemas (RDP) forma el área primaria dentro del enfoque lógico cognoscitivo a la construcción de sistemas artificiales distribuidos. RDP considera el como dividir una tarea particular entre un número determinado de generadores de soluciones a problemas parciales (agentes inteligentes) que compartiendo su conocimiento podrían "cooperar" en la construcción evolutiva de una solución. Los sistemas para RDP se caracterizaban por una forma de actuación concurrente en los diferentes nodos de una red, en general con control centralizado. Ejemplo paradigmático de la comunicación y coordinación para RDP es la arquitectura de pizarra del sistema Hearsay.

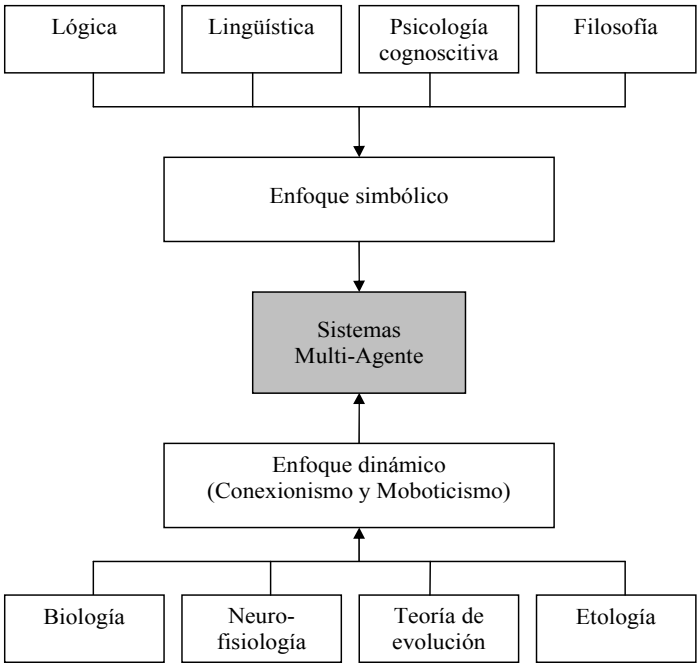


Fig. 1. Principales predecesores de sistemas multi-agente

El trabajo por Hewitt y sus colegas [21] en lenguajes de actores era una aplicación significante de modelos de comunicación entre entidades de solución

de problemas basados en la transmisión de mensajes. Un actor puede definirse como "*un agente computacional que lleva a cabo sus acciones en respuesta a procesar una comunicación*". Las acciones que puede realizar son: enviar comunicación a otros actores, crear más actores, etc. A diferencia con los agentes clásicos de la RDP donde cada nodo es un "solucionador de problemas", los actores de Hewitt son entidades simples que cooperan en la construcción de una solución. Este planteamiento se asemeja con los modelos desarrollados dentro del otro enfoque: enfoque dinámico.

Según el enfoque dinámico que tiene sus raíces en biología, neurofisiología, teoría de evolución y etología (estudio de comportamiento de sistemas), la idea principal es el estudio de inteligencia como producto emergente de la interacción de un sistema dinámico complejo con el ambiente. Estos sistemas por lo general se forman de una diversidad de procesos (agentes) simples que interactuando entre si y con el ambiente generan un comportamiento inteligente [25]. En consecuencia, dado que el comportamiento en principio es reflexivo y no es resultado de la deliberación, puede desarrollarse sin una representación explicita de conocimiento [6]. Estas ideas también se desarrollan dentro del área que se llama *Vida Artificial*, término atribuido a fines de los años 1980 a Christopher Langton, como el estudio de la vida y de los sistemas artificiales que exhiben propiedades similares a los seres vivos, a través de modelos de simulación.

La IA paralela parte de una idea similar de que la mente humana es como una red de miles de millones de agentes trabajando en paralelo. Para producir la verdadera inteligencia artificial, esta escuela sustenta que se deben construir sistemas de cómputo que contengan muchos agentes y sistemas para arbitrar entre los resultados competentes de agentes.

Al principio de los años 1990 la investigación en la IAD se concentra en el estudio de agentes inteligentes y SMA donde diferentes enfoques de inteligencia artificial se han convergido. En ésta época han surgido numerosas definiciones de agentes y SMA que han dado lugar a las definiciones comúnmente aceptadas por la comunidad actual de investigadores en agentes:

Definición 2: *Un agente inteligente es una entidad, situada en algún ambiente, que es capaz de actuar de manera autónoma y flexible para satisfacer los objetivos de diseño.*

Definición 3: *Un sistema multi-agente es un sistema donde varios agentes inteligentes interactúan para perseguir un conjunto de metas o ejecutar tareas que están fuera de su alcance (capacidad) individual.*

Diferentes enfoques de IA nos llevan a diversos tipos de agentes que se distinguen por los tipos de comportamiento y formas de generarlo. Estos aspectos se consideran en la próxima sección.

2.2 Propiedades de agentes inteligentes

La definición de un agente inteligente (**Def. 2**) implica que para poder llamarse inteligente, un agente debe tener ciertas propiedades. Los requisitos para su autonomía derivan de nuestro deseo de que un agente sea capaz de realizar actividades de una manera flexible e inteligente en respuesta a los cambios en el entorno sin requerir constante dirección o intervención humana. Idealmente, un agente que funciona continuamente en un entorno por un largo período de tiempo debería ser capaz de aprender de sus experiencias. Un agente que habite en un entorno con otros agentes y procesos, debería ser capaz de comunicarse y cooperar con ellos, y quizá moverse de un lugar a otro. En base a estos con estos requerimientos, cada agente puede poseer (en un mayor o menor grado) los siguientes atributos [4] (**Fig. 2**):

1. Reactividad: la habilidad para sentir selectivamente y actuar. Los agentes perciben su ambiente (el cual puede ser el mundo físico, un usuario a través de una interfaz gráfica, una colección de otros agentes, o quizás todas estas combinadas) y responden de una manera oportuna frente a los cambios que ocurren en el.
2. Autonomía: los agentes operan sin la intervención directa de los seres humanos u otros, y tienen alguna clase de control sobre sus acciones y el estado interno. No simplemente actúan como respuesta a su ambiente, son capaces de exhibir por iniciativa propia un comportamiento dirigido por metas.
3. Comportamiento colaborativo: los agentes interactúan y pueden trabajar en concierto con otros agentes para lograr una meta en común.
4. Habilidad de comunicación a nivel de conocimiento: la habilidad para comunicarse con otros agentes en un lenguaje de comunicación de agentes (ACL) más parecido a los actos del habla humana que a los típicos protocolos a nivel simbólico de comunicación entre software.
5. Capacidad de inferencias: pueden actuar en especificación de tareas abstractas usando un conocimiento previo de metas generales y métodos preferidos para lograr flexibilidad. Esto puede requerir modelos explícitos de sí mismos, usuarios, situaciones y/o otros agentes.
6. Continuidad temporal: persistencia de identidad por largos períodos de tiempo.
7. Personalidad: la capacidad para manifestar los atributos de un carácter "creíble"' tales como las emociones.
8. Adaptividad: ser apto para aprender e improvisar con experiencias.
9. Movilidad: ser apto para migrar de una plataforma anfitriona a otra.

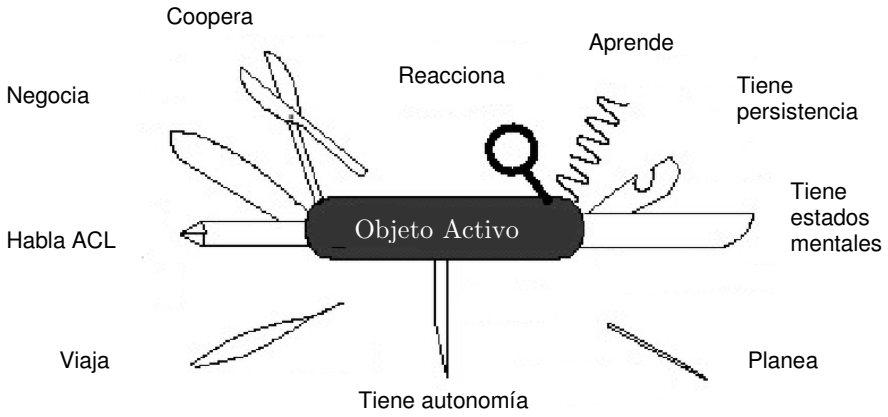


Fig. 2. Composición de un agente inteligente social (adaptada de [44])

Un artículo de IBM [16] describe a los agentes inteligentes en términos de un espacio definido por tres dimensiones de agencia, inteligencia, y movilidad (**Fig. 3**). *Agencia* es el grado de autonomía y autoridad establecida en el agente, y puede ser medida cualitativamente por la naturaleza de la interacción entre el agente y otras entidades en el sistema. Como mínimo, un agente debe correr de manera asíncrona. El grado de agencia es mayor si un agente representa un usuario de alguna manera. Un agente más avanzado puede interactuar con datos, aplicaciones, servicios u otros agentes. *Inteligencia* es el grado de razonamiento y conducta aprendida: la habilidad del agente para aceptar las declaraciones de las metas del usuario y llevar a cabo las tareas encomendadas. Como mínimo, estas pueden ser algunas declaraciones de preferencias. Los niveles más altos de inteligencia son un modelo de usuario y de otros agentes, así como la capacidad de razonamiento. Más allá de la escala de inteligencia están los sistemas que planean sus acciones para alcanzar metas, aprenden y se adaptan a su medio ambiente, ambos en términos de los objetivos del usuario y en términos de los recursos disponibles del agente.

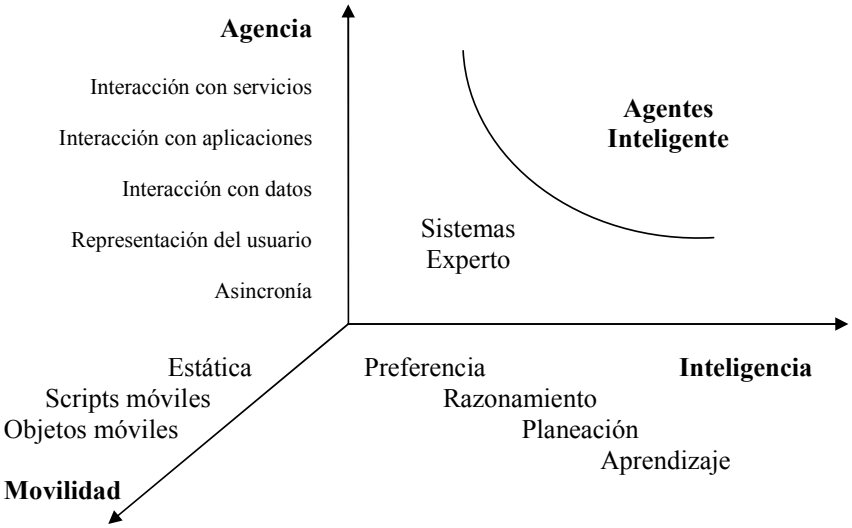


Fig. 3. Agentes inteligentes en el espacio de agencia, inteligencia y movilidad

Por último, *movilidad* es el grado por el cual los agentes por si mismos viajan a través de la red. Los *scripts* móviles pueden ser compuestos en una máquina y desarmados en otra para su ejecución. Los objetos móviles se transportan de máquina a máquina durante la ejecución junto con el estado de datos acumulado en ellos.

2.3 Tipos básicos de agentes

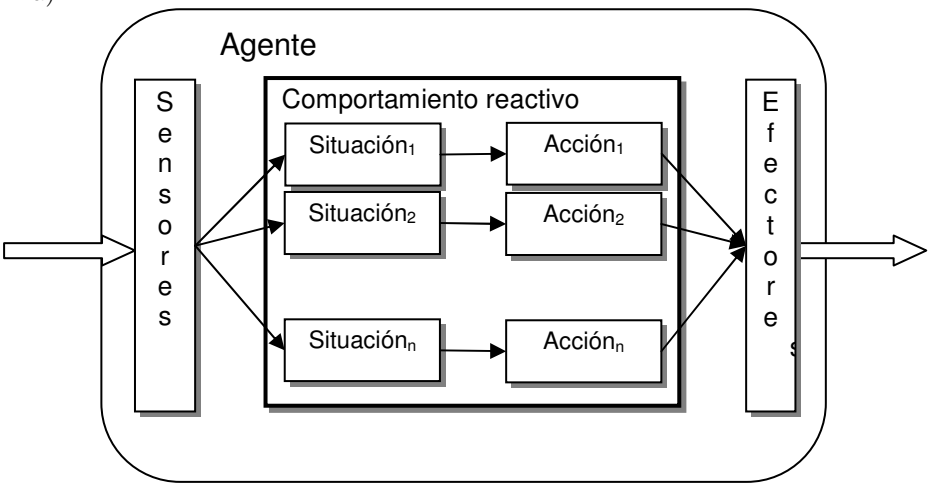
Moulin y Chaib-draa [24] han caracterizado a los agentes por el grado de capacidad de solución de problemas y han definido tres tipos de agentes (**Fig. 4**).

Agentes Reactivos. Su conceptualización esta inspirada en modelos del enfoque dinámico. Estos agentes son diseñados para reaccionar ante los cambios en su ambiente (a través de la recepción de un mensaje o algún evento): su inteligencia proviene de la interacción con el ambiente. Los agentes reactivos tienen una representación interna del mundo muy simple: realizan acciones en respuesta a la activación de reglas situación-acción. Estos agentes están limitados por su inhabilidad para construir algún plan de acciones, es decir alcanzar una meta.

Agentes Deliberativos. Su conceptualización esta inspirada en el enfoque simbólico, es decir, en la idea de que un comportamiento inteligente se genera

en un sistema por medio de una representación simbólica de su ambiente y comportamiento deseado, y manejo sintáctico de esta representación. Un agente deliberativo es capaz de razonar acerca de las metas de su diseño: i) seleccionar cuáles metas desea satisfacer, ii) cómo satisfacer dicha meta (por planeación), y iii) más adelante, modificar sus acciones en caso de que su plan falle. Agentes deliberativos que utilizan el método de razonamiento práctico basado en el análisis de creencias, deseos e intenciones se conocen como *Agentes Intencionales o Agentes BDI* (por las siglas, en ingles, Belief-Desire-Intention - creencias, deseos e intenciones). A diferencia del razonamiento teórico dirigido a inferior nuevo conocimiento (nuevas creencias), el razonamiento práctico es el razonamiento dirigido a la acción.

a)



b)

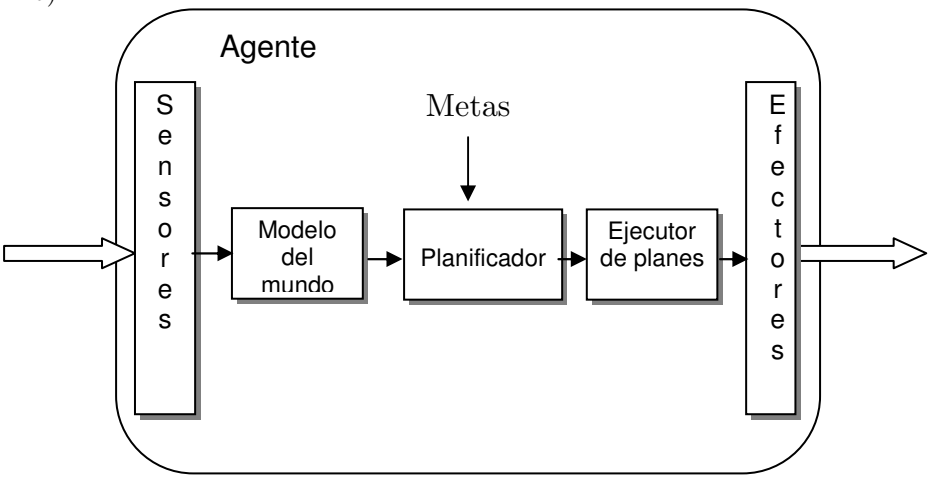


Fig. 4. Arquitecturas de agentes: a) agentes reactivos, b) agentes deliberativos

Agentes Sociales. Los agentes sociales no sólo son capaces de razonar acerca de sí mismos, sino además acerca de otros agentes que conocen. Estos agentes son de este modo capaces de modificar su propia conducta ajustándose no solo a sus metas locales pero también a las metas globales del sistema. Por lo general, los agentes sociales extienden su arquitectura interna deliberativa con los componentes que les permiten interactuar con los demás agentes.

Los aspectos sociales de las actividades en el mundo real representan las partes más importantes de un comportamiento inteligente. La interacción de alto nivel entre agentes dentro de un contexto organizacional es considerada como una propiedad esencial de la computación basada en agentes [19]. Es por eso que en la investigación sobre agentes recientemente se ha hecho más énfasis en los aspectos de comportamiento social de agentes [10,34,42]. En el resto del capítulo analizaremos los componentes esenciales de comportamiento social empezando por un modelo de interacción entre los agentes intencionales.

3. Sistemas multi-agente con agentes intencionales

Para construir agentes racionales, predominan los modelos de agentes deliberativos, particularmente los modelos BDI basados en un modelo filosófico de razonamiento práctico introducido por Bratman [5]. Modelos BDI plantean que un agente debe ser conceptualizado y/o implementado usando conceptos y nociones o estados mentales usualmente asociados a los humanos, como por ejemplo creencias, deseos, intenciones, obligaciones, compromisos, etc. Sus principales componentes son:

- *Beliefs (Creencias)*: Son sentencias que un agente toma como verdaderas (que a diferencia del conocimiento pueden ser falsas) acerca de propiedades de su mundo (y de sí mismo). Las creencias intentan capturar el estado de información del agente.
- *Desires (Deseos)*: Son acciones que un agente desea realizar o situaciones que prefiere y quiere lograr. Los deseos intentan capturar el estado de motivación del agente. Deseos también se conocen como *objetivos o metas - goals*), pero los objetivos involucran cierto grado de compromiso del agente en su realización y que el conjunto de objetivos perseguidos sea consistente. En modelos BDI este compromiso se asocia con intenciones.
- *Intentions (Intenciones):* Acciones factibles, planes o situaciones deseadas que el agente ha seleccionado y se ha comprometido a realizar o lograr. Las intenciones intentan capturar el estado deliberativo del agente.

De esta forma, las creencias capturan actitudes informativas, deseos - actitudes de motivación, y las intenciones - actitudes deliberativas de agentes. Modelo BDI considera como fundamental dos procesos de razonamiento: Deliberación y Razonamiento de Medios y Fines. La *deliberación* es razonamiento orientado a decidir qué cosas o situaciones alcanzar. Esta actividad usualmente involucra la ponderación de alternativas competitivas. El *razonamiento de medios y fines* es el proceso que determina cómo alcanzar las situaciones u objetivos determinados por la deliberación. El resultado de este proceso es un plan de acciones para lograr la situación elegida. Para la ejecución de una acción deberán de satisfacerse determinadas (pre)condiciones que hagan factible su ejecución. La obtención de metas como resultado de la ejecución de acciones modifica tanto el estado mental de cada agente como el mundo real.

Cohen y Levesque [7] y, por otra parte Rao y Georgeff [28], han adoptado este modelo y lo transforman en una teoría formal y un modelo de ejecución para agentes de software, basado en las nociones de creencias, planes y metas. En lo restante de esta sección describiremos las principales modalidades de creencias, metas y acciones dentro de una teoría de agentes: un formalismo lógico para razonar acerca de agentes y sus propiedades expresadas en este formalismo.

3.1 Formalización de agentes BDI

Para definir los modelos BDI se utilizan lógicas temporalmente arborescentes que se construyen a partir de los sistemas lógicos modales y temporales. Estas lógicas tienen un componente de primer orden para representar a los objetos en el universo del discurso, sus propiedades y sus relaciones. El componente BDI para representar creencias, deseos e intenciones, se basa en operadores modales. Las lógicas BDI incluyen además un componente temporal para representar y razonar sobre los aspectos dinámicos del sistema y como éstos cambian sobre el tiempo. Las lógicas computacionales de árbol (CTL) han sido usadas con este propósito. Finalmente, las lógicas BDI pueden incluir un componente de acción para representar los eventos registrados por los agentes y sus acciones. Este componente se basa normalmente en la lógica dinámica, o se define usando formulas de estado para expresar la ocurrencia de eventos. A continuación presentamos el modelo desarrollado en [1] para el lenguaje modal de la lógica de interacción L_{int}.

Sea Ag={1,...,n} el conjunto de todos los agentes de un sistema, siendo $i,j \in Ag$. Un agente esta definido por la 3-tupla $<\beta_i, \Im_i, \chi_i>$, donde β_i son las creencias del agente (precondiciones), $\Im_i$ el conjunto de metas (postcondiciones) y χ_i el conjunto de acciones que el agente puede ejecutar.

3.1.1 Semántica de creencias

La formula φ es una creencia $B_i\varphi$ para un agente i en un mundo w, sii en todo mundo w' relacionado mediante creencias con w, φ es valida. Los mundos relacionados por creencias están determinados por la relación parcial $R^{Bi} \subseteq W \times W$, donde R^{Bi} es una relación reflexiva, antisimetrica, transitiva y serial. Esto se define a continuación.

$$M, w| = \ B_i\varphi \text{ sii } \ \forall(w, w') \in R^{Bi} \ \text{ tal que } M,w' \ |= \varphi. \quad (\mathbf{1})$$

La buena formación de formulas para creencias entre dos agentes usando conectivos lógicos es la siguiente.

$$\begin{gathered} M,w|=B_i\varphi \wedge B_j\varphi \text{ sii } \ M,w \ |=B_i\varphi \text{ y } M,w|= B_j \ \varphi \\ M,w|=B_i\varphi \vee B_j\varphi \text{ sii } \ M,w \ |=B_i\varphi \text{ o } M,w|= B_j \ \varphi \\ M,w|=B_i\neg\varphi \text{ sii } \forall w^i \ \text{ donde } (w, w') \in \ R^{Bi} \text{ tal que } M, w'|=\neg\varphi \\ M,w|=\neg B_i\varphi \text{ sii } \exists \ w^i \ \text{ donde } (w, w') \in \ R^{Bi} \text{ tal que } M, w'|\neq\varphi \\ M,w|=B_i\varphi \rightarrow B_j\psi, \text{ por definición, se tiene } \ M,w|=\neg B_i\varphi \vee B_j\psi \end{gathered} \quad (\mathbf{2})$$

Es decir $\exists w'$ donde $(w, w') \in R^{Bi}$ tal que $M, w'|\neq\varphi$ o $M,w|=B_j\psi$.

Cabe recordar que cada agente (i y j) tiene sus propios mundos accesibles por creencias. Por lo tanto φ se debe evaluar en el mundo correspondiente al agente i y ψ para el agente j.

$$M,w|=B_i\varphi \leftrightarrow B_j\psi \text{ sii } M,w|=(B_i\varphi \rightarrow B_j\psi) \ \wedge(\ B_j\psi \rightarrow B_i\varphi \ ) \quad (\mathbf{3})$$

Los axiomas para creencias se presentan en la **Tabla 1**.

Tabla 1. Axiomas para creencias

$M, s \models (B\varphi \wedge B(\varphi \rightarrow \psi)) \rightarrow B\psi$	K	Modus Ponens (Distributividad)
$M, s \models \varphi \rightarrow B\varphi$	D	
$M, s \models B\varphi \rightarrow B(B\varphi)$	4	Introspección Positiva
$M, s \models \neg B\varphi \rightarrow B(\neg B\varphi)$	5	Introspección Negativa

3.1.2 Semántica de metas

La formula φ es una meta para un agente i en un mundo w, sii existe un mundo w' relacionado con w en el cual φ es verdadera. Los mundos relacionados por metas están determinados por la relación parcial $R^{Gi} \subseteq W \times W$, donde R^{Gi} es una relación antisimetrica, no reflexiva y no transitiva. En general la semántica de metas se define como sigue:

$$M,w|= G_i\ \varphi \ \text{ sii } \exists(w, w') \in R^{Gi} \ \text{ tal que } M,w' |= \varphi. \quad (4)$$

Las metas han de ser verdaderas en al menos uno de los mundos posibles dado que es necesaria la realización de una acción, bajo ciertas precondiciones, para lograr la meta que se desprende de la ejecución de tal acción. Puede ser que ejecutando la misma acción, bajo precondiciones distintas, se logre la meta pero en un mundo distinto. La buena formación de formulas para metas entre dos agentes usando conectivos lógicos es la siguiente.

$$M,w|=G_i\varphi\wedge G_j\varphi \text{ sii } \ M,w |=G_i\varphi \text{ y } M,w|= G_j\ \varphi \quad (5)$$

$$M,w|=G_i\varphi\vee G_j\varphi \text{ sii } \ M,w |=G_i\varphi \text{ o } M,w|= G_j\ \varphi$$

$$M,w|=G_i\neg\varphi \text{ sii } \exists\ w^i \ \text{ donde } (w, w')\in\ R^{Gi} \ \text{ es tal que } M, w'|=\neg\varphi$$

$$M,w|=\neg G_i\varphi \text{ sii } \forall\ w^i \ \text{ donde } (w, w')\in\ R^{Gi} \ \text{ es tal que } M, w'|\neq\varphi$$

$$M,w|=G_i\varphi\rightarrow G_j\psi, \text{ por definición, se tiene } \ M,w|=\neg G_i\varphi\vee G_j\psi$$

$$M,w|=Gi\varphi\leftrightarrow Gj\psi \text{ sii } M,w|=(Gi\varphi \rightarrow Gj\psi) \ \wedge(\ Gj\psi \rightarrow Gi\varphi\ )$$

Los axiomas para metas se presentan en la **Tabla 2**.

Tabla 2. Axiomas para metas.

$M, s \models (G\varphi \wedge G(\varphi \rightarrow \psi)) \rightarrow G\psi$	K	Modus Ponens (Distributividad)
$M, s, t \models \varphi \rightarrow \neg G\varphi$		
$M, s \models G\varphi \rightarrow G(G\varphi)$	4	Introspección Positiva
$M, s \models \neg G\varphi \rightarrow G(\neg G\varphi)$	5	Introspección Negativa

El axioma de distributividad en metas conlleva la existencia de una ruta en el orden parcial de mundos posibles, en la cual, dado que determinadas metas han sido alcanzadas, también lo serán, *eventualmente,* las metas que se desprendan de las ya alcanzadas. En este sentido, si bien no se define en este trabajo un operador "eventualmente", la similitud es directa.

3.1.3 Semántica de Acciones

Sea χ_i el conjunto de acciones, donde $a,b\in\ \chi_i$, el agente i ejecuta la acción a, en el mundo w, para alcanzar la meta φ si existe un mundo relacionado por acciones, en el cual al realizar a se obtiene φ.

$$M,w \models [do_i\ a]\varphi \text{ iff } M,w \models G_i\varphi \text{ y } \exists(w, w') \in R^{Doi} \text{ tal que } M,w' \models \varphi. \quad (6)$$

Hay que aclarar que se asume que en el mundo w se satisfacen las precondiciones necesarias para ejecutar la acción a; si esto no es así deberán lograrse antes. Por otro lado, las post-condiciones son el resultado de la ejecución de las acciones.

La buena formación de formulas entre dos agentes usando conectivos lógicos es la siguiente.

$$\begin{array}{l} M,w \models [do_i\ a]\varphi \wedge [do_j\ b]\psi \text{ sii } M,w \models [do_i\ a]\varphi \text{ y } M,w \models [do_j\ b]\ \psi \\ M,w \models [do_i\ a]\varphi \vee [do_j\ b]\psi \text{ sii } M,w \models [do_i\ a]\varphi \text{ o } M,w \models [do_j\ b]\ \psi \\ M,w \models [do_i\ a]\neg\varphi \text{ sii } \exists\ (w, w') \in R^{Doi} \text{ tal que } M, w' \models \neg\varphi \\ M,w \models \neg[do_i\ a]\varphi \text{ sii } \forall w^i \text{ donde } (w, w') \in R^{Doi} \text{ tal que } M, w' \neq \varphi \\ M,w \models [do_i\ a]\varphi \rightarrow [do_j\ b]\psi \text{ sii } M,w \models \neg\ [do_i\ a]\varphi \vee [\ do_j\ b]\psi \\ M,w \models [do_i\ a]\varphi \leftrightarrow [do_j\ b]\psi \\ \text{sii } M,w \models \quad [do_i\ a]\varphi \rightarrow [\ doj\ b]\psi \text{ y } M,w \models [\ do_i\ b]\psi \rightarrow [do_i\ a]\varphi. \end{array} \quad (7)$$

La combinación de formulas de creencias, metas y acciones, utilizando los conectivos $\neg$, $\vee$, $\wedge$, $\rightarrow$ y $\leftrightarrow$ es el lenguaje modal de la lógica de interacción L_{int} que utilizaremos en lo sucesivo. Por tanto este lenguaje puede involucrar, en una misma fórmula, creencias, metas y acciones.

3.2 Interacción entre agentes BDI basada en roles

Cuando un agente convive con otros agentes en un ambiente compartido, el concepto de acción adquiere un nuevo significado. El agente ya no puede ser considerado una entidad aislada y única dentro de un ambiente que responde exclusivamente a sus acciones y debe involucrarse para completar sus objetivos en las interacciones sociales con otros agentes, y que puedan operar dentro de las estructuras de organización flexibles (**Fig. 5**). Ahora el ambiente incluye explícitamente a otros agentes, que pueden compartir o no sus intereses. Estos agentes también actúan y las influencias recíprocas que tienen sus acciones (interacciones), necesitan ser consideradas explícitamente. Es a partir de la complejidad y flexibilidad de estas interacciones, que los SMA obtienen un valor adicionado como un todo, que no se podría lograr si los agentes trabajaran en forma aislada. Se podría afirmar por lo tanto que así como el concepto de acción constituye la base de la metáfora de agentes, la idea de interacción en un contexto organizacional es la fundamental de los SMA.

A continuación se definen la estructura de interacción o escenario y de éste, los *roles* y las *acciones* que realizan los agentes para lograr sus *metas*. El conjunto de metas organizadas conforman el *plan grupal* del escenario, el cual persigue una meta. La idea clave del modelo es la manera en la cual se lleva a

cabo el plan a través de la interacción de los roles que realizando acciones van alcanzando las metas parciales del plan.

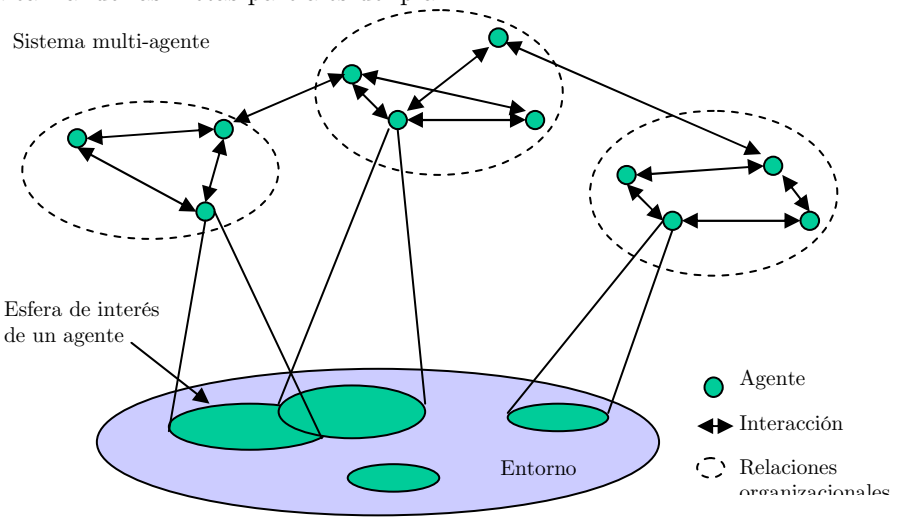


Fig. 5. Vista genérica de un sistema multiagente

Para lograr cualquier meta del plan, los agentes coordinan acciones, interactuando entre sí conforme las relaciones jerárquicas ó de preeminencia bajo las cuales se organicen en el escenario. Puede haber distintos planes para un mismo escenario tales que logren resultados equivalentes. Un mismo rol dentro del escenario puede ser ejecutado por uno o varios agentes; asimismo, varios agentes pueden tomar solo un rol. Un rol realiza acciones para lograr sus metas particulares, ó bien, para lograr una meta, que es parte de una meta conjunta con otros roles. En cualquiera de los casos, el rol, la acción que se ejecute y la meta que se logre, son partes de una *interacción*, la cual ocurre en un estado y tiempo dados de la estructura de interacción o escenario. Para la formalización de un interacciones entre agentes (roles) se han de considerar,

- La estructura de interacción o escenario
- El funcionamiento *social* del escenario.

Enseguida se pasa a formalizar elemento de interacción, interacción y la estructura de interacción en un contexto de lógica modal epistémico.

3.2.1 Estructura de interacción

Sean Ag un conjunto de agentes que tomando roles interactúan en el escenario, R el conjunto de roles tomados por los agentes y T el conjunto de instantes de tiempo; Ac es el conjunto de acciones que pueden ser ejecutadas por los roles, las cuales son proposiciones (predicados de primer orden), W el

conjunto de estados (mundos, situaciones), G el conjunto de creencias que son proposiciones.

Para la formalización de una interacción se han de considerar los elementos que intervienen, a saber, el rol, la acción que éste ejecuta y la meta que se logra como resultado de ejecutar la acción; a la tríada (rol, acción, meta) se le llama *elemento de interacción*. Sea,

$$\mathrm{I} = (\mathrm{RxAcxG}), \tag{8}$$

el conjunto de elementos de interacción. Luego, las *interacciones* I son el n-producto cartesiano de los n elementos de interacción que intervienen en una interacción, dado un tiempo y estado del escenario,

$$I = \mathrm{I}^n \mathrm{x\ W\ x\ T}. \tag{9}$$

Sea Υ = (W, T, I) una *estructura de interacción*, la cual es una estructura de Kripke. La estructura de interacción está formada por los elementos que soportan la actuación e interacción social de los roles en el escenario.

El modelado dinámico Ω de las interacciones que ocurren sobre la estructura de interacción corresponde a un conjunto de metas (acciones) parcialmente ordenadas P=G/$\angle$. El orden parcial $\angle$ indica las secuencias en que los roles realizarán las acciones para lograr sus metas, y así cumplir con el plan. En Ω se distinguen los elementos participantes en la dinámica de interacción:

$$\Omega = < w_0, \mathrm{P}, \mathrm{g} >, \tag{10}$$

donde, $w_0 \in \mathrm{W}$ es el mundo donde se satisfacen las creencias iniciales de los roles, $\bigcup_{i=0}^{n} R_{\psi i}$ donde $r_{\psi i}$ son las creencias del rol *i-esimo;* $g \in G$ es la meta final (principal) del escenario, y P es el conjunto de metas parcialmente ordenadas que determinan el plan que se realiza dadas las condiciones de partida en w_0 hasta el logro de la meta g. P es una estructura arborescente que puede variar.

3.2.2 Comportamiento social

Ferber en [11], define un escenario de interacción "*... como un ensamble de comportamientos resultantes de una agrupación de agentes que tienen que actuar para alcanzar sus objetivos con la atención en limitaciones de recursos disponibles y en sus capacidades individuales.*" El aspecto interesante de esta definición es que claramente destaca los criterios principales que deben ser considerados al caracterizar un escenario de interacción:

- Objetivos de los agentes (¿compatibles o incompatibles?)
- Recursos disponibles (¿suficientes o insuficientes?)

– Capacidades de los agentes en relación a las tareas (¿pueden realizar las tareas solos o no?)

En el funcionamiento social del escenario se considera lo referente a las políticas y normas que rigen las interacciones de los roles dentro del escenario. Por tanto, deberán reflejarse también las relaciones de jerarquía o de preeminencia que haya entre los roles, dependiendo del escenario de que se trate. Si el escenario es el de una organización establecida (empresa, industria, etc.) las relaciones son de jerarquía entre jefes y subordinados; si el escenario es un evento que se modela, las relaciones pueden ser de preeminencia. Por ejemplo, en el escenario de un diálogo, el rol que tiene preeminencia es el de quien está hablando; si el escenario es el de un juego de pelota, el rol preeminente pudiera ser el que representa al jugador que tiene el balón en su poder, etc.

Las relaciones de jerarquía o preeminencia entre los roles son modeladas, de manera general por la estructura *S,* mediante el orden parcial $\leq$; así, *S* equivale a $R/\leq$. Por otro lado, *Social* es la estructura

$$Social = < I,\ S >, \qquad (\mathbf{11})$$

que integra a la estructura de interaccione *I* previamente definida junto con *S*. De esta manera, las interacciones entre los roles (agentes), atendiendo sus relaciones de jerarquía o relevancia respecto a los otros roles es modelada. Cabe mencionar, sin embargo, que el aspecto de interacciones sociales entre agentes es más complejo y debe incluir normas, compromisos, obligaciones, derechos, permisos y responsabilidades [10].

El resultado de una interacción son las metas logradas por cada rol luego de la interacción y determinan las *post-condiciones* que se satisfacen (proposiciones verdaderas) en algún subconjunto de estados de *W*.

De lo dicho en los párrafos anteriores se implica tener un balance entre los mecanismos para la revisión de creencias y los efectos derivados de la ejecución de las acciones. Además hay que tomar en cuenta que las acciones que se ejecutan en el escenario pueden afectarse entre sí, por lo tanto habrá de que realizar una verificación antes de llevar a cabo una interacción. De forma general, las situaciones que se pueden presentar y que originan la revisión de creencias son:

Caso 1: Cuando en un agente no existan las precondiciones necesarias para ejecutar una acción por lo tanto tendrá que expandir el conjunto β_i y obtener un conjunto β_i' tal que $\beta_i \subseteq \beta_i'$ y en β_i' se cuenten con dichas precondiciones.

Caso 2: En las precondiciones de dos agentes, de cuya interacción se pretende lograr una meta γ (véase ejemplo en la sección 3.3) se presenta inconsistencia, pero esta inconsistencia se puede eliminar tomando un subconjunto de las creencias de estos agentes. Así, si podemos obtener los subconjuntos β_i' y β_j' tal que $\beta_i' \subseteq \beta_i$, $\beta_j' \subseteq \beta_j$ y $\beta_I' \cup \beta_j' \not\vdash \perp$(no existen

inconsistencias) y en la unión de los subconjuntos se tienen la precondiciones suficientes para alcanzar la meta φ, β_I'∪β_j' /— φ.

Caso 3: Las precondiciones de los agentes *i,j* presentan inconsistencias y no existen los conjuntos β_i',β_j' tal que de su intersección se obtenga un conjunto consistente tal que sea suficiente para derivar de él la meta; entonces se tiene que emplear un mecanismo de revisión de creencias mas complejo, el cual consistirá en la eliminación de la información que provoca la inconsistencia, seguido del incremento del conjunto de creencias con las cuales pueda derivarse la meta. Este proceso consiste en obtener β_i', β_j' tal que β_i'⊆β_i , β_j'⊆β_j y posteriormente β_i'⊆β_i'' , β_j'⊆β_j'' tal que β_i''∪β_j''/—φ

3.2.3 Agentes y Roles

Sea β el conjunto de toda formula que puede ser una creencia, ℑ, el conjunto de toda formula que pueda ser una meta y χ, el conjunto de todas las acciones sobre las cuales al ejecutarlas se pueden alcanzar las metas, decimos que un rol *r* pertenece al producto cruz formado por el conjunto potencia de las creencias, conjunto potencia de las metas y conjunto potencia de las acciones:

$$r \in 2^{\beta} \times 2^{\Im} \times 2^{\chi}; \qquad (\mathbf{12})$$

igualmente, un agente *i*, es elemento de este conjunto:

$$i \in 2^{\beta} \times 2^{\Im} \times 2^{\chi} \qquad (\mathbf{13})$$

Un rol esta conformado por creencias (precondiciones) que es la información que maneja en ese mundo ó estado, de metas (objetivos), que es lo que pretende lograr y un conjunto de acciones, las cuales al ejecutarlas como parte de un plan, va consiguiendo sus metas. Un rol es representado por una 3-tupla r=<β_r, $\Im_r$,χ_r> tal que:

β_r.- Conjunto de creencias (precondiciones)

$\Im_r$.- Conjunto de Metas

χ_r.- Conjunto de las acciones del rol.

Un agente *i* esta conformado por un conjunto de roles, r_1,r_2...r_n; el agente en si mismo es, a su vez, una 3-tupla donde sus elementos son, respectivamente, conjunto de creencias, metas, y acciones, representado por *i*=<β_i, $\Im_i$, χ_i>. La definición de cada elemento de la 3-tupla es la siguiente:

$\beta_i = \bigcup_{k=1}^{n} \beta_{rk}$ Representa la unión de todas las creencias de los roles que conforman al agente, que ahora son creencias del agente.

$\Im_i = \bigcup_{k=1}^{n} \Im_{rk}$ Representa el conjunto de metas de los roles, que representan submetas y metas (estado interno) dentro de un agente.

$\chi_i = \bigcup_{k=1}^{n} \chi_{rk}$ Representa el conjunto de acciones capaces de efectuar el agente.

La relación entre un agente y un rol que toma el agente está definida mediante *i/r:*

Definición 4. *Sea un agente* $i=<\beta_i, \Im_i, \chi_i>$ *y un rol* $r=<\beta_r, \Im_r, \chi_r>$, *decimos que i toma el rol r, denotado i/r sii* $\beta_r \subseteq \beta_i$ *y* $\Im_r \subseteq \Im_i$ *y* $\chi_r \subseteq \chi_i$. *Es importante mencionar que un agente solo puede tomar un rol a la vez, por lo tanto no existe la posibilidad de conflicto entre sus creencias.*

3.3 Ejemplo: exploración de un planeta

Para mostrar la aplicación del modelo de interacción basada en roles, se plantea un escenario en el cual el objetivo es explorar un planeta distante, concretamente, recolectar un tipo particular de rocas preciosas [40]. La localización de las muestras no se conoce de antemano, pero normalmente se encuentran distribuidas por grupos. Existe un grupo de vehículos autónomos que se pueden desplazar por el planeta recolectando muestras y trasladarlas a una nave espacial nodriza para enviarla de regreso a la tierra. No se cuenta con un mapa detallado del planeta explorado, pero se sabe que el terreno esta lleno de obstáculos – valles, colinas, etc.- que dificultan la comunicación entre los vehículos.

Se deberá construir una arquitectura de control para cada agente vehículo, que permita la cooperación para recolectar las muestras en forma eficiente.

Para simplificar el ejemplo se supondrá que el planeta esta dividido en cuadros de igual tamaño, que corresponderán a la unidad de movimiento de los vehículos. Además se asume lo siguiente:

- Un vehículo se puede mover hacia delante, hacia la izquierda o a la derecha
- Dos vehículos no podrán ocupar el mismo cuadro en el mismo tiempo.
- Algunas muestras ocupan el mismo cuadro.
- Los vehículos solo pueden trabajar con un cluster y transportar una muestra a la vez.
- Los vehículos están equipados con un censor que permite detectar obstáculos en los cuadros adyacentes (es decir, su vista se limita a 9 cuadros del escenario).
- Los vehículos pueden comunicarse con los que estén a dos cuadros de distancia: Si un vehículo detecta a otro, entonces le comunica sus creencias.
- Los vehículos se desplazarán un cuadro a la vez en una dirección específica.
- El cambio de dirección de un vehículo es de forma aleatoria.

A continuación se describen en detalle los roles, agentes y el escenario: R={ almacén, explorador, transportador}, Ag={{vehículo}, nodriza}.

La función de los roles es la siguiente: *rol almacén* recibe y almacena todas las muestras recolectadas, *rol explorador* explora el planeta buscando muestras y *rol transportador* transporta una muestra a la nave nodriza. Sus modelos se presentan en la **Tabla 3**.

Tabla 3. Modelos de roles de agentes

Rol	Creencias	Metas	Acciones
almacén (r_a)	B_r(muestraspresentes(X)) B_r(hayvehiculo(X,Y))	G_r(muestraspresentes(X))	recibirmuestras()
explorador (r_e)	B_r(haycluster(X,Y)) B_r(siguientepos(X,Y,D)) B_r(posicionactual(X,Y)) B_r(direccion(D)) B_r(hayobstaculo(X,Y))	G_r(haycluster(X,Y)) G_r(siguientepos(X,Y,D)∧¬ hayobstaculo(X,Y))	mover(X,Y,D) cambiardir(D) detectar(X,Y,D)
transportador (r_t)	B_r(posnodriza(X,Y) B_r(siguientepos(X,Y,D)) B_r(posicionactual(X,Y)) B_r(direccion(D)) B_r(hayobstaculo(X,Y)) B_r(traemuestra())	G_r(¬traemuestra()) G_r(siguientepos(X,Y,D)∧ ¬hayobstaculo(X,Y))	mover(X,Y,D) cambiardir(D) depositarmuestra()

El agente nodriza únicamente tomará el rol de almacén, por lo tanto *nodriza*=(B_i=B_{ra}∪∅, G_i=G_{ra}∪∅, A_i=A_{ra}∪∅), sin embargo, en un escenario más amplio, donde se contemple el retorno de la nodriza a la tierra, la nodriza podría tomar un rol que controle dicho retorno.

Una vez definidos los roles, por la definición propia de agente *vehículo* es, *vehículo* = (B_i=B_{rt}∪B_{re}, G_i=G_{rt}∪G_{re}, A_i=A_{rt}∪A_{re})

Como ejemplo, la siguiente formula modela el comportamiento del sistema cuando dos agentes pretenden entregar la muestra a la nodriza al mismo tiempo.

M,w|= [B_i(tiene(muestra) ∧ siguientepos(X, Y, D) ∧ posnodriza(X, Y)) ∧ B_j(tiene(muestra) ∧ siguientepos(X, Y, D) ∧ posnodriza(X, Y)]→ do_i[cambiadir(X,Y,D)]G(evitar(obstáculo))

Esta formula significa que el agente *i* cree que tiene una muestra para depositar en la nodriza y desplazándose en su dirección actual llegará a una posición (x, y), donde esta la nave nodriza; a la vez, el agente j cree que siguiendo su dirección actual se moverá a una posición (x,y), donde está la nodriza y que tiene una muestra para la nodriza, y tal que los valores x, y sean los mismos para los dos agentes, entonces se debe realizar la acción

cambiardir(x, y, z) en uno de los agentes(i en este caso), para alcanzar la meta de evitar el obstáculo.

3.4 Arquitecturas de agentes BDI

Los modelos BDI se formalizan generalmente en una lógica multi-modal cuantificada, conteniendo alguno o todas las siguientes características: modalidades para las actitudes de la información (conocimiento, creencias), modalidades para las actitudes pro-activas (deseos, metas, intenciones, compromisos, etc.) y representación de la acción. Además, prevemos que una lógica del agente sea capaz de representar los aspectos dinámicos (temporales) de la agencia. Una teoría completa del agente, expresada en una lógica con estas propiedades, debe definir cómo los atributos de la agencia son relacionados. Por ejemplo, necesitará mostrar cómo las actitudes de la información y pro-activas de un agente son relacionadas; cómo el estado cognoscitivo de un agente cambia en un cierto plazo; cómo el ambiente afecta el estado cognoscitivo de un agente; y cómo la información y las actitudes de un agente lo conducen para realizar acciones.

Para cambiar del foco teórico a la práctica tenemos que analizar como estos modelos pueden usarse para construir agentes. Hay dos posibilidades para hacerlo. Primero, las teorías del agente se podrían utilizar como formalismos de la representación del conocimiento, que sin embargo, suscita el problema de algoritmos para razonar con tales teorías. Suponga que uno desea razonar sobre nociones intencionales en un marco lógico. Claramente, las lógicas clásicas no son convenientes en su forma estándar para razonar sobre nociones intencionales: se requieren los formalismos alternativos. Revoque que la lógica de primer orden no es incluso *decidible*, pero las extensiones modales a ella (representaciones incluyendo las creencias, deseos, tiempo, y así sucesivamente) tienden ser *altamente* indecidibles. Así que, la idea de agentes como demostradores de teoremas parece, de momento por lo menos, ser irrealizable en la práctica.

Según el segundo enfoque, tales teorías representan las especificaciones para agentes y las lógicas del agente se consideran como lenguajes de especificación respectivamente. El problema dominante es ¿como ejecutar lenguajes de especificación del agente, con las modalidades para las creencia, deseos, y demás? Aunque fuera teóricamente posible, ¿podríamos hacerlo computacionalmente manejable?[2]

Las respuestas a estas preguntas se buscan en arquitecturas de agentes que determinan técnicas y algoritmos que soporten una metodología para construir agentes [22]. Particularmente, especifican como descomponer un agente en un

[2] Noten que los lenguajes como AGENT0 [39] no implementan las lógicas asociadas.

conjunto de componentes y como hacerlos interactuar con el fin de convertir el estado actual del agente y su percepción del ambiente en acciones.

Inicialmente, el objetivo del modelo BDI era modelar los mecanismos de generación de conductas racionales por un solo agente. Sin embargo, posteriormente a las arquitecturas BDI se agregaron los componentes de arquitecturas sociales y hasta elementos de arquitecturas reactivas como se muestra en la **Fig. 6**.

El proceso de decisión de un agente BDI empieza con el análisis de sus metas (tanto locales como cooperativas donde se requiere de intervención de otros agentes). Obviamente, este análisis se basa en el modelo del mundo que actualmente tiene el agente representado en la base de creencias. Durante el análisis se identifican las *situaciones* (de rutina, emergentes, locales y cooperativas) que corresponden al estado del mundo y del agente. El análisis de estos dos componentes (*creencias y metas*) le permite al agente generar una lista de opciones disponibles. Con la ponderación de las alternativas se termina el proceso de *deliberación* e inicia el proceso de *razonamiento medios y fines.*

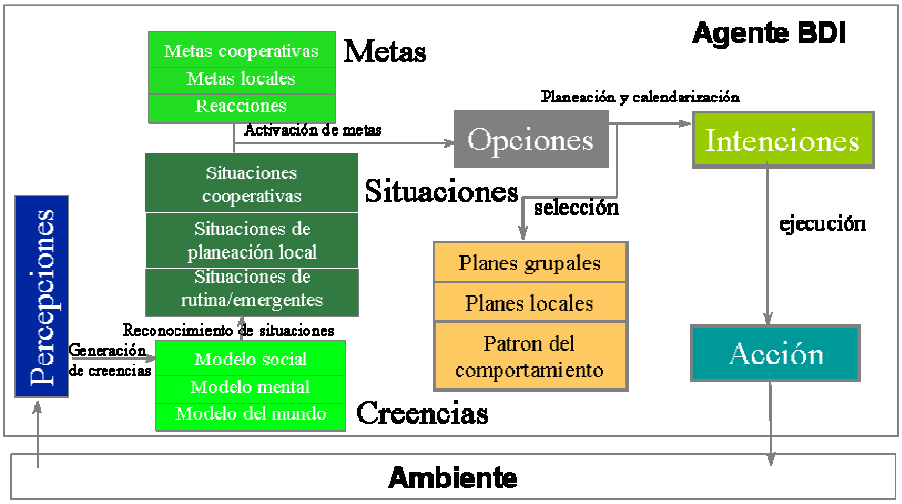


Fig. 6. Ejemplo de arquitectura BDI

En esta segunda fase, el agente debe seleccionar las mejores alternativas (desde el punto de vista de su utilidad para el agente) y comprometerse a algunas de ellas. Así se generan las *intenciones* del agente junto con los planes para alcanzarlas. De esta manera, las acciones de un agente están determinadas por sus decisiones, o por sus opciones. Las decisiones están lógicamente restringidas, aunque no determinadas, por las creencias del agente; estas creencias se refieren al estado del mundo (en el pasado, presente, y futuro), al estado mental de otros agentes, y a las capacidades de éste y otros agentes. Las decisiones son también restringidas por decisiones

anteriores. El componente de planes a nivel básico maneja ciertos *patrones de comportamiento* (*patterns of behaviour* - PoB). Éstos son estructuras que contienen las precondiciones (situaciones) que deben cumplirse para que un comportamiento (una acción) sea activado junto con las post-condiciones que definen los efectos de su ejecución.

Daremos mayor detalle sobre la implementación de arquitecturas BDI para programación de agentes en la última sección de este capítulo.

4. Interacciones sociales en SMA

En la sección anterior introdujimos algunos conceptos de comportamiento social de agentes (interacción, rol, escenario). En éste, analizaremos el concepto de interacción a mayor detalle. Cuando un agente interactúa con otros agentes, esta interacción constituye tanto la fuente de su poder como el origen de sus problemas. Las interacciones pueden ser positivas, permitiéndoles cooperar y realizar tareas que van más allá de sus capacidades individuales. Sin embargo, también pueden tener efectos negativos por la generación de conflictos entre los agentes. Estos conflictos pueden estar motivados por sus distintos puntos de vista sobre el problema a resolver, por sus intereses personales o sencillamente porque deben compartir recursos escasos. La posible existencia de conflictos, requiere de mecanismos para resolverlos o, mejor aún, evitarlos, lo que implica en general la realización de una serie de actividades extras. Tomando en cuenta esta visión, se podría decir que gran parte del trabajo realizado en SMA tiene como último fin, el diseño de mecanismos que permitan que las interacciones positivas sean maximizadas y las perjudiciales reducidas (o directamente evitadas).

4.1 Mecanismos de interacción

En cualquier sistema multiagente (o sociedad) suelen existir mecanismos generales que regulan las posibles formas en que estas interacciones se llevarán a cabo. A estos mecanismos los denominaremos *mecanismos de interacción*. Un mecanismo de interacción usualmente especificará:

- El conjunto de agentes Ag={1,...,n}, siendo i ∈ Ag un agente arbitrario que participa de alguna forma de decisión social: $i = \langle B_i, Ac_i, D_i \rangle$, donde $D_i : B_i \rightarrow Ac_i$ es la función de decisión que determina las acciones particulares que realizará el agente dado su conjunto de creencias B_i.
- El conjunto de posibles resultados sociales *Res*.
- El protocolo de interacción I_p.

– Un proceso de control, p_c, encargado de supervisar el cumplimiento de cualquier compromiso contraído por los agentes, en el resultado $r \in Res$ o durante el proceso de interacción que generó este resultado.

Sin lugar a dudas, la componente fundamental de un mecanismo de interacción es el protocolo de interacción, I_p, que especifica las posibles acciones que los agentes pueden tomar en las distintas etapas de la interacción.

Las creencias de un agente podrán incluir cualquier información disponible sobre el estado actual del sistema y las interacciones pasadas del agente, así como podrán representar a otros agentes que le permitirán razonar en forma estratégica sobre sus posibles comportamientos. Sin embargo, la componente más significativa de creencias en cuanto al comportamiento del agente estará representada usualmente por sus *preferencias* sobre los resultados sociales, P_r. Las preferencias P_r representarán la relevancia o importancia que tienen para el agente los posibles resultados en *Res*. Estas preferencias podrán estar resumidas en una relación de preferencia, que denotaremos ≥. Esta relación permitirá comparar pares de alternativas $x,y \in Res$, y es usual leer x ≥ y como 'x es al menos tan bueno como y', o 'x es débilmente preferido a y' o bien 'x es preferible o indiferente respecto de y'. La relación de preferencia constituye una relación de orden débil o simplemente un orden, que establece una escala ordinal entre las distintas alternativas en *Res*. Las preferencias también podrán estar representadas mediante una función de utilidad, $u : Res \rightarrow \Re$. Esta función de utilidad representará una relación de preferencia ≥ si, para todo $x,y \in Res$, se cumple que: $x \geq y \Leftrightarrow u(x) \geq u(y)$.

La elección de un mecanismo o protocolo de interacción particular, dependerá de los criterios empleados. Por ejemplo, bienestar social es la suma de los beneficios o utilidades[3] de todos los agentes en una solución o resultado dado. Este criterio, mide el beneficio global de los agentes. Cuando es medido en términos de utilidades, este criterio requiere que las utilidades de diferentes agentes se midan de la misma forma.

Al igual que el criterio anterior, el criterio de eficiencia Pareto toma una perspectiva global del resultado social. Un resultado (o solución) $r \in Res$ es Pareto eficiente (o Pareto óptimo) si no existe otro resultado $r' \in Res$ en el que al menos un agente obtiene un mayor beneficio en r' que en o y ningún otro agente obtiene un menor beneficio en r' que en r. En otras palabras, un resultado será Pareto óptimo, si no puede ser mejorado en favor de un agente sin perjudicar a alguno de los agentes restantes.

[3] En la teoría de juegos también se utiliza el término "pagos" (ver la Sección 4.4)

4.2 Agentes cooperativos y agentes auto-interesados

Es poco factible un análisis correcto de los mecanismos de interacción entre agentes si no se toma en cuenta el tipo de dominio en que será aplicado. No serán iguales las interacciones entre agentes que cooperan conjuntamente en la realización de una tarea u objetivo común (agentes cooperativos en tareas de la RDP), que en aquellos dominios donde el objetivo de un agente es derrotar o destruir al otro (agentes competitivos). También existirán casos intermedios donde si bien existe una idea de objetivo global o social, los agentes se comportarán en forma egoísta intentando maximizar su propio beneficio y sin importarles el bienestar social. De esta manera, los sistemas multi-agente se dividen en tres grandes grupos que intentan resaltar la preponderancia que tienen los objetivos individuales de los agentes en relación a los de los otros agentes y a los objetivos globales de la sociedad: *agentes cooperativos, antagónicos y auto-interesados*[4]. En cada uno de estos casos, los mecanismos de interacción serán completamente distintos, ya que los supuestos en que se basan difieren completamente.

Los agentes pueden necesitar de cooperación debido a que las tareas exceden sus capacidades físicas o conocimiento individuales. En otros casos, aún cuando estos agentes podrían realizar sus tareas en forma separada, un trabajo conjunto cooperativo podría maximizar el aprovechamiento de las interacciones positivas potenciales. Este es el caso por ejemplo, de aquellos agentes que tienen objetivos personales, pero su realización implica actividades que también deben ser realizadas por otros agentes. La identificación y redistribución de estas actividades en común podría mejorar los beneficios personales de cada uno de los agentes.

En el caso de RDP, los objetivos de los agentes por lo general pueden ser planteados como conjuntos de tareas que cada uno debe realizar, y el proceso de interacción tiene como objetivo encontrar una redistribución de las tareas que permita aprovechar las posibilidades de cooperación entre los agentes. El rol del mecanismo de interacción en este contexto es coordinar las decisiones locales e intercambiar la información local para promover el objetivo global del sistema. Por otro lado, para los sistemas donde las acciones de un agente afectan directamente a los demás (por ejemplo en situaciones de conflicto por el acceso a recursos compartidos), los objetivos de los agentes pueden ser planteados como conjuntos de estados aceptables para cada uno de ellos, y el proceso de negociación tiene como objetivo encontrar un plan conjunto coordinado que aproveche las posibilidades de cooperación y evite o resuelva las situaciones de conflicto.

Finalmente, en las situaciones donde los agentes pueden asignar un valor a cada estado potencial que refleja su grado de importancia o deseabilidad, los

[4] Cabe destacar que el término sistema multi-agente ha sido utilizado mayormente como un sinónimo de sistemas con agentes auto-interesados o individualistas.

agentes podrán relajar sus objetivos iniciales, llegándose a situaciones que satisfacen sólo parcialmente sus objetivos. El objetivo del proceso de negociación será determinar que partes de sus objetivos serán satisfechas, y en forma paralela cómo será implementado el plan conjunto que implementa estas partes.

En este caso de agentes auto-interesados, los agentes son provistos con un protocolo de interacción donde cada agente podrá elegir su propia estrategia (ver la sección 4.4). Por este motivo, los mecanismos de interacción deberán ser diseñados usando una perspectiva estratégica: el análisis estará centrado ahora en determinar los resultados sociales que se obtendrán dado un protocolo que garantiza que un comportamiento deseado del agente es su mejor estrategia y por lo tanto el agente la usará. Este tipo de enfoque es requerido para diseñar sistemas multiagente robustos y abiertos donde los agentes pueden ser construidos por distintos diseñadores y/o representar intereses de entidades diferentes en el mundo real.

Vemos por lo tanto que la cooperación puede ser beneficiosa para aprovechar las interacciones positivas y reducir el efecto de las interacciones negativas (conflictos), tanto en los sistemas multiagente cooperativos como en los auto-interesados. Desde el punto de vista de la cooperación, uno de los principales mecanismos que han sido utilizados históricamente a este fin, es el referenciado bajo el nombre general de coordinación de acciones.

Definición 5. *La coordinación de acciones es un proceso generado por la necesidad de tomar en cuenta las acciones de otros agentes y que tiene como objetivo principal una mejora en la acción conjunta del grupo mediante la articulación de las acciones individuales de los agentes.*

Una forma frecuente de coordinación de acciones que ocurre entre agentes con diferentes objetivos es denominado negociación. Por otra parte, los agentes cooperativos persiguiendo un objetivo en común, también pueden tener distintos tipos de conflictos. La distribución del control y los datos entre distintos agentes, puede llevar a que éstos tengan una visión parcial y distinta de la realidad y tengan diferentes perspectivas sobre la mejor manera de resolver el problema compartido. Un caso de conflicto típico también surge cuando estos agentes deben acceder a recursos compartidos insuficientes. Es cierto que este tipo de conflictos tiene características distintas de los originados por la existencia de intereses personales. Sin embargo, en cualquiera de los dos casos, será necesario que los agentes negocien para evitar o resolver estos conflictos.

4.3 Mecanismos de negociación

El término negociación ha sido usado con diferentes significados en la literatura relacionada a los SMA. En [34] por ejemplo, se utiliza el término

protocolo de negociación, para referenciar a distintos mecanismos de interacción que son aplicables al contexto de sistemas multiagente con agentes auto-interesados. Estos mecanismos incluyen las subastas, sistemas de votación, mecanismos de mercado, etc. [20]. En otros casos, la palabra negociación ha sido usada como una metáfora para referenciar a distintos procesos de coordinación y resolución de conflictos en sistemas multiagente. Estos procesos, podían consistir en la resolución de disparidad de objetivos en planeación, resolución de restricciones por el acceso a recursos, comunicación de cambios de plan, etc.

Definición 6. *La negociación entre agentes estará caracterizada por la existencia de alguna forma de conflicto que debe ser resuelto de manera descentralizada por agentes con intereses personales en forma de un trato, generado mediante un intercambio de propuestas y contrapropuestas en base a algún protocolo de negociación.*

En un contexto de negociación, el principal problema que se debe enfrentar es el de determinar cual es el trato específico al cual deberían llegar los agentes. En general, existe consenso sobre cual sería el conjunto de posibles tratos que sería razonable que se consideraran en el proceso de negociación. Este conjunto es denominado el conjunto *bargaining* o conjunto de negociación [31]. El conjunto *bargaining* es el conjunto de todos los tratos que son racionales individualmente y Pareto óptimos (ver también el concepto de *core* de juego en la Sección 4.4).

Como es sabido, las soluciones que se basan en el concepto de eficiencia Pareto constituyen en realidad una frontera de posibles soluciones. La elección de una solución particular de una frontera Pareto, requiere en general de algún criterio extra que defina la elección a favor de alguna de estas soluciones. En el contexto de *bargaining*, John Nash argumentó que se pueden decir más cosas sobre los posibles tratos que resultarán de la negociación de dos agentes racionales, más allá del hecho de que pertenecerán a un conjunto *bargaining*. En su argumentación, Nash postuló una serie de axiomas que un trato debería satisfacer para un problema bargaining abstracto y demostró además que existe sólo un trato que satisface estos axiomas. Esta solución es usualmente referenciada como la solución *bargaining* de Nash que ha sido utilizado como referencia para evaluar los tratos alcanzados mediante distintos mecanismos computacionales de negociación.

4.3.1 Mecanismos de subastas

En muchas aplicaciones prácticas que implican algún tipo de negociación (como asignación de tareas y recursos), han sido utilizados los mecanismos de subastas representando escenarios de interacción de fácil automatización[5]. Un mecanismo de subastas clásico está compuesto por un conjunto de agentes Ag

[5] Subastas también se pueden considerar como mecanismo de asignación de recursos.

donde uno de los agentes cumple el rol de subastador (o rematador) y los agentes restantes son los licitadores (o postores). En las subastas el resultado social es un trato entre sólo dos agentes: el subastador y uno de los oferentes quienes buscan maximizar su propio beneficio.

El escenario de subastas clásico asume que el subastador desea vender un ítem al precio más alto posible, mientras que los oferentes desean comprarlo al precio más bajo posible. Los mecanismos de subastas han utilizado distintos protocolos de subastas que varían de acuerdo a varias dimensiones a saber.

Subasta Inglesa: subastas con el precio ascendente *first-price open cry*. El precio comienza bajo (usualmente un precio de reservación) y las ofertas se realizan en cantidades incrementales.

Subasta Danesa: subastas con el precio descendente. El subastador va reduciendo paulatinamente el precio del ítem. El primero que ofrece lo que se pide gana.

Licitación a sobre cerrado: Subastas *one shot first-price sealed bid*. Existe una sola ronda de ofertas, después de la cual el subastador asigna el bien al ganador. Los agentes no son capaces de determinar las ofertas realizadas por los otros agentes.

Subasta de Vickrey[6]: subastas *one shot second-price sealed-bid*. En este caso, al igual que en la licitación por sobre cerrado, los oferentes proveen sus ofertas sin conocer las ofertas de los otros oferentes, ganando aquel oferente que realizó la propuesta más alta. Sin embargo, el precio que efectivamente debe pagar el ganador es el precio de la segunda oferta más alta. La propiedad interesante de este protocolo es que en determinado tipos de subastas, si se usa el protocolo de Vickrey la estrategia dominante de cualquier oferente consiste en ofrecer su verdadera valoración del ítem a comprar, convirtiéndola en buena candidata para el caso de oferentes no sinceros (estratégicos). Otra de las características importantes de estas subastas, es que garantice un resultado Pareto-óptimo: un acuerdo x (el ganador que obtiene el artículo por el precio especifico del subastador) tal que no haya otro acuerdo y que este sería mejor para al menos uno de los agentes sin ser el peor para cualesquiera de los demás: pagando un precio mas alto podría ser peor para el ganador, pagar un precio mas bajo sería peor para el subastador y dar el artículo a un comprador diferente sería peor para el ganador (quién habría pagado el precio menor o al menos igual a su valuación privada, dado el acuerdo x).

4.3.2 Mecanismos basados en la teoría de juegos

Otro tipo de mecanismos de negociación, que se ha desarrollado en el contexto de agencias desde los años 1980, utiliza los modelos de la teoría de juegos debida a J. Von Neumann y O. Morgenstern [30]. Cada agente es un jugador el cual tiene un número finito (infinito) de posibles estrategias de

[6] Propuesta por William Vickrey en 1961 (Premio Nobel en Economía en 1996).

juego, y deberá jugar ó tomar decisiones de selección de estrategias que le maximice su utilidad.

John F. Nash analiza juegos cooperativos, en los cuales se pueden llegar a acuerdos obligatorios, y juegos no cooperativos, donde dichos acuerdos no son factibles, desarrollando un concepto de equilibrio para predecir el resultado de dichos juegos (equilibrio de Nash), analizando la interacción de estrategias dinámicas entre los diferentes agentes, incluso dando los fundamentos teóricos para predecir los resultados de dichas interacciones entre agentes con información parcial, acerca de los objetivos de los otros individuos. De esta manera, los juegos se clasifican en:

- Juegos Matriciales. Dos ó más jugadores toman decisiones en situación de conflicto ó competitiva. Cada jugador tiene como objetivo alcanzar un resultado que le sea ventajoso, tanto como sea posible. Aquí se considera una matriz de pagos, donde las estrategias para cada jugador tienen asociados pagos. El juego puede considerar un número finito o infinito (juegos continuos) de estrategias. Cuando la suma de los pagos de los jugadores es cero, esto es, si el jugador 1 recibe $P(x, y)$ y el jugador 2 recibe $P(x, y)$, asociados a la selección de las estrategias x, y, por los jugadores 1 y 2 respectivamente, con $x, y \in [0,1]$, las estrategias son puras. Sin embargo, existen las estrategias mixtas, donde los jugadores seleccionan éstas atendiendo a funciones de probabilidad y entonces sus pagos son esperanzas matemáticas asociadas. Los pagos son números reales, aunque se pueden manejar pagos difusos, esto es, intervalos de pagos con ciertas funciones de membresía.
- Juegos no cooperativos entre n agentes. Aquí la cooperación entre agentes se olvida, no hay comunicaciones de pre-juegos, esto es, no hay acuerdos obligatorios, no existe correlación de estrategias, ni se permite la transferencia de pagos (juegos TU). Cada jugador busca maximizar su pago individual y busca la estrategia que le resulte más ventajosa, sin embargo, puede ocurrir que lo que es ventajoso para un jugador puede ser desventajoso para otros jugadores.
- Juegos cooperativos entre n agentes. Se dan en ambientes donde se forman grupos ó coaliciones entre jugadores, en un plano benevolente.

En la siguiente sección consideraremos más a detalle los juegos cooperativos en el contexto de formación de coaliciones entre agentes.

4.4 Formación de coaliciones de agentes

En los SMA´s, un agente auto-interesado seleccionará la mejor estrategia por si mismo, la cual no puede explícitamente ser impuesta desde fuera. La pregunta principal es qué resultados sociales siguen dando un protocolo que

garantice que las estrategias seleccionadas sean las de utilidad global máxima ó de mayor consenso, y entonces sea la que use. Esta aproximación normativa se requiere en el diseño de SMA´s robustos donde los agentes se puedan construir por diferentes diseñadores y/ó puedan representar diferentes partes del mundo real.

En los ambientes cooperativos, pudiera ser más benéfico formar grupos, de tal suerte que el desempeño grupal fuera más eficiente que el individual, porque las capacidades individuales se ven incrementadas ó se expanden al actuar en grupo, y además porque el desempeño grupal es más que la suma de los desempeños individuales, debido a las interacciones entre los agentes que forman el grupo. La creación dinámica de grupos de agentes como resultado de negociación entre ellos se conoce como formación de coaliciones.

Definición 7. *Una coalición es una sociedad de agentes interesados que en base a protocolos de negociación deciden cooperar para desarrollar una tarea ó alcanzar una meta, un objetivo y hasta un ideal en el límite* [14].

Las soluciones a problemas de formación de coaliciones desarrollados dentro de la IAD clásica, se concentran por lo general sobre el caso especial de agentes auto-interesados en un ambiente super-aditivo, cuando los pagos a los agentes no son tan importantes. En ambientes super-aditivos la gran coalición (coalición que incluye a todos los agentes) es la más ventajosa, por tanto la formación de coaliciones en ambientes super-aditivos usando como metodología la RDP se reduce a constituir la gran coalición. A diferencia con este enfoque, en SMA´s de agentes auto-interesados, se buscan soluciones en ambientes no necesariamente super-aditivos.

Los protocolos de formación de coaliciones se han evolucionado desde una perspectiva centralizada (característica de sistemas de la RDP) hacia mecanismos descentralizados. Los modelos de formación de coaliciones se clasifican en: modelos basados en utilidades y modelos basados en complementariedad. Los primeros utilizan métodos clásicos y protocolos para formar coaliciones estables, en equilibrio y que giran sobre el principio "*bellum omnium contra omnes*" (en latín, *la guerra de todos contra todos*), y corresponden a la Teoría de Juegos e Investigación de Operaciones, mientras que los segundos giran sobre el uso colaborativo de las herramientas de complementariedad individual para agrandar la potencia de cada agente para lograr sus metas.

4.4.1 Formación de coaliciones basada en juegos cooperativos

En juegos cooperativos con coaliciones, la Teoría de Juegos usualmente se enfoca en buscar estabilidad (o equidad) y el cálculo de los pagos correspondientes a la estructura coalicional, dada una configuración previamente formada de coaliciones (esto es, una partición de agentes en subconjuntos, llamada estructura de coaliciones). A diferencia con este enfoque

estático, en SMA con agentes auto-interesados la generación dinámica de la estructura coalicional representa un problema por si mismo.

Recientemente ha sido estudiado la teoría de juegos con coaliciones difusas, introducida con el uso del enfoque de *core* (un conjunto de configuraciones de pagos, donde cada vector de pagos es tal que los agentes de una coalición no están motivados a formar subgrupos y a que se separen de la estructura coalicional). Lamentablemente, en los juegos *crisp* en muchas ocasiones el *core* es vacío, el problema que se puede resolver con el enfoque del *core* difuso introducido por J. P. Aubin que considera una familia de coaliciones por un lado y un conjunto de multi-estrategias asociadas a ésta, con funciones de utilidad de los agentes pertenecientes a la familia. El *core* del juego lo constituyen las multi-estrategias que no son rechazadas por alguna coalición y a medida que el *core* admite más coaliciones a formarse y rechaza estrategias, generando así un conjunto de coaliciones difusas, convirtiendo al juego en uno difuso, pudiendo existir un equilibrio débil en el juego. En los modelos clásicos, la borrosidad es debida al hecho de que los jugadores pueden tener diferentes grados de participación en diferentes coaliciones.

Este enfoque ha sido extendido por M. Mareš [23], quien construye el *core* de imputaciones (pagos asociados para llegar a un acuerdo de formar coalición entre agentes) y da las condiciones para la existencia de solución del juego, así como criterios para formar estructuras de coaliciones efectivas ó estables. Este modelo agrega la vaguedad de los pagos (son cantidades difusas, esto es, un intervalo cuyos valores tienen asociada una función de membresía) esperados por jugadores individuales que implica a su vez la vaguedad en la formación de las coaliciones ventajosas en pagos, y consecuentemente la vaguedad de las estructuras de coaliciones resultantes.

Posteriormente en el trabajo [37] se propone un modelo generalizado con el *core* difuso, expandiendo los supuestos del *core* convencional e involucrando elementos adicionales como variables y restricciones. Esto permite eliminar las deficiencias del enfoque tradicional donde (i) los argumentos del *core* no involucran el criterio para generar coaliciones efectivas; (ii) el *core* se limita a estructuras de coaliciones mutuamente exclusivas y (iii) no se permite incorporar características como secuenciación en tareas. Como método de solución del juego se utiliza un algoritmo genético.

Es importante constar, que los modelos de la teoría de juegos también contemplan la negociación entre agentes, solo que esta negociación se da implícitamente con base en las utilidades difusas y los intereses difusos de los agentes reflejados en sus pagos. Obviamente, el algoritmo que implemente este modelo requiere de un módulo central para construir el *core* del juego recibiendo las expectativas de los pagos de agentes individuales, de las coaliciones y de la gran coalición, así como para obtener la solución. Sin embargo, esta "centralización" se debe exclusivamente al método de solución, conservando la naturaleza distribuida del SMA. El protocolo de negociación

asociado al modelo extiende el Protocolo de Red de Contratos (CNP) para construir el *core* de juego y generar una estructura de coaliciones efectivas.

Otros enfoques a la formación de coaliciones buscan explorar la comunicación par-a-par entre agentes, a continuación consideraremos uno de ellos.

4.4.2 Formación de coaliciones descentralizada

Formación de coaliciones puede ser vista como un método de asignación (distribución) de tareas entre agentes auto-interesados: de tal forma que si tenemos un grupo de agentes $Ag=\{1,...,n\}$ y un conjunto de tareas independientes $T=\{T_1,T_2,...,T_n\}$ las cuales pueden ser satisfechas por dichos agentes, podemos considerar situaciones donde cada tarea puede ser resuelta por un grupo de agentes. Cada agente tiene un conjunto de capacidades $B_i=\{b_1,b_2,..,b_n\}$, propias que le permiten contribuir para la realización de la tarea y es en base a esas capacidades es tomado en cuenta dentro de alguna coalición. Un agente solo tendrá conocimiento de las capacidades de otros agentes, si estos forman parte de las coaliciones que va a calcular, en otro caso solo tendrá conocimiento de su existencia en el sistema. El uso de las capacidades del agente implica un costo, lo cual permite determinar el costo de llevar a cabo la coalición así como sus beneficios.

Cada una de las tareas es representada de manera independiente por un vector de capacidades $Bl=\{bl_1,bl_2,..,bl_n\}$, las cuales deben de ser cumplidas para que dicha tarea pueda ser ejecutada. Por su parte, cada coalición calculada tiene un vector de capacidades $Bc = \{bc_1,bc_2,...,bc_n\}$ en el que cada elemento representa la suma de las capacidades de los agentes que contribuyen a dicha coalición, dicho vector nos permite determinar si una coalición en especifico puede ejecutar una tarea, para que una tarea pueda ser ejecutada por una coalición cada una de las capacidades de Bc debe ser igual o mayor a cada una de las capacidades del vector de tarea Bl (esto es, $0 \leq i \leq n$, $bl_i \leq bc_i$). En este algoritmo no se permite el traslape entre coaliciones: para cada par de coaliciones $C1$ y $C2$ en el sistema, $C1 \cap C2=0$, es decir, las coaliciones son disjuntas.

Para poder determinar el valor de la ganancia de llevar a cabo una coalición, también se considera la existencia de un conjunto de ganancias $Gt =\{Gt_1,Gt_2,..,Gtn\}$, en la que cada elemento es representada por un vector de ganancias $Gt_1 =\{Gc_1,Gc_2...,Gc_n\}$, en donde los elementos indican la ganancia adquirida por llevar a cabo cada capacidad de la tarea. Cada coalición tiene asociado un valor que representa la ganancia que se espera y de que se ejecute la tarea por la coalición.

El algoritmo de formación de coaliciones se basa en [36] y consta de dos fases principales. En la primera, todas las posibles coaliciones son distribuidamente calculadas. La segunda, los valores de ganancia de las coaliciones son calculadas por cada agente, lo cual permitirá elegir a la

coalición que mejor cumpla con la tarea, y, posteriormente, los agentes deciden que coalición es la mejor y la forman (Fig. 7).

El estado inicial del sistema consiste de n agentes individuales. Los agentes comenzarán a negociar y paso a paso formarán coaliciones, los agentes que formen una coalición terminaran su proceso de cálculo de formación de coaliciones, y solo los agentes restantes continuaran negociando.

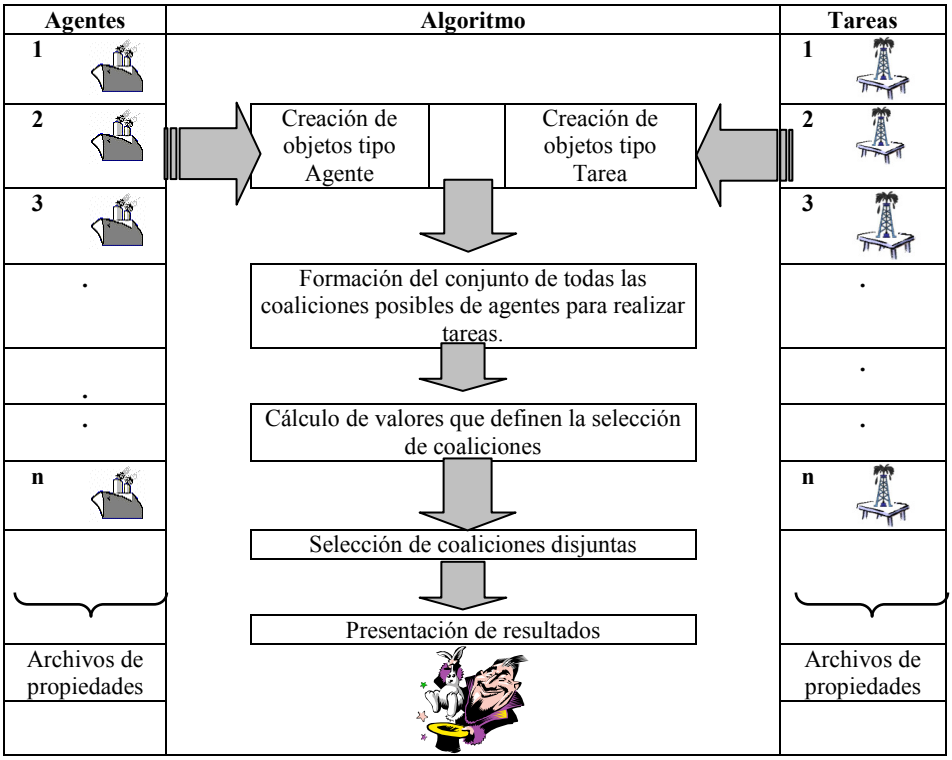


Fig. 7. Algoritmo de formación de coaliciones

Fase 1. Acuerdos preeliminares de distribución y cálculo.

En esta etapa los agentes buscan formar acuerdos entre ellos para el cálculo distribuido del valor de las coaliciones. Para llevar a cabo este paso los agentes realizan los siguientes pasos:

1. Calculan todas las permutaciones posibles con los n agentes existentes, y el resultado de esto lo almacenan en P_i, el cual es el conjunto de coaliciones potenciales del agente i.

2. Mientras P_i no sea vacío, hacer:

- Contactar a algún agente j que forma parte de alguna coalición en P_i.
- Si este es el primer contacto de j solicitarle sus capacidades
- Una vez que se hayan contactado, ponerse de acuerdo, que agente calculará las coaliciones comunes, es decir, comprometerse al cálculo de los valores del subconjunto S_{ij} el cual es formado por las coaliciones comunes entre los agentes i y j.
- Eliminar del conjunto de coaliciones potenciales a las coaliciones del subconjunto S_{ij} y agregar S_{ij} a la lista de coaliciones comprometidas del agente i que se representa por L_i.
- Para cada agente k que se haya contactado con el agente i, eliminar de la lista de coaliciones potenciales P_i el subconjunto de coaliciones comunes S_{ik} que el agente k se ha comprometido a calcular.
- Calcular los valores de las coaliciones con las que se comprometió calcular, en el caso de capacidades desconocidas contactar a los agentes necesarios.
- Seguir contactando a otros agentes hasta que el conjunto de coaliciones potenciales $P_i = \{i\}$, lo que significa que no hay agentes con quien contactar.

Fase 2. Calculo distribuido y repetido de los valores coalicionales y selección de las coaliciones.

1. Calculo distribuido y repetido de los valores de las coaliciones.

Este algoritmo de formación de coaliciones requiere la repetición de los cálculos de las coaliciones. Para llevar a cabo este procedimiento, repetir por cada coalición C que esté en la lista de coaliciones comprometidas (L_i) lo siguiente:

- Calcular el vector de capacidades potenciales (B_c^{pc}) de la coalición, esto es, sumar todas las capacidades de los agentes que participan en la coalición.
- Formar una lista Ec de las salidas esperadas de las subtareas en T cuando la coalición la lleve a cabo. Para cada subtarea Bl, hacer:
  - Revisar que capacidades B_l^j son necesarias para la satisfacción de la tarea T_j.
  - Comparar las capacidades necesarias B_l^j con las capacidades B_c^{pc} de la coalición, con el fin de de saber si la tarea puede ser satisfecha por la coalición.
- De la lista de salidas esperadas elegir a aquella que tenga el máximo valor coalicional (Vc), y calcular el costo de llevar a cabo la coalición el cual es $(Cc=1/Vc)$.

En el caso de que el agente i termine rápidamente de calcular sus coaliciones comprometidas y otro agente j no ha terminado sus cálculos, el agentes i puede contactar al agente j para comprometerse a realizar el calculo de los valores de las coaliciones S_{ik} comunes entre ellos, en las que dichas

coaliciones son parte de Lj, de esta forma el agente i puede agregar a S_{ik} a su lista de coaliciones comprometidas L_i y realizar las operaciones que ya se describieron con anterioridad.

2. Selección de las coaliciones.

En esta fase los agentes paso a paso determinan que coaliciones son las más factibles a ser formadas, esto es en base a las ganancias y costos que cada una de dichas coaliciones implican. Se establece una proporción entre el costo de la coalición y su tamaño por $w_i = c_i / |C_i|$ el cual es llamado peso coalicional. Al final de la primera sub fase de cada iteración del algoritmo, cada agente habrá calculado una lista de coaliciones, valores coalicionales y sus pesos.

Cada agente debe de realizar de forma iterativa lo siguiente:

- Encontrar en L_i la coalición Cj con el menor peso w_j.
- Anunciar a los demás agentes el peso wj que ha sido hallado.
- Elegir el menor peso entre todos los pesos de las coaliciones anunciadas. Este w_{low} será elegido por todos los agentes, y también seleccionar la coalición C_{low} y tarea T_{low} correspondiente.
- Borrar a los miembros de la coalición elegida C_{low} de la lista de agentes candidatos para nuevas coaliciones.
- Si el agente es parte de la coalición elegida Cl_{ow}, unirse a los demás miembros y formar la coalición.
- Borrar de L_i todas las posibles coaliciones que incluyan a los agentes borrados
- Borrar de T la tarea T_{low} elegida.
- Asignarle a L_i de las coaliciones restantes las coaliciones en L_i, cuyos valores pueden ser recalculadas.

El algoritmo anterior puede ser repetido hasta que los agentes son borrados (forman parte de alguna coalición elegida), o no existen mas tareas que satisfacer o ninguna de las posibles coaliciones es beneficiosa de llevar a cabo.

En los algoritmos descentralizados cuando el tamaño poblacional de agentes es grande, se genera complejidad en el sistema, por la cantidad de interacciones, haciendo que los protocolos sean NP-completos, en cuanto a comunicaciones y cómputo, y por tanto ineficientes para resolver problemas distribuidos en SMA´s. Para reducir la complejidad a una polinomial ó menor se involucran los artificios (por ejemplo, se restringe el tamaño de las coaliciones).

5. Comunicación entre agentes

Interacciones sociales como la cooperación, coordinación y negociación se realizan a través de un lenguaje de comunicación y los protocolos de interacción asociados a este lenguaje. Este lenguaje de comunicación de

agentes (ACL por sus siglas en Ingles: *Agent Communication Language*) inhabilita la comunicación a nivel de conocimiento, mientras que el servicio de transporte de mensajes complementa la infraestructura de comunicación para la entrega de mensajes a nivel físico.

Las diferencias principales de comunicación basada en ACL de otros modelos de comunicación conocidos en la computación (p.ej. *Remote Procedure Calls, Remote Method Invocation,* etc.) radican en la complejidad semántica de mensajes y su contenido que permite i) representar proposiciones, reglas y acciones en lugar de objetos simples sin alguna semántica asociada y ii) describir los estados de agentes en un lenguaje declarativo en lugar de una función o método a invocar. Lo podemos ilustrar con el siguiente ejemplo. En una aplicación distribuida de dos objetos el Objeto O_2 invoca el método *m* del Objeto O_1 (en Java: *o1.m(arg)*). En este escenario, O_2 mantiene el control sobre la invocación del método y O_1 esta obligado ejecutar *m*. En un escenario similar, Agente A_2 le solicita al Agente A_1 ejecutar una acción α (A_2 envía un mensaje de solicitud: *request*). A_1 mantiene el control sobre la decisión de atender o no la solicitud ejecutando la acción α y así conserva su autonomía.

5.1 Lenguaje de comunicación de agentes (ACL)

Desde los primeros trabajos sobre agentes, se enfatizaba la importancia para agentes de la comprensión de un idioma de comunicación: "una entidad es un agente si y sólo si comunica correctamente en un Lenguaje de Comunicación de Agentes (ACL)" [15]. La importancia de tener un ACL se reconoce comúnmente - imaginen una sociedad sin un lengua común. Las ACLs derivan su inspiración de la teoría de los actos de habla que fue desarrollado por los filósofos y lingüistas en un esfuerzo por entender cómo los humanos usan un idioma en cada situación del día. Su desarrollo se asocia con John Austin quien fue el primero en notar que algunos enunciados (*utterances)* igual que acciones físicas buscan cambiar el estado del mundo, o al menos, representar nuestra intención de satisfacer una meta o intención. El distingue tres diferentes aspectos de los actos de habla: *locución* – la enunciación misma, *ilocución* – la acción (intención) codificada en el mensaje (solicitud, orden, etc.) y *perlocución* – efecto esperado del acto de habla.

La especificación de FIPA ACL consiste en un conjunto de tipos de mensajes y la descripción de sus pragmáticas que son los efectos en las actitudes mentales del agente: tanto del remitente como del receptor. Cada acto comunicativo tiene una semántica formal basada en la lógica modal. La **Fig. 8** resume los elementos estructurales principales de un mensaje ACL. El primer elemento del mensaje es una palabra que identifica el acto

comunicativo[7], que define el significado principal del mensaje. Sigue entonces una secuencia de parámetros, compuesta por palabras claves del parámetro el cual empieza con un ":". Uno de los parámetros define el contenido del mensaje (*content*), codificado como una expresión con cierto formalismo (*language sl*). Otros parámetros ayudan al servicio de entregar el mensaje correctamente (remitente y receptor), otros ayudan a que el receptor interprete el significado del mensaje (tanto su lenguaje de codificación de contenido como ontología que se usará para interpretar los conceptos de contenido), o ayudan a mantener la conversación (por ejemplo *reply-with, in-reply-to, protocol*).

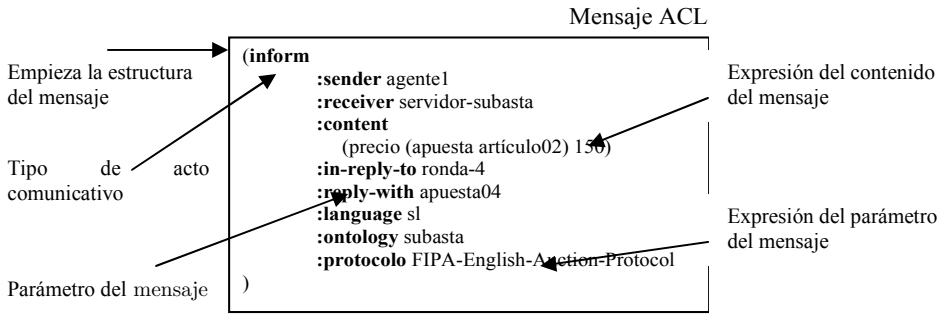


Fig. 8. Estructura de mensaje FIPA-ACL

El modelo conceptual de comunicación con el lenguaje FIPA-ACL ilustrando lo anterior se representa en la **Fig. 9**. El acto comunicativo es considerado como una sentencia, donde el contenido es el objeto gramatical de la sentencia. En general, el contenido puede codificarse en cualquier lenguaje, y ese lenguaje será denotado por el parámetro "*:language*". El único requisito en el lenguaje de contenido es el que tenga la expresividad necesaria para poder expresar proposiciones, objetos y acciones. Una proposición da lugar a que alguna frase en un lenguaje sea verdadera o falsa. Un objeto, en este contexto, es una estructura que representa una "cosa" identificable (qué puede ser abstracta o concreta) en el dominio del discurso. Finalmente, una acción es una estructura que el agente interpretará como ser una actividad que puede ser llevada a cabo por algún agente. Para identificar los objetos del universo de discurso, se puede asociar una ontología al mensaje.

[7] En la Teoría de los Actos de Habla las acciones comunicativas se llaman "performatives". Este término se conserva también en otro lenguaje de comunicación de agentes – KQML *(Knowledge Query and Manipulation Language*) [12].

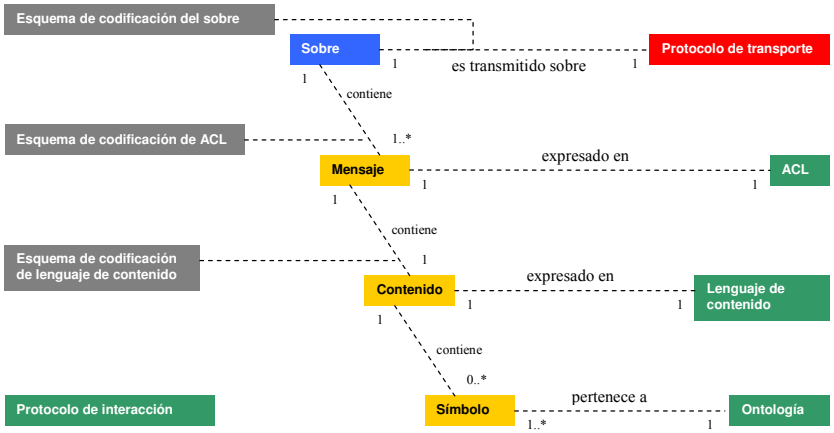


Fig. 9. Componentes del modelo de comunicación de FIPA

5.1.1 Semántica del lenguaje FIPA-ACL

Para definir la semántica del lenguaje FIPA-ACL se utiliza la conceptualización del modelo BDI de agentes considerado anteriormente. Supongamos que, en términos abstractos, el agente i tiene entre sus actitudes mentales lo siguiente: alguna meta u objetivo G (**Fig. 10**). Al decidir satisfacer G, el agente adopta una intención específica, I. Note que ninguna de estas declaraciones trae consigo un compromiso en el diseño del agente: G e I, ambos podrían ser equivalentemente codificados como términos explícitos en las estructuras mentales de un agente BDI, o implícitamente en un lenguaje de programación como Java o C# así como en un lenguaje de consultas a una base de datos.

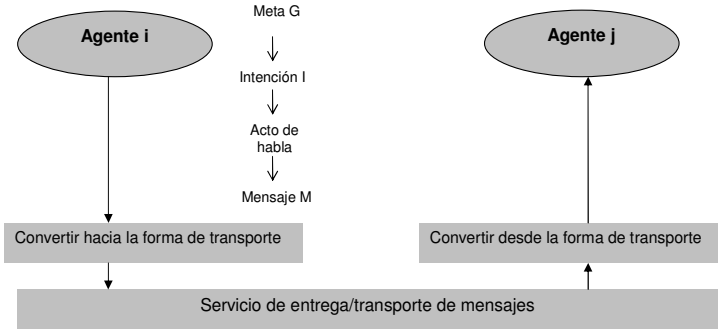


Fig. 10. Conceptualización del paso de mensaje entre dos agentes

Asumiendo que i no puede llevar a cabo la intención por sí mismo, la pregunta se vuelve entonces ¿qué mensaje o conjunto de mensajes deben enviarse a otro agente (j en la **Fig. 10**) para ayudar que la intención I sea satisfecha? Si el agente i está comportándose en algún sentido razonable, este seleccionará un acto comunicativo (enviará un mensaje) cuyo efecto podrá satisfacer la intención y de esta manera logrará la meta. En resumen: un agente planea explícitamente o implícitamente (a través de la construcción de su software) lograr sus metas finalmente, comunicándose con otros agentes, es decir, enviando mensajes a ellos y recibiendo mensajes de ellos. El agente seleccionará las acciones basadas en la relevancia de la acción esperada o su efecto racional a sus metas. Sin embargo, no puede asumir que el efecto racional de los mensajes enviados necesariamente se logrará.

SL, es la abreviación para Lenguaje Semántico (*Semantic Language*), el lenguaje formal usado para definir las semánticas de FIPA ACL. En SL, las proposiciones lógicas son expresadas en una lógica de actitudes mentales y acciones, formalizadas en un lenguaje modal de primer orden con identidad [33]. Los componentes del formalismo usados en lo siguiente son definidos como sigue:

p, p_1,... - fórmulas cerradas que denotan proposiciones,

ϕ y ψ - son esquemas de fórmula, que simbolizan cualquier proposición cerrada

i y j - son variables esquemáticas que denotan a agentes

$/=\phi$ - significa que ϕ es válido.

El modelo mental de un agente es basado en la representación de tres actitudes primitivas: creencia, incertidumbre y elección (o de algún modo, meta). Son formalizados respectivamente por los operadores modales *B, U, y C*. Las fórmulas que usan estos operadores pueden leerse como:

$B_i p$ – el agente i (implícitamente) cree (que) p,

$U_i p$ - el agente i esta incierto sobre p pero piensa que p es más probable que $\neg p$,

$C_i p$ - el agente i desea que p actualmente se sostenga.

El modelo lógico para el operador B es una estructura semántica de mundos posibles de Kripke KD45 con el principio del dominio fijo.

Para habilitar el razonamiento sobre acciones, el universo del discurso involucra, además de los objetos individuales y agentes, secuencias de eventos. Una secuencia puede formarse con un solo evento. Las expresiones de acción serán denotadas por *a*. Para hablar sobre los planes complejos, eventos (o acciones) pueden ser combinados para formar expresiones de acción:

a_1 ; a_2 - es una secuencia en la que a_2 sigue a a_1

a_1 | a_2 - es una elección no determinística en que suceden a_1 o a_2, pero no ambas a la vez.

Los operadores *Feasible, Done* y *Agent* son introducidos para habilitar el razonamiento sobre las acciones, tal como sigue:

Feasible(a, p) significa que *a* puede tomar el lugar y si lo hace, *p* será verdadero sólo después de eso.

Done(a, p) significa que *a* acaba de tomar el lugar y *p* fue verdadero antes de eso.

Agent(i, a) significa que *i* es el agente que puede emprender las acciones que aparecen en la expresión de acción *a*.

Single(a) significa que *a* es una expresión de acción que no es una secuencia. Cualquier acción individual es *Single*. La acción compuesta *a ; b* no es *Single*. La acción compuesta *a | b* es *Single* si y solo si ambos *a y b* son *Single*.

Los componentes del modelo de una acción comunicativa que están implicados en un proceso de planificación caracterizan tanto las razones por las cuales la acción sea seleccionada (es decir, que condiciones deben satisfacerse para que la acción tenga éxito) como las condiciones que tienen que ser satisfechas para que la acción pueda ser planeada. Para una acción dada las anteriores se refieren al efecto racional o RE (*Rational Effect*), y las últimas como las condiciones previas de viabilidad o FP (*Feasibility Preconditions*).

Un modelo de acción comunicativa se presenta de la siguiente manera:

$<i, Act(\ j,\ C\ )>$

$FP:\ p_1$

$RE:\ p_2$

donde *i* es el agente que realiza una acción comunicativa, *j* es el agente destinatario, *Act* el nombre de la acción comunicativa, *C* es el contenido semántico o el contenido preposicional del mensaje, y p_1 y p_2 son proposiciones. Representando de esta manera un mensaje de la **Fig. 8** se obtiene:

(Act
:sender i
:receiver j
:content C)

Para ilustrar la semántica de ACL, consideremos tres actos de mayor uso en SMA: INFORM y REQUEST[8]. La selección de estos dos actos no es arbitraria: se puede decir que la mayoría de otros actos se puede ver como macro-definiciones construidas con base en estos dos. El acto de informar es uno de los más interesantes actos comunicativos con respecto a la esencia de actitudes mentales. La meta del agente quien informa es hacer al otro agente creer algo. Formalmente:

$<i, INFORM\ (\ j,\ \varphi\ )>$

$FP:\ B_i\varphi \wedge \neg B_i\ (Bif_j\varphi \vee Uif_j\varphi\ )$

$RE:\ B_j\varphi\ ,$

[8] Para consultar la lista completa de actos comunicativos de FIPA-ACL y su respectiva semántica le sugerimos al lector consultar la especificación de FIPA en www.fipa.org.

donde $Bif_i\varphi \equiv B_i\varphi \vee B_i\neg\varphi$ significa que el agente i cree que φ o cree $\neg\varphi$.

Un Agente i puede informarle a un agente j que alguna proposición p es verdadera si y solo si i cree p (es decir, sólo si B_ip). También, se considera que esta acción sólo es relevante en el contexto si i no piensa que j ya cree p o su negación, o que j tiene incertidumbre sobre p. Si i ya sabe que j ya cree p, no hay necesidad de la acción realizada por i. Si i cree que j no cree p, i debe desconformar p (es decir utilizar otro acto comunicativo DISCONFIRM). Si j tiene incertidumbre sobre p, i debe confirmar p (a través del acto CONFIRM). De esta manera, las condiciones de factibilidad (FP) se han construido para asegurar la mutua exclusividad entre diferentes actos comunicativos, cuando más de uno pueda generar el mismo efecto racional. Para los fines prácticos, es importante la interpretación de las creencias en la segunda parte de FP dado que se trata de modalidades anidadas, es decir de creencias de un agente sobre las creencias del otro. Se interpretará $\neg B_i$ de tal manera que si el agente en su base de creencias no tiene creencias sobre $(Bif_j\varphi \vee Uif_j\varphi)$ entonces se cumple la condición. En el caso contrario (si se le obligaría al agente tener estas creencias anidadas), estaría muy difícil de cumplir con este requerimiento en un escenario real.

En el caso de un acto comunicativo REQUEST:

$<i, request(j,a)>$
$FP: B_iAgent(a,j) \wedge\neg B_i(I_j(Done(a))$
$RE: Done(a)$

donde $I_j(a)$ significa que el agente j tiene la intención (esta comprometido) de ejecutar la acción α. Como observamos, las condiciones de factibilidad le obligan al agente i creer que α esta dentro de las capacidades del agente j (esto se puede realizar utilizando los servicios de paginas amarillas de una plataforma de agentes – ver en las secciones posteriores más detalles) y, por otra parte, no creer que el agente j ya esta comprometido ejecutar α. Para la interpretación de esta segunda condición es cierto el comentario anterior que hicimos para el INFORM.

Cabe mencionar que debido a las críticas del modelo semántico de FIPA-ACL, recientemente fue propuesta una semántica corregida junto con la semántica de protocolos de interacción [26]. Para definir la semántica de ACL, los autores introducen un lenguaje que combina una forma extendida de la Lógica Proposicional Dinámica (PDL por sus siglas en Ingles de *Propositional Dynamic Logic*) con las modalidades de Creencias e Intenciones. Sin embargo, por las restricciones de espacio en este capítulo, lo dirigimos al lector a esta referencia para los detalles.

5.2 Lenguajes de contenido y ontologías

El ACL representa uno de los pilares de comunicación entre agentes que les permite a ellos comunicar información y conocimiento. Sin embargo, la interoperabilidad de agentes no se limita a ACL: de acuerdo con la **Fig. 9**, otros componentes importantes de comunicación es el lenguaje de contenido y el lenguaje de ontologías. Cada agente tiene una vista parcial del entorno (**Fig. 5**), pero ninguno es capaz de percibir el entorno de forma completa. Sin embargo, se presentan dos situaciones: i) cuando dos agentes tienen sus esferas de interés independientes y ii) cuando haya intersección entre modelos. Para que un SMA sea interoperable se requiere que i) los agentes intercambien su conocimiento codificado en el mismo formato y ii) que interpreten el significado de los conceptos contenidos en el mensaje de forma similar. Cuando un agente i se comunica con el agente j (**Fig. 10**), la información (predicados o proposiciones del dominio, acciones, etc.) se transfiere entre ellos en el parámetro de contenido del mensaje ACL codificado en algún lenguaje. FIPA recomienda que el mismo lenguaje SL que empleamos para definir la semántica de ACL, se utilice para intercambiar los modelos del mundo de agentes. SL incluye un número de operadores útiles, como, por ejemplo operadores lógicos (AND, OR, NOT) y operadores modales (BELIEF, INTENTION, UNCERTAINTY).

Para los agentes programados en algún lenguaje orientado a objetos, sea Java o C# (que es una práctica común, por ejemplo usando plataformas JADE o CAPNET), es impráctico representar internamente esta información en SL. La forma más común sería representar esta información como un objeto. En este caso, según la **Fig. 10**, el agente i necesita convertir este objeto en una representación en el lenguaje de contenido y j debe hacer todo al revés. Más aun, j debe realizar varias pruebas semánticas para asegurarse que la información recibida en realidad tiene el significado acorde a las reglas semánticas de ACL (por ejemplo, si el mensaje es un REQUEST entonces el contenido contiene un nombre válido de una acción) y del modelo de dominio. Estas reglas se pueden codificar en una ontología compartida entre i y j.

Definición 8. *Ontología es una especificación explícita formal de una conceptualización compartida.* [17].

¿Por que necesitamos una ontología? Para que se pueda acordar la terminología que describe un dominio compartido entre varios agentes. Tim Berners-Lee considera ontologías como una parte crítica de la Web Semántica, "la extensión de la Web actual donde a la información se le da un significado bien definido habilitando mejor cooperación entre computadoras y los humanos." [3].

En el contexto de agentes, una ontología para un dominio particular es un conjunto de esquemas que definen la estructura de los predicados, acciones de agentes y conceptos (básicamente sus nombres y ranuras) pertinentes al

dominio. Actualmente, el lenguaje de ontologías de mayor uso en los escenarios Web es OWL que esta basado en RDF: *Resource Description Framework*. RDF estandariza la descripción de meta-datos sobre los recursos Web utilizando tripletas: object-attribute-value – *A(O,V):* objeto *O* tiene atributo *A* con valor *V*. La modificación de RDF (FIPA-RDF) también puede considerarse como alternativa de codificación de contenido de mensajes ACL.

6. Aprendizaje en sistemas multiagente

El aprendizaje en SMA se considera como uno de los pilares del enfoque permitiendo construir agentes adaptables y autónomos que recientemente ha llamado la atención en la investigación sobre agente [45]. Hay diferentes ángulos de los cuales podemos ver la característica de aprendizaje de agentes. Por una parte, un agente debe adaptarse a su entorno y al comportamiento de otros agentes. Hay varias maneras para hacer un agente adaptable: dentro de las cuales se destacan los modelos de preferencias del usuario, mecanismos de adaptación abstracta de organizaciones humanas (mercados) y sistemas naturales (evolución). Sin embargo, desde el punto de vista de aprendizaje individual, un agente puede utilizar cualquier modelo conocido de aprendizaje automático.

Por otra parte, la meta de la investigación de los sistemas multiagentes es encontrar los métodos que nos permitan construir sistemas complejos compuestos por agentes autónomos que, mientras estén funcionando con conocimiento local y posean solamente capacidades limitadas, son no obstante capaces de decretar los comportamientos globales deseados. De allí la necesidad (y posibilidad) de aprendizaje colectivo, es decir en el caso de agentes inteligentes cooperativos es la necesidad de adaptar su comportamiento a los objetivos globales del sistema. En este aspecto, se presenta el problema de balance entre un comportamiento óptimo local de agente y comportamiento óptimo global del sistema.

6.1 Aprendizaje colectivo basado en la teoría de inteligencias colectivas

Revisaremos un modelo de aprendizaje colectivo desarrollado en [29,38] en el contexto de la teoría de INteligencia COlectiva (INCO) [46] como una extensión de los algoritmos de aprendizaje por refuerzo (por ejemplo Q-learning). Dentro del marco de esta teoría, un SMA es conceptualizado como un sistema en línea (que aprende interactuando con un ambiente real y observando los resultados) donde:

– El ambiente es de una naturaleza dinámica e incierta; su modelo inicial de comportamiento es desconocido.
– Cada agente tiene un comportamiento autónomo y una función local de utilidad.
– El control y la comunicación entre los agentes es descentralizado.
– Los agentes adaptan su comportamiento local a los cambios en el ambiente, ejecutando algoritmos del Aprendizaje por Refuerzo (AR) con el objetivo de optimizar el comportamiento global de SMA.
– Los agentes utilizan una técnica de aprendizaje colectivo como una generalización del algoritmo Q-neural para optimizar el comportamiento global.

El aprendizaje por refuerzo (véase también el capítulo 5 de este libro) amplía las ideas de la programación dinámica para encontrar metas más completas y ambiciosas que se encuentran en situaciones del mundo real. Contrario de la programación dinámica, la cual es basada solamente en la resolución de los problemas de control óptimo, AR es un agregado de ideas de la psicología, de la estadística, de la ciencia cognoscitiva y de la ciencia computacional. AR contesta a la pregunta: ¿Cómo hacer el mapeo entre los estados (de un ambiente cuyo modelo no es completamente conocido) y las acciones de un agente (el cual interactúa con su ambiente en línea) para maximizar/minimizar una señal numérica de premio/castigo? Es decir, esto permite buscar lo óptimo de las decisiones locales de cada agente bajo restricciones del comportamiento óptimo del sistema asumiendo la ausencia del modelo matemático inicial. El hecho de aprender por prueba y error, y tener un premio/castigo con retraso que afectará el comportamiento futuro del agente, son las dos características más importantes que lo distinguen de otros tipos de métodos de aprendizaje.

Uno de los avances más importantes del campo de AR fue el desarrollo de Q-aprendizaje (Q-Learning), un algoritmo que sigue una estrategia de aprendizaje fuera de línea [41]. En este caso, la función aprendida de acción-valor, Q, aproxima la óptima función de acción-valor Q* independientemente de la estrategia seguida. En el estado *x(t)*, si los valores Q representan el modelo del ambiente de una manera exacta, la mejor acción a tomar será la que tiene el valor importante mayor/menor (según sea el caso) entre todas las acciones posibles del agente k $a_{i,k} \in A_i$.

Los valores de Q son aprendidos usando una regla de actualización la cual usa una compensación *r(t+1)* calculada por el ambiente y una función de valores Q de los estados alcanzables tomando la acción $a_{x(t)}$ en el estado *x(t)*. La regla de actualización de Q-aprendizaje esta definido por :

$$Q_{(x(t),a_{x(t)})}(t+1) = Q_{(x(t),a_{x(t)})}(t) + \alpha\left[r(t+1) + \gamma \min_{a_{x(t+1)}} Q_{(x(t+1),a_{x(t+1)})}(t+1) - Q_{(x(t),a_{x(t)})}(t) \right] \quad (\mathbf{14})$$

Donde:

- α es la velocidad de aprendizaje, γ es la velocidad de reducción. La velocidad de aprendizaje α determina la dinámica de cambios de Q-valores.
- Los Q-valores $Q_{(x(t),a_{x(t)})}$ dan una estimación del ambiente. La manera en la cual los valores Q son actualizados puede ser considerada como uno de los problemas más importantes para resolver en el modelo.
- El refuerzo por la acción desempeñada es representado por $r(t+1)$. El resultado de esta función puede representar cualquier costo relacionado con la realización de una acción en particular
- Este método AR hace posible resolver problemas de aprendizaje para un único agente. Sin embargo, cuando varios agentes actúan en un ambiente común, estos métodos no son muy eficientes.

Cada agente tiene las siguientes características:

- El conjunto de estados del ambiente $X = \{x_1, x_2, x_3, ...\}$. El conocimiento acerca de otros agentes es considerado ser parte del estado del ambiente.
- La capacidad del agente para actuar es representado como un conjunto de acciones: $A_i = \{a_1, a_2, ..., a_k\}$.
- Las relaciones entre los agentes son definidos por: $R = \{r_1, r_2, r_3, ...\}$. Los agentes conocidos por el agente actual forman la lista de sus vecinos dentro de la estructura organizacional de SMA: $N = \{n_1, n_2, n_3, ...\}$. Para cada agente vecino, se consideran los parámetros siguientes: a) su relación al agente actual, b) la naturaleza del acuerdo que gobierna la interacción y c) los derechos de acceso a la información compartida (el estado local del agente para ser considerado durante el proceso de toma de decisiones).
- La Función de Utilidad Local (*Local Utility Function* - LUF) es representado como la ecuación de Q-aprendizaje (14).
- El conjunto de elementos del control: $C = \{c_1, c_2, c_3, ...\}$. Se invoca un elemento control cuando hay una decisión que tomar mientras se procesa un mensaje
- Cada agente tiene un manejador de mensajes que sea responsable del envío y recepción de diferentes mensajes para facilitar la comunicación entre agentes

Los agentes están involucrados en una actividad colaborativa ejecutando planes en conjunto. Supongamos que en este modelo solo aquellos agentes que tienen capacidades para ejecutar las acciones adyacentes de los planes se conocen entre si y se consideran vecinos (los planes se pueden definir por los procesos tecnológicos como en el ejemplo de la sección 6.2). En el caso de que un plan tiene el fragmento a_{i-1}, a_i, a_{i+1}, este se asigna a los agentes Ag_{i-1}, Ag_i Ag_{i+1} respectivamente.

A continuación se describe un algoritmo de aprendizaje colectivo llamado *Q-neural*. El comportamiento de *Q-neural* fue inspirado por los algoritmos de aprendizaje por refuerzo y la teoría de INCO, especialmente los algoritmos basados en el comportamiento de las colonias de hormigas. El aprendizaje se realiza en dos niveles: localmente, en el nivel del agente que actualiza los valores Q usando la regla de AR, y globalmente, en el nivel de sistema ajustando la función de utilidad. Los mensajes de control permiten actualizar el conocimiento de cada agente por la actualización de valores Q, los cuales son aproximados por el uso de un aproximador de la función (tabla de búsqueda, red neuronal, etc). En *Q-neural* hay cinco tipos de mensajes de control:

1. *Un 'mensaje-ambiente' (flag_ret=1)* se genera por un agente intermedio después de la recepción de la solicitud de ejecutar una acción del plan si el intervalo de tiempo ω ya paso.
2. *Un 'mensaje-hormiga' (flag_ret=2)* generado por el agente que ejecute la última acción del plan.
3. *Un 'mensaje-actualización' (flag_ret=3)* generado en la fase de planeación (en términos de aprendizaje automático) cada $\varepsilon\_\_update$ segundos para solicitar a los agentes sus estimaciones sobre su capacidad de ejecutar las acciones del plan.
4. *Un 'mensaje-actualización-regreso' (flag_ret=4)* generado después de la recepción del mensaje-actualización para acelerar el aprendizaje del ambiente.
5. *Un 'mensaje-castigo' (flag_ret=5)* usado para castigar al agente por el uso de un recurso congestionado.

Estos mensajes se utilizan dentro del algoritmo *Q-neural* integrado por los algoritmos de planeación, mensaje-hormiga y castigo, cada uno de ellos implementando una funcionalidad específica permitiendo la optimización descentralizada.

El algoritmo de planeación (**Algoritmo 1**) explora las mejores opciones (en términos de las estimaciones de los valores Q) que resultan de cambios inesperados en el ambiente. La velocidad de exploración ε indica el porcentaje ($0 \leq \varepsilon < 100$) de tiempo cuando el agente seleccionará al siguiente agente en forma aleatoria. La exploración puede implicar pérdida de tiempo significativo. En *Q-neural*, un mecanismo de planeación fue desarrollado a nivel local de cada agente. Este mecanismo consiste en enviar un *mensaje-actualización* cada ε_update de segundos. Este *mensaje-actualización* pedirá las estimaciones de los valores Q de todos los planes, conocidos hasta ese momento por los agentes vecinos para ir actualizando las estimaciones Q del estado del ambiente de cada agente.

Algoritmo 1: Planeación
Cada ε_update

Enviar un 'mensaje-aclualización' (flag_ret = 3) a cada vecino para solicitar sus estimaciones de todos los planes conocidos

if *('mensaje-aclualización' es recibido) (flag_ret = 3)*

Enviar un 'mensaje-actualización-regreso' (flag_ret = 4) al agente emisor de 'mensaje-aclualización' con las estimaciones $Q^{Ag_i}_{(x(t),a_{x(t)})}(t)$ *de todos los planes conocidos en el instante* t

if *('mensaje-actualización-regreso' es recibido) (flag_ret = 4)*

Actualizar los valores Q en la misma manera que se usa para un 'mensaje-ambiente' (flag_ret = 1)

Los *Mensajes-Hormiga* se utilizan como retroalimentación del sistema: cada mensaje intercambia la estadística obtenida en su camino (también en términos de valores Q) y permite la comunicación de la información del ambiente entre los agentes (**Algoritmo 2**). Cuando un *mensaje-ambiente* llega al agente que termina la ejecución del plan, una hormiga es enviada de regreso si ha pasado el período de tiempo ω_ants. Cuando llega al agente iniciador del plan, éste muere. La hormiga pone al día el valor Q de cada agente intermedio que participó en la ejecución del plan.

Finalmente, el algoritmo del castigo intenta identificar y resolver conflictos (entre las mejores estimaciones) entre los agentes vecinos (**Algoritmo 3**). En algunos casos, diversos agentes pueden tener la misma mejor estimación (es decir, prefieren los mismos agentes para ejecutar la siguiente acción de sus planes). Si actúan de una manera codiciosa, la congestión ocurre en la cola. Para evitar congestiones, un agente debe sacrificar su utilidad individual y utilizar alguna otra ruta. Para tratar este problema, es desarrollado un algoritmo del castigo que fuerza a un agente a que reciba un *mensaje-castigo* para calcular la segunda mejor estimación.

Algoritmo 2: Hormiga

if *(un se termina la última acción del plan)*

then *el último agente genera un 'mensaje-hormiga' si el tiempo* w_ant *ha pasado*

if *un agente* Ag_{i-1} *recibe el 'mensaje-hormiga' de un agente vecino* Ag_i*)*

then

Leer la estimación $Q^{A}_{(x(t),a_{x(t)})}(t)$ *del encabezado del 'mensaje-hormiga' (flag_ret = 2)*

Consultar la mejor estimación en el momento de tiempo actual t $Q^{Ag_{i-1}}_{(x(t),a_{x(t)})}(t)$

if $Q^{Ag_{i-1}}_{(x(t),a_{x(t)})}(t) > Q^{A}_{(x(t),a_{x(t)})}(t)$ *y (un ciclo no se detecta)*

then *actualizar el valor Q por* $Q^{A}_{(x(t),a_{x(t)})}(t)$

else *no actualizar*

El algoritmo *Q-neural* utiliza los algoritmos antes mencionados para obtener una retroalimentación del ambiente y para alinear las recompensas locales con el objetivo global (**Algoritmo 4**).

Algoritmo 3: Castigo

if *un 'mensaje-castigo' es recibido por* Ag_i *de* Ag_{i+1}

calcular la segunda mejor estimación $Q^{Ag_i}_{(x(t),a_{x(t)})}(t)$ *para llegar al final del plan usando otra ruta que no fuera* $l_{Ag_i,Ag_{i+1}}$

Enviar un mensaje a todos los agentes productores vecinos Ag_{i-1} *solicitando la segunda mejor estimación.*

Recibir la segunda mejor estimación de todos los agentes vecinos Ag_{i-1}.

Seleccionar la mejor estimación: $\arg\min\ Q^{Ag_{i-1}}_{(x(t),i')}(t)$

if *(la segunda estimación* $Q^{Ag_i}_{(x(t),i')}(t)$ *existe)*

if *(la mejor estimación* $\arg\min\ Q^{Ag_{i-1}}_{(x(t),i')}(t)$ *de los vecinos existe)*

if $((Q^{Ag_i}_{(x(t),i+1')}(t) < (\arg\min\ Q^{Ag_{i-1}}_{(x(t),i')}(t))))$

Castigar la línea $l_{Ag_i,Ag_{i+1}}$*:* $Q^{Ag_i}_{(x(t),i+1)}(t)$ *= segunda estimación* $Q^{Ag_i}_{(x(t),i+1')}(t)+\Delta$

else

Castigar la línea l_{Ag_{i-1},Ag_i} *: enviar un 'mensaje-castigo'*

else

Castigar la línea $l_{Ag_i,Ag_{i+1}}$*:* $Q^{Ag_i}_{(x(t),i+1)}(t)$ *= segunda estimación* $Q^{Ag_i}_{(x(t),i+1')}(t)+\Delta$

else

if *(la segunda mejor estimación* $\arg\min\ Q^{Ag_{i-1}}_{(x(t),i')}(t)$ *de los vecinos existe)*

Castigar la línea l_{Ag_{i-1},Ag_i} *:* $Q^{Ag_{i-1}}_{(x(t),i)}(t)$ *= segunda estimación* $Q^{Ag_{i-1}}_{(x(t),i')}(t)+\Delta$

Enviar un 'mensaje-castigo' con la estimación

El primer paso del algoritmo consiste en fijar inicialmente los valores Q y los parámetros de AR tales como α,γ y exploración. Entonces, cada agente ejecutando el algoritmo *Q-neural* lee un encabezado con información acerca del ambiente del otro agente y envía mensajes de retroalimentación a otros agentes. El paso siguiente consiste en ejecutar la acción óptima según la política aprendida. Entonces, el agente recibe una recompensa o un castigo del ambiente (un agente puede ser parte del ambiente para otro agente). La planeación, el castigo, y los algoritmos-hormiga se ejecutan simultáneamente y ayudan en la reducción de tiempo para encontrar una política local óptima para cada agente que ejecuta el algoritmo.

Algoritmo 4: "*Q-neural*"

Inicializar en el momento $t=0$*: todos los Q-valores* $Q_{x(t),a_{x(t)}}$ *con valores altos, los parámetros de AR:* α,γ*, exploración,* w, w_ants

REPEAT

Actualizar la instante t

if *el agente* Ag_i *recibe la solicitud de ejecución de la acción* a_i

Leer el vector de entrada x *del encabezado de la solicitud y las variables del ambiente*

Enviar un mensaje al agente Ag_{i-1} *que la solicitud llega con el valor de la función de refuerzo* $r_{(t+1)}$ *y la estimación* $Q^{Ag_i}_{(x(t),a_{x(t)})}(t)$

Ejecutar la acción a_i *y asignar la siguiente acción al mejor agente* $a_{x(t)}=Ag_{i+1}$ *en función del vector de entrada* x *usando la estrategia* ε_greedy *derivada de* $Q^{Ag_i}_{(x(t),a_{x(t)})}(t)$

Enviar la solicitud $a_{x(t)}=Ag_{i+1}$

En el siguiente paso de tiempo, recibir el mensaje del agente Ag_{i+1} *con el valor de la función de refuerzo* $r(t+1)$ *y la estimación* $Q^{Ag_{i+1}}_{(x(t+1),a_{x(t)+1})}(t+1)\, Q_{x(t+1),a_{x(t+1)}}(t+1)$

Aplicar la regla de actualización de Q-aprendizaje:

$$Q_{(x(t),a_{x(t)})}(t+1) = Q_{(x(t),a_{x(t)})}(t) + \alpha\left[r(t+1) + \gamma \min_{a_{x(t+1)}} Q_{(x(t+1),a_{x(t+1)})}(t+1) - Q_{(x(t),a_{x(t)})}(t) \right]$$

REPEAT

Algoritmo de planeación ε_update

Algoritmo de castigo

Algoritmo de hormiga

En la siguiente sección revisaremos los detalles de implementación de los algoritmos de aprendizaje colectivo en un ejemplo de ruteo de trabajos en un taller de manufactura.

6.2 Ejemplo de aprendizaje colectivo para el ruteo de trabajos

El problema consiste en planificar de la mejor manera un proceso justo-a-tiempo (*just-in-time*) de manufactura. Este proceso asume que no existen las rutas pre-determinadas (planeadas con anterioridad por los mecanismos de calendarización) para realizar operaciones, es decir, se debe seleccionar la siguiente maquina (que implemente la operación definida en el proceso tecnológico del producto) después de que termine cada operación. Nuestro sistema está compuesto por los siguientes elementos (**Fig. 11**):

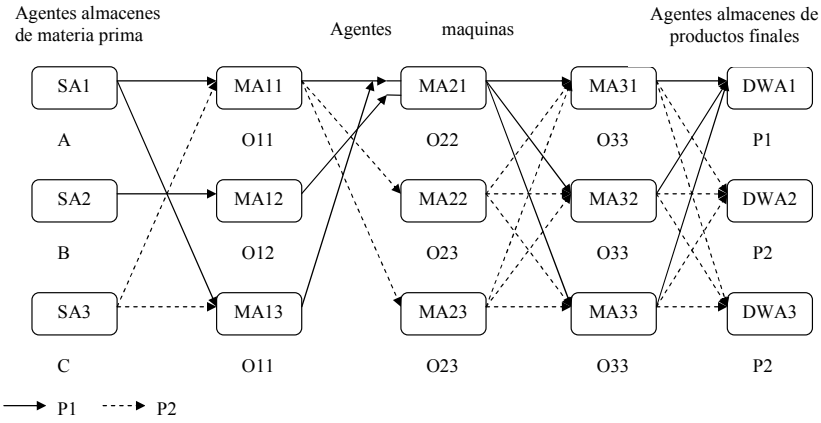


Fig. 11. Conceptualización de flujo multi-producto en un taller de manufactura

- Un conjunto de máquinas $M = \{M_1, M_2, ..., M_m\}$ las cuales pueden ejecutar determinadas operaciones. Las maquinas están distribuidas por capas de acuerdo a los tipos de operaciones que pueden ejecutar dentro de un proceso tecnológico.
- Un conjunto de productos (o trabajos) $P = \{P_1, P_2, ..., P_n\}$, donde cada producto P requiere de una serie de operaciones. En la **Fig. 11** solo se presentan dos productos.
- Un conjunto de almacenes de materia prima (componentes A, B y C en el ejemplo) y productos finales $S = \{S_1, S_2, ..., S_l\}$
- Un conjunto de operaciones $O = \{O_1, O_2, ..., O_k\}$, donde las operaciones forman procesos tecnológicos de cada producto P_j, por ejemplo, $P1 = ((AO_{11} + BO_{12})O_{22})O_{33}$ y $P2 = ((CO_{11})O_{23})O_{33}$.
- Un conjunto de agentes Ag={1,...,n} (**Fig. 12**) asociados a cada almacén de materia prima (SA), maquina (MA) y almacén de productos finales (DWA). Para cada agente se tiene una tabla de Q-valores asociada (Q-tabla), en base a la cual se selecciona la acción *a(t)* de acuerdo al valor mínimo de Q.

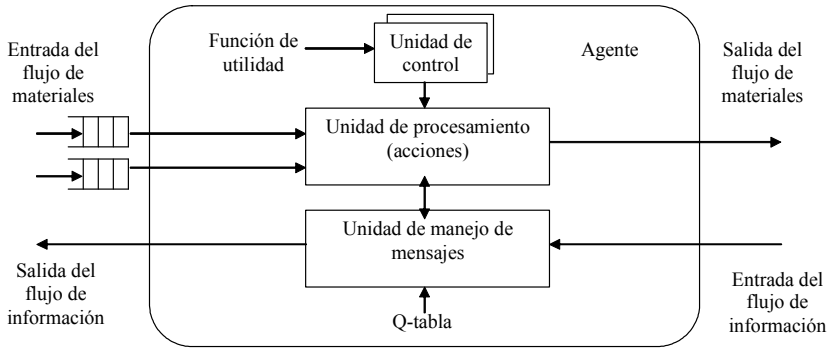


Fig. 12. Vista genérica de un agente dentro del marco de INCO

Primero veamos las fases principales del algoritmo Q-neural que permite que un agente ajuste dinámicamente sus decisiones al estado actual del sistema. Cada vez que la mejor opción va a ser seleccionada, buscan al mejor oferente, la mejor operación de proceso, el mejor almacén de distribución, etc. dentro de las tablas Q. La **Fig. 13** describe la inicialización de los Q-valores de cada agente en el SCN. Los valores sobre agentes enlazados representan un costo lógico de tomar una acción referida al miembro implicado. Pues puede ser notado, en la primera iteración del algoritmo, el Q-valor es este costo lógico (tiempo previsto de la operación y de la transición).

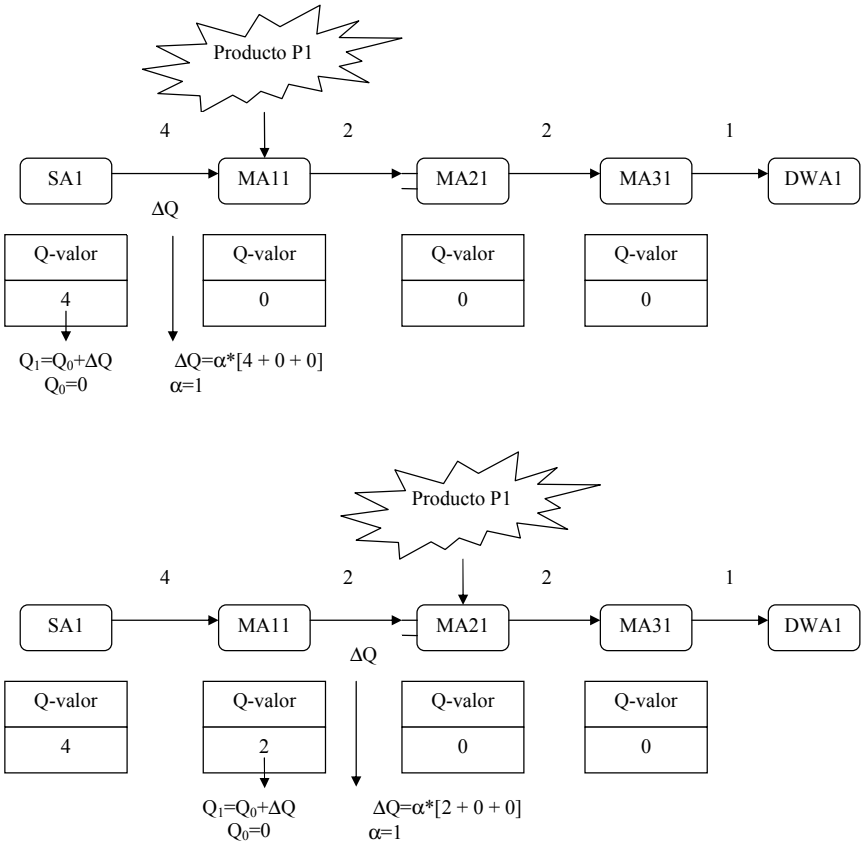


Fig. 13. Algoritmo Q-aprendizaje: inicialización de las Q-tablas par alas primeras dos operaciones del producto P1, ΔQ se calcula con la formula (1).

Una vez que el sistema consiga estar implicado en la operación, se ejecuta el algoritmo de planeación. Este algoritmo representa la parte pro-activa de nuestro enfoque. Con este algoritmo, no es necesario ejecutar una acción para obtener una regeneración del ambiente: cada agente construye un estado interno local del ambiente. Según lo mostrado en **Fig. 14** para el agente SA3 (componente C del producto P2), durante la ejecución del algoritmo de planeación se obtiene el mínimo Q-valor de todos los vecinos del agente MA11: $\min Q^{MA_{31}}_{(\vec{x}(t),a_{\vec{x}(t)})}(t) = 17.90$. Los valores Q del agente MA11 que le corresponden al producto P1 no se consideran (fila sombreada de la Q-tabla). Se suma al valor obtenido en el paso anterior el tiempo que tarda el producto P2 (el componente C) de ir de SA3 a MA11 (6 unidades):

$\min Q^{M_{x_1}}_{(\bar{x}(t),a_{\bar{x}(t)})}(t) + t_{s\to x_1} = 17.90 + 6 = 23.9$. En este momento, se actualiza el Q-valor del agente SA3 respecto al MA11: 23.9 en vez de 26.5.

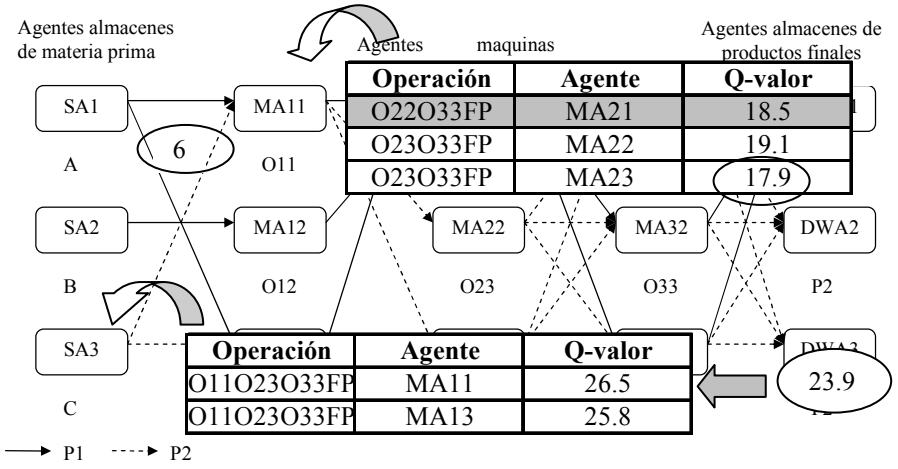


Fig. 14. Algoritmo de planeación: actualización de los Q-tablas del agente almacén SA3 (producto P2).

Finalmente, la **Fig. 15** ilustra el algoritmo hormiga. El objetivo principal de cada hormiga es comunicar cuanto antes las buenas y malas noticias sobre los cambios en el ambiente. Supongamos que en cada fase de operación el componente B del producto P1 tardó 6 unidades de tiempo. Según lo mostrado, los agentes MA12 y MA21 tienen que poner al día sus respectivos Q-valores basados en el mensaje de la hormiga. Estos valores representan el estado verdadero del sistema en el momento que la última instancia del producto P1 fue entregado al agente-almacén de productos finales DWA1.

7. Desarrollo de agentes inteligentes y SMA

Los desarrollos teóricos de diferentes facetas de agentes inteligentes y SMA han dado lugar al paradigma de computación basada en agente (que también se conoce como 'computación orientada a agentes' o 'programación orientada a agentes')[19]. Los programas orientados a agentes agregan el concepto explícito de agentes autónomos al mundo de objetos pasivos integrando componentes activos con capacidades individuales del razonamiento e interacción (**Fig. 2**). La necesidad de programar diferentes facetas de un agente inteligente (estados mentales y aprendizaje, conocimiento y comunicación, movilidad y planeación, etc.) que a menudo tienen sus propios

modelos y lenguajes de programación marca la principal dificultad de desarrollo de los SMA.

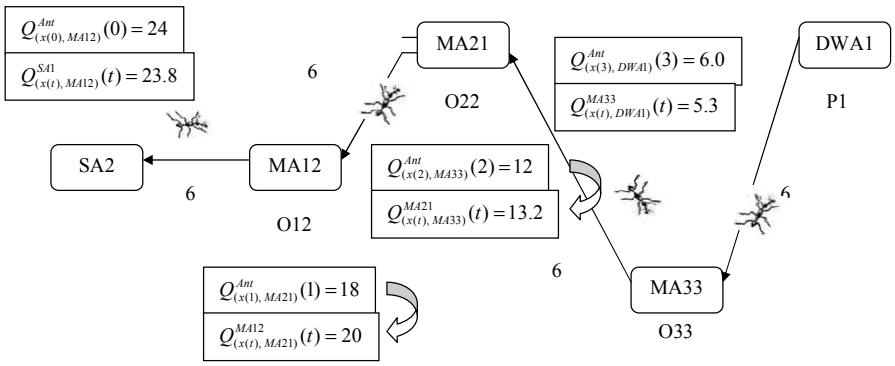


Fig. 15. Algoritmo de hormiga: actualización de Q-valores por medio de 'mensajes-hormiga' para una instancia particular del producto P1 que terminó su ciclo de producción (llegó al almacén de productos finales).

Lenguajes de programación de agentes siempre han sido un punto débil del paradigma de agentes. Por un lado, obviamente un nuevo paradigma de programación requiere de un enfoque implementado en modelos, lenguajes y herramientas de programación que lo distinguen de otros paradigmas. Dado que el paradigma de agentes se ha conceptualizado como el sucesor de la programación orientada a objetos [19], surge la pregunta ¿cual es su modelo de programación? En el famoso artículo de Yoav Shoham, la Programación Orientada a Agentes (POA) fue conceptualizada a través de agentes que implementaban la arquitectura BDI y se programaban en el lenguaje Agent-0 [39].

Desde entonces la evolución de la POA se desarrollaba de una forma muy parecida a la evolución del paradigma mismo que hemos mencionado al principio de este capítulo: i) para la programación de agentes inteligentes se desarrollaban los lenguajes sucesores de Agent-0 bajo la arquitectura BDI y, por otro lado, ii) agentes de software se programan en algún lenguaje de programación orientada a objetos pero tanto el desarrollo de agentes como su ejecución se realiza sobre una infraestructura conocida como *Plataforma de Agentes* (PA). Obviamente, hoy en día existe la posibilidad de fusionar estos dos enfoques y lenguajes de programación de agentes BDI están disponibles como extensión de PA. En esta sección les presentamos una breve introducción a éste tema.

7.1 Lenguajes de programación de agentes inteligentes

Como lo hemos visto en secciones anteriores de éste capítulo, agentes inteligentes se conceptualizan usualmente en términos de conocimiento, creencias, deseos, intenciones, compromisos, metas y planes. Si el SMA, sin embargo, se implementara usando un lenguaje de programación arbitrario, sería difícil de validar si realmente cumple sus especificaciones expresadas en estos términos. A este problema M. Wooldridge refiere como la semántica infundada de lenguajes de especificación de agentes [47]. Más aun, será más difícil pasar de la especificación a la implantación si no existe una correspondencia clara entre los conceptos usados en la especificación y en la programación.

De ahí, en esta sección revisaremos dos diferentes enfoques a programación de agentes BDI. Como ejemplos de estos enfoques, consideraremos 3APL (*An Abstract Agent Programming Language*) desarrollado en la Universidad de Utrecht, Holanda y a Jadex desarrollado por the Distributed Systems and Information Systems Group at the University of Hamburg. Cabe mencionar que 3APL es un intérprete de un lenguaje de programación, a cambio, Jadex es un motor de razonamiento para implementar agentes racionales con XML y lenguaje de programación Java, es decir, no introduce ningún nuevo lenguaje de programación.

7.1.1 3APL

3APL es un lenguaje de programación para implementar agentes deliberativos. Éste proporciona construcciones formales para representar creencias, metas y planes de un agente basados en una noción declarativa de metas. Por otra parte, proporciona las construcciones de programación para programar creencias, metas y capacidades básicas (tales como actualizaciones de creencias, acciones externas, o acciones comunicativas) de los agentes y un conjunto de reglas del razonamiento práctico con las cuales las metas pueden ser actualizadas o revisadas. Esta combinación de metas declarativas y procesales (reducida a la de los planes, que se seleccionan para alcanzar metas declarativas) es una de las características atractivas principales de 3APL para desarrollar agentes inteligentes.

Las nociones de creencias y de metas son tradicionales para un agente intencional BDI (véase Sección 3). La base de creencias puede contener la información que el agente cree sobre el mundo y puede contener la información que es interna al agente. Las metas del agente por otra parte, denotan la situación que el agente desea realizar. Las creencias y las metas se pueden representar con un lenguaje del dominio de primer orden. Para alcanzar sus metas, un agente 3APL adopta planes. Un plan es una secuencia construida de elementos básicos. Los elementos básicos pueden ser acciones básicas, pruebas sobre la base de creencias o planes abstractos. El efecto de la ejecución de una acción básica no es un cambio en el mundo, sino un cambio

en la base de creencias del agente. Los planes abstractos sirven como mecanismo de la abstracción como, por ejemplo, procedimientos en la programación imperativa. Si un plan consiste en un plan abstracto, este plan abstracto se podría transformar en acciones básicas a través de reglas del razonamiento.

Varias reglas son propuestas para razonar con metas, planes, y sus interacciones. Las reglas de la meta (GR) se utilizan para revisar, para generar o para desechar metas. Las reglas de la interacción (IR) se utilizan para generar planes para alcanzar metas. La condición de creencias de tales reglas indica cuando un plan se podría seleccionar para alcanzar la meta especificada. Finalmente, las reglas del plan (PR), se utilizan para revisar y para desechar planes. La base de planes del agente es un conjunto de pares de plan-meta (PG). La descripción completa de la sintaxis y de la semántica de 3APL puede ser encontrada en [9].

Cada programa 3APL se ejecuta por medio de un intérprete que delibere sobre las actitudes cognoscitivas de agentes y determine el orden en la cual se aplican las reglas, cuando las acciones se deben realizar, cuando las actualizaciones de creencias deben ser hechas, etc. El ciclo de la deliberación para los agentes 3APL se muestra en **Fig. 16**. La adición de metas declarativas influencia substancialmente el ciclo de la deliberación.

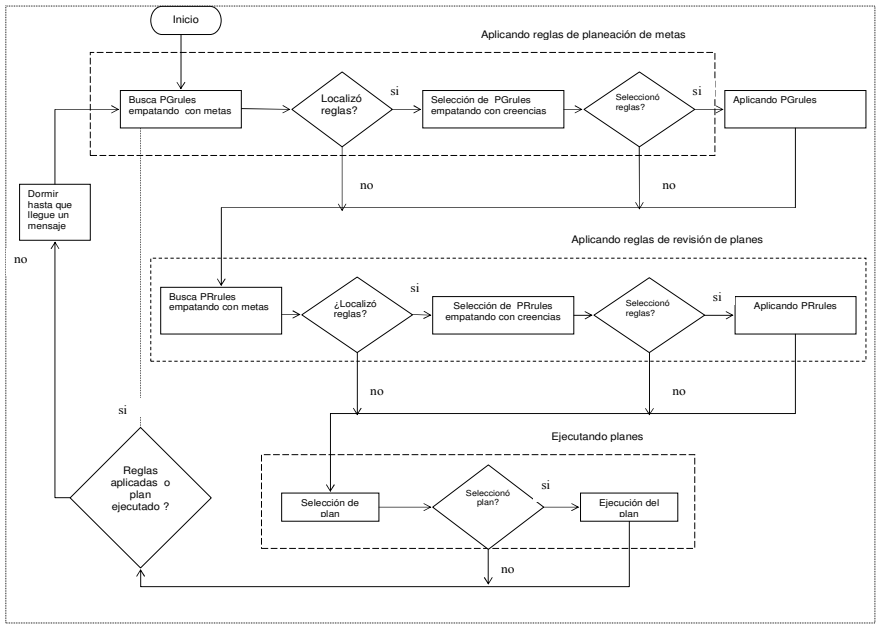


Fig. 16. Ciclo de deliberación de agentes 3APL (intérprete cíclico)

Programar un agente 3APL significa especificar metas y creencias iniciales y escribir sistemas de reglas de la meta, reglas de interacción y reglas de planes.

Capítulo 9.

Procesos de decisión de Markov y aprendizaje por refuerzo

1. Introducción

Un agente, ya sea humano o computacional, se enfrenta continuamente al problema de toma de decisiones bajo incertidumbre. En base a información limitada del ambiente, el agente debe tomar la *mejor* decisión de acuerdo a sus objetivos. En muchas ocaciones este proceso se repite en forma secuencial en el tiempo, de manera que en cada instante el agente recibe información y decide que acción tomar, en base a sus objetivos a largo plazo. A esto se le denomina *problemas de decisión secuencial*. Por ejemplo, un agente de bolsa tiene que decidir en cada momento en base a la información con la que cuenta, que acciones vender y comprar, para maximizar su utilidad a largo plazo. Otro ejemplo es el de un robot que debe navegar de un punto a otro, de forma que en cada instante de tiempo debe decidir sus movimientos, en base a la información de sus sensores y conocimiento del ambiente, para llegar en la forma más directa (de menor costo) a la meta.

Esta clase de problemas se pueden modelar, desde una perspectiva de teoría de decisiones, como *Procesos de Decisión de Markov*; donde se especifican los posibles estados en que puede estar el agente, sus posibles acciones, una función de recompensa basada en sus preferencias, y un modelo de su dinámica representado como una función de probabilidad. Establecido el modelo, se puede encontrar una solución que de la mejor acción para cada estado (política) para maximizar la utilidad del agente a largo plazo. Otra forma de resolver este tipo de problemas es mediante *Aprendizaje por Refuerzo*. En este caso no se tiene un modelo dinámico del agente, por lo que se aprende la política óptima en base a prueba y error explorando el ambiente.

En este capítulo se presenta un panorama general de procesos de decisión de Markov (MDPs) y aprendizaje por refuerzo (RL). En primer lugar se da una definición formal de lo que es un MDP, que se ilustra con un ejemplo. Después se

describen las técnicas de solución básicas, ya sea cuando se tiene un modelo (programación dinámica) o cuando no se tiene (métodos Monte Carlo y aprendizaje por refuerzo). En la sección 4 se presentan otras técnicas de solución alternativas a los enfoques básicos. En seguida se presentan métodos de solución para problemas más complejos, mediante modelos factorizados y abstractos. Finalmente se ilustran estos métodos en 3 aplicaciones: aprendiendo a volar un avión, controlando una planta eléctrica y coordinando a un robot mensajero.

2. Procesos de Decisión de Markov

En esta sección analizaremos el modelo básico de un MDP, así como su extensión a los procesos parcialmente observables o POMDPS.

2.1. MDPs

Un MDP modela un problema de decisión sequencial en donde el sistema evoluciona en el tiempo y es controlado por un agente. La dinámica del sistema está determinada por una función de transición de probabilidad que mapea estados y acciones a otros estados.

Formalmente, un MDP es una tupla $M =< S, A, \Phi, R >$ [29]. Los elementos de un MDP son:

- Un conjunto finito de estados $S : \{s_1, \ldots, s_n\}$, donde s_t denota el estado $s \in S$ al tiempo t.
- Un conjunto finito de acciones que pueden depender de cada estado, $A(s)$, donde $a_t \in A(s)$ denota la acción realizada en un estado s en el tiempo t.
- Una función de recompensa ($\mathcal{R}^a_{ss'}$) que regresa un número real indicando lo deseado de estar en un estado $s' \in S$ dado que en el estado $s \in S$ se realizó la acción $a \in A(s)$.
- Una función de transición de estados dada como una distribución de probabilidad ($\mathcal{P}^a_{ss'}$) que denota la probabilidad de llegar al estado $s' \in S$ dado que se tomó la acción $a \in A(s)$ en el estado $s \in S$, que también denotaremos como $\Phi(s, a, s')$.

Dado un estado $s_t \in S$ y una acción $a_t \in A(s_t)$, el agente se mueve a un nuevo estado s_{t+1} y recibe una recompensa r_{t+1}. El mapeo de estados a probabilidades de seleccionar una acción particular definen una *política* (π_t). El problema fundamental en MDPs es encontrar una política óptima; es decir, aquella que maximiza la recompensa que espera recibir el agente a largo plazo.

Las políticas o reglas de decisión pueden ser [29]: (i) deterministas o aleatorias, (ii) estacionarias o no–estacionarias. Una política determinista selecciona simpre la misma acción para cierto estado, mientras que una aleatoria selecciona con cierta probabilidad (de acuerdo a cierta distribución especificada) una acción de un conjunto posible de acciones. Una política estacionaria decide la misma

acción para cada estado independientemente del tiempo, esto es, $a_t(s) = a(s)$ para toda $t \in T$; lo cual no se cumple para una no–estacionaria. En este capítulo nos enfocamos a políticas estacionarias y deterministas.

Si las recompensas recibidas después de un tiempo t se denotan como: r_{t+1}, r_{t+2}, r_{t+3}, ..., lo que se quiere es maximizar la recompensa total (R_t), que en el caso más simple es:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T$$

Si se tiene un punto terminal se llaman tareas *episódicas*, si no se tiene se llaman tareas *continuas*. En este último caso, la fórmula de arriba presenta problemas, ya que no podemos hacer el cálculo cuando T no tiene límite. Podemos usar una forma alternativa en donde se van haciendo cada vez más pequeñas las contribuciones de las recompensas más lejanas:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

donde γ se conoce como la *razón de descuento* y está entre: $0 \leq \gamma < 1$. Si $\gamma = 0$ se trata de maximizar la recompensa total tomando en cuenta sólo las recompensas inmediatas.

En general existen los siguientes tipos modelos de acuerdo a lo que se busca optimizar:

1. Horizonte finito: el agente trata de optimizar su recompensa esperada en los siguientes h pasos, sin preocuparse de lo que ocurra después:

   $$E(\sum_{t=0}^{h} r_t)$$

   donde r_t significa la recompensa recibida t pasos en el futuro. Este modelo se puede usar de dos formas: (i) *política no estacionaria*: donde en el primer paso se toman los siguientes h pasos, en el siguiente los $h-1$, etc., hasta terminar. El problema principal es que no siempre se conoce cuántos pasos considerar. (ii) *receding-horizon control*: siempre se toman los siguientes h pasos.
2. Horizonte infinito: las recompensas que recibe un agente son reducidas geométricamente de acuerdo a un factor de descuento γ ($0 \leq \gamma < 1$) considerando un número infinito de pasos:

   $$E(\sum_{t=0}^{\infty} \gamma^t r_t)$$

3. Recompensa promedio: al agente optimiza a largo plazo la recompensa promedio:

   $$lim_{h \to \infty} E(\frac{1}{h} \sum_{t=0}^{h} r_t)$$

En este caso, un problema es que no hay forma de distinguir políticas que reciban grandes recompensas al principio de las que no.

El modelo más utilizado es el de horizonte infinito.

Las acciones del agente determinan no sólo la recompensa inmediata, sino también en forma probabilística el siguiente estado. Los MDPs, como el nombre lo indica, son procesos markovianos; es decir, que el siguiente estado es independiente de los estados anteriores dado el estado actual:

$$\mathcal{P}^a_{ss'} = P(s_{t+1} = s' \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = P(s_{t+1} = s' \mid s_t = s, a_t = a)$$

La política π es un mapeo de cada estado $s \in S$ y acción $a \in A(s)$ a la probabilidad $\pi(s, a)$ de tomar la acción a estando en estado s. El *valor* de un estado s bajo la política π, denotado como $V^\pi(s)$, es la recompensa total esperada en el estado s y siguiendo la política π. El valor esperado se puede expresar como:

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\} = E_\pi \left\{ \sum_{k=o}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\}$$

y el valor esperado tomando una acción a en estado s bajo la política π, denotado como $Q^\pi(s, a)$, es:

$$Q^\pi(s, a) = E_\pi\{R_t \mid s_t = s, a_t = a\} = E_\pi \left\{ \sum_{k=o}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\}$$

Las funciones de valor óptimas se definen como:

$$V^*(s) = max_\pi V^\pi(s) \text{ y } Q^*(s, a) = max_\pi Q^\pi(s, a)$$

Las cuales se pueden expresar mediante las ecuaciones de optimalidad de Bellman [3]:

$$V^*(s) = max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$$

y

$$Q^*(s, a) = \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$$

o

$$Q^*(s, a) = \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma \text{max}_{a'} Q^*(s', a')]$$

De esta forma, lo que se busca al resolver un MDP es encontrar la solución a las ecuaciones de Bellman; que equivale a encontrar la política óptima. Existen dos enfoques básicos para resolver MDPs: (i) cuando el modelo es conocido $(\mathcal{P}, \mathcal{R})$, mediante métodos de programación dinámica o programación lineal, y (ii) cuando el modelo es desconocido usando métodos de Monte Carlo o de aprendizaje por refuerzo. También existen métodos híbridos que combinan ambos enfoques. En la sección 3 estudiamos los diferentes métodos de solución.

Ejemplo de un MDP Veremos ahora un ejemplo muy sencillo de como podemos definir un MDP. Supongamos que en un mundo virtual en forma de rejilla (ver figura 1) hay un agente que se desplaza de un cuadro a otro, pudiendo moverse a los cuadros continuos: arriba, abajo, derecha o izquierda. El agente desea desplazarse para llegar a cierta meta (recompensa positiva), evitando los obstáculos en el camino (recompensa negativa). Dado que cada movimiento implica cierto gasto de energía para el agente, cada movimiento tiene asociado cierto costo. Como existe incertidumbre en las acciones del agente, asumimos que con un 80 % de probabilidad llega a la celda desaeada, y con un 20 % de probabilidad cae en otra celda contigua.

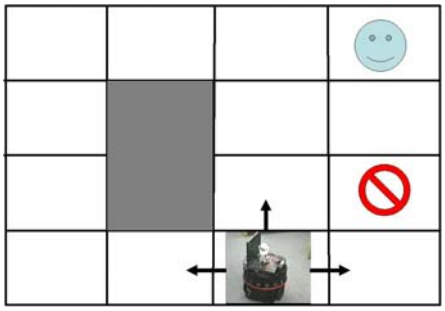


Figura 1. Ejemplo del mundo virtual en que el agente debe navegar. La celda de arriba a la derecha (S_4) representa la meta, con una recompensa positiva; mientras que las celdas grises (obstáculos) y la celda con el signo de *prohibido* (peligro), tienen recompensas negativas. El agente (en el dibujo en el estado s_{15}) puede desplazarse a las 4 celdas vecinas, excepto si está en la *orilla* del mundo virtual.

En base a la descripción anterior, considerando un mundo de 4×4 celdas, podemos definir un MDP que representa este problema:
Estados:

$$S = \begin{Bmatrix} s_1 & s_2 & s_3 & s_4 \\ s_5 & s_6 & s_7 & s_8 \\ s_9 & s_{10} & s_{11} & s_{12} \\ s_{13} & s_{14} & s_{15} & s_{16} \end{Bmatrix}$$

Acciones:

$$A = \{izquierda, arriba, derecha, abajo\}$$

Recompensas:

$$R = \begin{Bmatrix} -1 & -1 & -1 & 10 \\ -1 & -100 & -1 & -1 \\ -1 & -100 & -1 & -10 \\ -1 & -1 & -1 & -1 \end{Bmatrix}$$

La *meta* tiene una recompensa positiva (+10), los obstáculos (-100) y la zona de peligro (-10) negativas. Las demás celdas tiene una pequeña recompensa negativas (-1) que indican el costo de moverse del agente, de forma que lo hacen buscar una trayectoria "corta" a la meta.
Función de transición:

$$\begin{array}{ll} \Phi(s_1, izquierda, \_) = 0{,}0 & \% \text{ no es posible irse a la izquierda} \\ \ldots & \ldots \\ \Phi(s_2, izquierda, s_1) = 0{,}8 & \% \text{ llega al estado deseado, } s_1 \\ \Phi(s_2, izquierda, s_5) = 0{,}1 & \% \text{ llega al estado } s_5 \\ \Phi(s_2, izquierda, s_2) = 0{,}1 & \% \text{ se queda en el mismo estado} \\ \ldots & \ldots \end{array}$$

Una vez definido el modelo, se obtiene la política óptima utilizando los métodos que veremos más adelante.

2.2. POMDPs

Un MDP asume que el estado es perfectamente observable; es decir, que el agente conoce con certeza el estado donde se encuentra en cada instante de tiempo. Pero en la práctica, en muchos casos no es así. Por ejemplo, considerando un robot móvil cuyo estado esta dado por su posición y orientación en el plano, (x, y, θ), es común que la posición no se conozca con certeza, por limitaciones de la odometría y los sensores del robot. Entonces el robot tiene cierta *creencia* sobre su posición actual (su estado), que se puede representar como una distribución de probabilidad sobre los posibles estados. Cuando el estado no es conocido con certeza, el problema se transforma en un proceso de decisión de Markov parcialmente observable (POMDP).

En forma similar a un MDP, un POMDP se define formalmente como una tupla $M =< S, A, \Phi, R, O >$. Los elementos S, A, Φ, R se definen de la misma forma que para un MDP. El elemento adicional es la Función de Observación, O, que da la probabilidad de observar o dado que el robot se encuentra en el estado s y ejecuto la acción a, $O(s, a, o)$.

Resolver un POMDP es mucho más complejo que un MDP, ya que al considerar el estado de creencia (la distribución de probabilidad sobre los estados), el espacio de estados de creencia se vuelve infinito, y la solución exacta mucho más compleja. El espacio de estados de creencia (*belief state*) considera la distribución de probabilidad de los estados dadas las observaciones, $Bel(S)$. Cada asignación de probabilidades a los estados originales es un estado de creencia, por lo que el número es infinito.

Se puede resolver un POMDP en forma exacta considerando un horizonte finito y un espacio de estados originales *pequeño*. Para problemas más complejos o de horizonte infinito, se han propuesto diferentes esquemas de solución aproximada [20,39,38]:

- Asumir que el agente se encuentra en el estado más probable se transforma en un MDP que se puede resolver por cualquiera de los métodos de solución para MDPs.
- Proyectar el estado de creencia a un espacio de menor dimensión, considerando el estado más probable con cierta medida de incertidumbre.
- Utilizar métodos de simulación Montecarlo, como los filtros de partículas.
- Considerar un número finito de pasos y modelar el problema como una red de decisión dinámica, donde la aproximación depende del número de estados que se "ven" hacia delante o *lookahead* (ver sección 5).

En este capítulo nos enfocamos solamente a MDPs, ahora veremos los métodos básicos de solución.

3. Métodos de Solución Básicos

Existen tres formas principales de resolver MDPs: (i) usando métodos de programación dinámica, (ii) usando métodos de Monte Carlo, y (iii) usando métodos de diferencias temporales o de aprendizaje por refuerzo.

3.1. Programación Dinámica (DP)

Si se conoce el modelo del ambiente; es decir, las funciones de probabilidad de las transiciones ($\mathcal{P}^a_{ss'}$) y los valores esperados de recompensas ($\mathcal{R}^a_{ss'}$), las ecuaciones de optimalidad de Bellman nos representan un sistema de $|S|$ ecuaciones y $|S|$ incógnitas.

Consideremos primero como calcular la función de valor V^π dada una política arbitraria π.

$$\begin{aligned} V^\pi(s) &= E_\pi\{R_t \mid s_t = s\} \\ &= E_\pi\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s_t = s\right\} \\ &= E_\pi\left\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\right\} \\ &= \textstyle\sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')] \end{aligned}$$

donde $\pi(s,a)$ es la probabilidad de tomar la acción a en estado s bajo la política π.

Podemos hacer aproximaciones sucesivas, evaluando $V_{k+1}(s)$ en términos de $V_k(s)$:

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')]$$

Podemos entonces definir un algoritmo de evaluación iterativa de políticas como se muestra en el Algoritmo 1, en donde, para una política dada, se evalúan los valores V hasta que no cambian dentro de una cierta tolerancia θ.

Una de las razones para calcular la función de valor de una política es para tratar de encontrar mejores políticas. Dada una función de valor para una política

Algoritmo 1 Algoritmo iterativo de evaluación de política.

Inicializa $V(s) = 0$ para toda $s \in S$
repeat
  $\Delta \leftarrow 0$
  for all $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$ (número positivo pequeño)
regresa $V \approx V^\pi$

dada, podemos probar una acción $a \neq \pi(s)$ y ver si su $V(s)$ es mejor o peor que el $V^\pi(s)$.

En lugar de hacer un cambio en un estado y ver el resultado, se pueden considerar cambios en todos los estados considerando todas las acciones de cada estado, seleccionando aquellas que parezcan mejor de acuerdo a una política *greedy* (seleccionado siempre el mejor). Podemos entonces calcular una nueva política $\pi'(s) = \text{argmax}_a Q^\pi(s,a)$ y continuar hasta que no mejoremos. Esto sugiere, partir de una política (π_0) y calcular la función de valor (V^{π_0}), con la cual encontrar una mejor política (π_1) y así sucesivamente hasta converger a π^* y V^*. A este procedimiento se le llama iteración de políticas y viene descrito en el Algorithmo 2 [19].

Uno de los problemas con el algoritmo de iteración de políticas es que cada iteración involucra evaluación de políticas que requiere recorrer todos los estados varias veces. Sin embargo, el paso de evaluación de política lo podemos truncar de varias formas, sin perder la garantía de convergencia. Una de ellas es pararla después de recorrer una sola vez todos los estados. A esta forma se le llama iteración de valor (*value iteration*). En particular, se puede escribir combinando la mejora en la política y la evaluación de la política truncada como sigue:

$$V_{k+1}(s) = max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')]$$

Esto último se puede ver como expresar la ecuación de Bellman en una regla de actualización. Es muy parecido a la regla de evaluación de políticas, solo que se evalúa el máximo sobre todas las acciones (ver Algoritmo 3).

Para espacios muy grandes, el ver todos los estados puede ser computacionalmente muy caro. Una opción es hacer estas actualizaciones al momento de estar explorando el espacio, y por lo tanto determinando sobre qué estados se hacen las actualizaciones. El hacer estimaciones en base a otras estimaciones se conoce también como *bootstrapping*.

Algoritmo 2 Algoritmo de iteración de política.

1. Inicialización:
$V(s) \in \mathcal{R}$ y $\pi(s) \in \mathcal{A}(s)$ arbitrariamente $\forall s \in S$
2. Evaluación de política:
repeat
  $\Delta \leftarrow 0$
  for all $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)}[\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s')]$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$ (número positivo pequeño)
3. Mejora de política:
pol-estable $\leftarrow$ true
for all $s \in S$: **do**
  $b \leftarrow \pi(s)$
  $\pi(s) \leftarrow \text{argmax}_a \sum_{s'} \mathcal{P}_{ss'}^{a}[\mathcal{R}_{ss'}^{a} + \gamma V(s')]$
  if $b \neq \pi(s)$ **then**
    pol-estable $\leftarrow$ false
  end if
end for
if *pol-estable* **then**
  stop
else
  go to 2
end if

Algoritmo 3 Algoritmo de iteración de valor.

Inicializa $V(s) = 0$ para toda $s \in S$
repeat
  $\Delta \leftarrow 0$
  for all $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow max_a \sum_{s'} \mathcal{P}_{ss'}^{a}[\mathcal{R}_{ss'}^{a} + \gamma V^*(s')]$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$ (número positivo pequeño)
Regresa una política determinísta
Tal que: $\pi(s) = argmax_a \sum_{s'} \mathcal{P}_{ss'}^{a}[\mathcal{R}_{ss'}^{a} + \gamma V^*(s')]$

3.2. Monte Carlo (MC)

Los métodos de Monte Carlo, solo requieren de experiencia y la actualización se hace por episodio en lugar de por cada paso. La función de valor de un estado es la recompensa esperada que se puede obtener a partir de ese estado. Para estimar V^π y Q^π podemos tomar estadísticas haciendo un promedio de las recompensas obtenidas. El procedimiento para V^π está descrito en el Algoritmo 4, donde se guardan los promedios de las recompenzas totales obtenidas en cada estado que se visita en la simulación.

Algoritmo 4 Algoritmo de Monte Carlo para estimar V^π.

loop
  Genera un episodio usando π
  for all estado s en ese episodio: **do**
    $R \leftarrow$ recompensa después de la primera ocurrencia de s
    Añade R a $recomp(s)$
    $V(s) \leftarrow$ promedio($recomp(s)$)
  end for
end loop

Para estimar pares estado-acción (Q^π) corremos el peligro de no ver todos los pares, por lo que se busca mantener la exploración. Lo que normalmente se hace es considerar solo políticas estocásticas; es decir, que tienen una probabilidad diferente de cero de seleccionar todas las acciones.

Con Monte Carlo podemos alternar entre evaluación y mejoras en base a cada episodio. La idea es que después de cada episodio las recompensas observadas se usan para evaluar la política y la política se mejora para todos los estados visitados en el episodio. El algoritmo viene descrito en el Algoritmo 5.

Algoritmo 5 Algoritmo de Monte Carlo para mejorar políticas.

loop
  Genera un episodio usando π con exploración
  for all par s, a en ese episodio: **do**
    $R \leftarrow$ recompensa después de la primera ocurrencia de s, a
    Añade R a $recomp(s, a)$
    $Q(s, a) \leftarrow$ promedio($recomp(s, a)$)
  end for
  for all s en el episodio: **do**
    $\pi(s) \leftarrow \text{argmax}_a Q(s, a)$
  end for
end loop

Para seleccionar una acción durante el proceso de aprendizaje se puede hacer de acuerdo a lo que diga la política o usando un esquema de selección diferente. Dentro de estos últimos, existen diferentes formas para seleccionar acciones, dos de las más comunes son:

- $\epsilon-$*greedy*: en donde la mayor parte del tiempo se selecciona la acción que da el mayor valor estimado, pero con probabilidad ϵ se selecciona una acción aleatoriamente.
- *softmax*, en donde la probabilidad de selección de cada acción depende de su valor estimado. La más común sigue una distribución de Boltzmann o de Gibbs, y selecciona una acción con la siguiente probabilidad:

$$\frac{e^{Q_t(s,a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(s,b)/\tau}}$$

  donde τ es un parámetro positivo (temperatura) que se disminuye con el tiempo.

Los algoritmos de aprendizaje los podemos dividir en base a la forma que siguen para seleccionar sus acciones durante el proceso de aprendizaje:

- Algoritmos *on-policy*: Estiman el valor de la política mientras la usan para el control. Se trata de mejorar la política que se usa para tomar decisiones.
- Algoritmos *off-policy*: Usan la política y el control en forma separada. La estimación de la política puede ser por ejemplo *greedy* y la política de comportamiento puede ser ϵ-*greedy*. Osea que la política de comportamiento está separada de la política que se quiere mejorar. Esto es lo que hace Q-learning, el cual describiremos en la siguiente sección.

3.3. Aprendizaje por Refuerzo (RL)

Los métodos de Aprendizaje por Refuerzo (*Reinforcement Learning*) o diferencias temporales, combinan las ventajas de los dos enfoques anteriores: permiten hacer *bootstrapping* (como DP) y no requiere tener un modelo del ambiente (como MC).

Métodos tipo RL sólo tienen que esperar el siguiente paso para hacer una actualización en la función de valor. Básicamente usan el error o diferencia entre predicciones sucesivas (en lugar del error entre la predicción y la salida final) para aprender. Su principal ventaja es que son incrementales y por lo tanto fáciles de computar.

El algoritmo de diferencias temporales más simple es TD(0), el cual viene descrito en el Algoritmo 6 y cuya función de actualización es:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$

La actualización de valores tomando en cuenta la acción sería:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Algoritmo 6 Algoritmo TD(0).

Inicializa $V(s)$ arbitrariamente y π a la política a evaluar
for all episodio **do**
  Inicializa s
  for all paso del episodio (hasta que s sea terminal) **do**
    $a \leftarrow$ acción dada por π para s
    Realiza acción a; observa recompensa, r, y siguiente estado, s'
    $V(s) \leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right]$
    $s \leftarrow s'$
  end for
end for

y el algoritmo es prácticamente el mismo, solo que se llama SARSA (State - Action - Reward - State' - Action'), y viene descrito en el Algoritmo 7.

Algoritmo 7 Algoritmo SARSA.

Inicializa $Q(s, a)$ arbitrariamente
for all episodio **do**
  Inicializa s
  Selecciona una a a partir de s usando la política dada por Q (e.g., ϵ–greedy)
  for all paso del episodio (hasta que s sea terminal) **do**
    Realiza acción a, observa r, s'
    Escoge a' de s' usando la política derivada de Q
    $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  end for
end for

Uno de los desarrollos más importantes en aprendizaje por refuerzo fué el desarrollo de un algoritmo "fuera-de-política" (*off-policy*) conocido como Q-learning. La idea principal es realizar la actualización de la siguiente forma (Watkins, 89):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

El algoritmo viene descrito en el Algoritmo 8.

4. Técnicas de Soluciones Avanzadas

Se han propuesto diferentes algoritmos de solución que combinan o proponen alguna solución intermedia entre algunas de las técnicas vistas en la sección 3. En particular entre Monte Carlo y Aprendizaje por Refuerzo (trazas de elegibilidad) y entre Programación Dinámica y Aprendizaje por Refuerzo (Dyna-Q y *Prioritized Sweeping*). En las siguientes secciones veremos estas técnicas.

Algoritmo 8 Algoritmo Q-Learning.

```
Inicializa Q(s, a) arbitrariamente
for all episodio do
  Inicializa s
  for all paso del episodio (hasta que s sea terminal) do
    Selecciona una a de s usando la política dada por Q (e.g., ϵ–greedy)
    Realiza acción a, observa r, s'
    Q(s, a) ← Q(s, a) + α [r + γ max'_a Q(s', a') − Q(s, a)]
    s ← s';
  end for
end for
```

4.1. Trazas de Elegibilidad (*eligibility traces*)

Esta técnica se puede considerar que está entre los métodos de Monte Carlo y RL de un paso. Los métodos Monte Carlo realizan la actualización considerando la secuencia completa de recompensas observadas, mientras que la actualización de los métodos de RL se hace utilizando únicamente la siguiente recompensa. La idea de las trazas de elegibilidad es considerar las recompensas de n estados posteriores (o afectar a n anteriores).

Si recordamos la recompenza total acumulada es:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-t-1} r_T$$

Lo que se hace en RL es usar:

$$R_t = r_{t+1} + \gamma V_t(s_{t+1})$$

lo cual hace sentido porque $V_t(s_{t+1})$ reemplaza a los términos siguientes ($\gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$). Sin embargo, hace igual sentido hacer:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})$$

y, en general, para n pasos en el futuro.

En la práctica, más que esperar n pasos para actualizar (*forward view*), se realiza al revés (*backward view*). Se guarda información sobre los estados por los que se pasó y se actualizan hacia atrás las recompensas (descontadas por la distancia). Se puede probar que ambos enfoques son equivalentes.

Para implementar la idea anterior, se asocia a cada estado o par estado-acción una variable extra, representando su traza de elegibilidad (*eligibility trace*) que denotaremos por $e_t(s)$ o $e_t(s, a)$. Este valor va decayendo con la longitud de la traza creada en cada episodio. Para $TD(\lambda)$:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{si } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{si } s = s_t \end{cases}$$

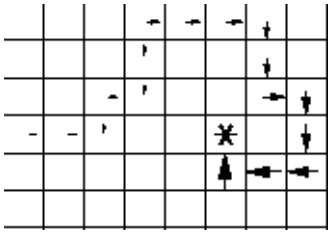


Figura 2. Comportamiento de las recompensas dadas a un cierto camino dadas por las trazas de elegibilidad.

Para SARSA se tiene lo siguiente:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{si } s \neq s_t \\ \gamma\lambda e_{t-1}(s,a) + 1 & \text{si } s = s_t \end{cases}$$

$SARSA(\lambda)$ viene descrito en el Algoritmo 9, donde todos los estados visitados reciben parte de las recompensas futuras dependiendo de su distancia, como se ilustra en la figura 2.

Algoritmo 9 $SARSA(\lambda)$ con trazas de elegibilidad.

```
Inicializa Q(s,a) arbitrariamente y e(s,a) = 0 ∀s,a
for all episodio (hasta que s sea terminal) do
  Inicializa s,a
  for all paso en el episodeo do
    Toma acción a y observa r,s'
    Selecciona a' de s' usando una política derivada de Q (e.g., ε−greedy)
    δ ← r + γQ(s',a') − Q(s,a)
    e(s,a) ← e(s,a) + 1
    for all (s,a) do
      Q(s,a) ← Q(s,a) + αδe(s,a)
      e(s,a) ← γλe(s,a)
      s ← s'; a ← a'
    end for
  end for
end for
```

Para Q-learning como la selección de acciones se hace, por ejemplo, siguiendo una política $\epsilon-greedy$, se tiene que tener cuidado, ya que a veces los movimientos, son movimientos exploratorios de los cuales no necesariamente queremos propagar sus recompensas hacia atrás. Aquí se puede mantener historia de la traza solo hasta el primer movimiento exploratorio, ignorar las acciones exploratorias, o hacer un esquema un poco más complicado que considera todas las posibles acciones en cada estado.

4.2. Planificación y Aprendizaje

Asumamos que tenemos un modelo del ambiente, esto es, que podemos predecir el siguiente estado y la recompensa dado un estado y una acción. La predicción puede ser un conjunto de posibles estados con su probabilidad asociada o puede ser un estado que es muestreado de acuerdo a la distribución de probabilidad de los estados resultantes.

Dado un modelo, es posible hacer planificación. Lo interesante es que podemos utilizar los estados y acciones utilizados en la planificación también para aprender. De hecho al sistema de aprendizaje no le importa si los pares estado-acción son dados de experiencias reales o simuladas.

Dado un modelo del ambiente, uno podría seleccionar aleatoriamente un par estado-acción, usar el modelo para predecir el siguiente estado, obtener una recompensa y actualizar valores Q. Esto se puede repetir indefinidamente hasta converger a Q^*.

El algoritmo Dyna-Q combina experiencias con planificación para aprender más rápidamente una política óptima. La idea es aprender de experiencia, pero también usar un modelo para simular experiencia adicional y así aprender más rápidamente (ver Algoritmo 10). La experiencia acumulada también se usa para actualizar el modelo.

Algoritmo 10 Algoritmo de Dyna-Q.

Inicializa $Q(s,a)$ y $Modelo(s,a)$ $\forall s \in S, a \in A$
loop
  $s \leftarrow$ estado actual
  $a \leftarrow \epsilon-greedy(s,a)$
  realiza acción a observa s' y r
  $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
  $Modelo(s,a) \leftarrow s', r$
  for N veces **do**
    $s \leftarrow$ estado anterior seleccionado aleatoriamente
    $a \leftarrow$ acción aleatoria tomada en s
    $s', r \leftarrow Modelo(s,a)$
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
  end for
end loop

El algoritmo de Dyna-Q selecciona pares estado-acción aleatoriamente de pares anteriores. Sin embargo, la planificación se puede usar mucho mejor si se enfoca a pares estado-acción específicos. Por ejemplo, enfocarnos en las metas e irnos hacia atrás o más generalmente, irnos hacia atrás de cualquer estado que cambie su valor. Esto es, enfocar la simulación al estado que cambio su valor. Esto nos lleva a todos los estados que llegan a ese estado y que también cambiarían su valor. Esto proceso se puede repetir sucesivamente, sin embargo, algunos estados

cambian mucho más que otros. Lo que podemos hacer es ordenarlos y realizar las simulaciones con el modelo solo en los estados que rebacen un cierto umbral. Esto es precisamente lo que hacer el algoritmo de *prioritized sweeping* descrito en el Algoritmo 11.

Algoritmo 11 Algoritmo de Prioritized sweeping.

Inicializa $Q(s,a)$ y $Modelo(s,a)$ $\forall s \in S, a \in A$ y $ColaP = \emptyset$
loop
  $s \leftarrow$ estado actual
  $a \leftarrow \epsilon-greedy(s,a)$
  realiza acción a observa s' y r
  $Modelo(s,a) \leftarrow s', r$
  $p \leftarrow\mid r + \gamma \max_{a'} Q(s',a') - Q(s,a) \mid$
  if $p > \theta$ **then**
    inserta s, a a $ColaP$ con prioridad p
  end if
  for N veces, mientras $ColaP \neq \emptyset$ **do**
    $s, a \leftarrow$ primero($ColaP$)
    $s', r \leftarrow Modelo(s,a)$
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    for all $\overline{s}, \overline{a}$ que se predice llegan a s **do**
      $\overline{r} \leftarrow$ recompensa predicha
      $p \leftarrow\mid \overline{r} + \gamma \max_a Q(s,a) - Q(\overline{s}, \overline{a}) \mid$
      if $p > \theta$ **then**
        inserta $\overline{s}, \overline{a}$ a $ColaP$ con prioridad p
      end if
    end for
  end for
end loop

4.3. Generalización en Aprendizaje por Refuerzo

Hasta ahora hemos asumido que se tiene una representación explícita de las funciones de valor en forma de tabla (i.e., una salida por cada tupla de entradas). Esto funciona para espacios pequeños, pero es impensable para dominios como ajedrez (10^{120} estados) o backgammon (10^{50} estados). Una forma para poder aprender una función de valor en espacios grandes, es hacerlo con una representación implícita, i.e., una función. Por ejemplo en juegos, una función de utilidad estimada de un estado se puede representar como una función lineal pesada sobre un conjunto de atributos (f_i's):

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

En ajedrez se tienen aproximadamente 10 pesos, por lo que es una compresión bastante significativa. La compresión lograda por una representación implícita permite al sistema de aprendizaje, generalizar de estados visitados a estados no visitados. Por otro lado, puede que no exista tal función. Como en todos los sistemas de aprendizaje, existe un balance entre el espacio de hipótesis y el tiempo que se toma aprender una hipótesis aceptable. Muchos sistemas de aprendizaje supervisado tratan de minimizar el error cuadratico medio (MSE) bajo cierta distribución $P(s)$ de las entradas. Supongamos que $\boldsymbol{\Theta}_t$ representa el vector de parámetros de la función parametrizada que queremos aprender, el error cuadrado se calcula como la diferencia entre el valor de la política real y el que nos da nuestra función:

$$MSE(\boldsymbol{\Theta}_t) = \sum_{s \in S} P(s)[V^{\pi(s)} - V_t(s)]^2$$

donde $P(s)$ es una distribución pesando los errores de diferentes estados. Cuando los estados son igualmente probables, podemos eliminar este término. Para ajustar los parámetros del vector de la función que queremos optimizar, se obtiene el gradiente y se ajustan los valores de los parámetros en la dirección que produce la máxima reducción en el error, esto es, ajustamos los parámetros de la función restandoles la derivada del error de la función con respecto a estos parámetros:

$$\begin{aligned} \boldsymbol{\Theta}_{t+1} &= \boldsymbol{\Theta}_t - \tfrac{1}{2}\alpha \nabla_{\boldsymbol{\Theta_t}} [V^{\pi(s_t)} - V_t(s_t)]^2 \\ &= \boldsymbol{\Theta}_t + \alpha [V^{\pi(s_t)} - V_t(s_t)] \nabla_{\boldsymbol{\Theta_t}} V_t(s_t) \end{aligned}$$

donde α es un parámetro positivo $0 \leq \alpha \leq 1$ que determina que tanto cambiamos los parámetros en función del error, y $\nabla_{\boldsymbol{\Theta_t}} f(\Theta_t)$ denota un vector de derivadas parciales de la función con respecto a los parámetros. Si representamos la función con n parámetros tendríamos:

$$\nabla_{\boldsymbol{\Theta_t}} V_t(s_t) = \left(\frac{\partial f(\boldsymbol{\Theta_t})}{\partial \Theta_t(1)}, \frac{\partial f(\boldsymbol{\Theta_t})}{\partial \Theta_t(2)}, \ldots, \frac{\partial f(\boldsymbol{\Theta_t})}{\partial \Theta_t(n)} \right)^T$$

Como no sabemos $V^{\pi(s_t)}$ la tenemos que aproximar. Una forma de hacerlo es con trazas de elegibilidad y actualizar la función Θ usando información de un función de valor aproximada usando la recompensa obtenida y la función de valor en el siguiente estado, como sigue:

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t + \alpha \delta_t \boldsymbol{e}_t$$

donde δ_t es el error:

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

y $\boldsymbol{e}_t$ es un vector de trazas de elegibilidad, una por cada componente de $\boldsymbol{\Theta}_t$, que se actualiza como:

$$\boldsymbol{e}_t = \gamma \lambda \boldsymbol{e}_{t-1} + \nabla_{\boldsymbol{\Theta}_t} V_t(s_t)$$

con $\boldsymbol{e}_0 = 0$. Como estamos usando trazas de elegibilidad, la aproximación de la función de valor se hace usando información de todo el camino recorrido y no sólo de la función de valor inmediata.

5. Representaciones Factorizadas y Abstractas

Uno de los principales problemas de los MDPs es que el espacio de estados y acciones puede crecer demasiado (miles o millones de estados, decenas de acciones). Aunque lo métodos de solución, como iteración de valor o política, tienen una complejidad polinomial, problemas tan grandes se vuelven intratables tanto en espacio como en tiempo. Para ello se han propuesto diversos esquemas para reducir o simplificar el modelo, en algunos casos sacrificando optimalidad. Estos métodos los podemos dividir en 3 tipos principales:

1. Factorización. El estado se descompone en un conjunto de variables, reduciendo el espacio para representar las funciones de transición y de recompensa.
2. Abstracción. Se agrupan estados con propiedades similares, o se utilizan representaciones relacionales basadas en lógica de predicados; reduciendo en ambos casos el espacio de estados.
3. Descomposición. El problema se divide en varios sub–problemas, que puedan ser resueltos independientemente, para luego conjuntar las soluciones.

En esta sección veremos las dos primeras estrategias, y en la siguiente hablaremos sobre decomposición. Antes introducimos las redes bayesianas, en las que se basan los MDPs factorizados.

5.1. Redes Bayesianas

Las redes bayesianas (RB) modelan un fenómeno mediante un conjunto de variables y las relaciones de dependencia entre ellas. Dado este modelo, se puede hacer inferencia bayesiana; es decir, estimar la probabilidad posterior de las variables no conocidas, en base a las variables conocidas. Estos modelos pueden tener diversas aplicaciones, para clasificación, predicción, diagnóstico, etc. Además, pueden dar información interesante en cuanto a cómo se relacionan las variables del dominio, las cuales pueden ser interpretadas en ocasiones como relaciones de causa–efecto. En particular, nos interesan las RB ya que permiten representar de un forma más compacta las funciones de transición de los MDPs.

Representación Las redes bayesianas [28] son una representación gráfica de dependencias para razonamiento probabilístico, en la cual los nodos representan variables aleatorias y los arcos representan relaciones de dependencia directa entre las variables. La Figura 3 muestra un ejemplo hipotético de una red

bayesiana (RB) que representa cierto conocimiento sobre medicina. En este caso, los nodos representan enfermedades, síntomas y factores que causan algunas enfermedades. La variable a la que apunta un arco es dependiente de la que está en el origen de éste, por ejemplo *Fiebre* depende de *Tifoidea* y *Gripe* en la red de la Figura 3. La topología o estructura de la red nos da información sobre las dependencias probabilísticas entre las variables. La red también representa las independencias condicionales de una variable (o conjunto de variables) dada(s) otra(s) variable(s). Por ejemplo, en la red de la Figura 3, *Reacciones* (R) es condicionalmente independiente de *Comida* (C) dado *Tifoidea* (T):

$$P(R|C,T) = P(R|T) \tag{1}$$

Esto se representa gráficamente por el nodo T *separando* al nodo C de R.

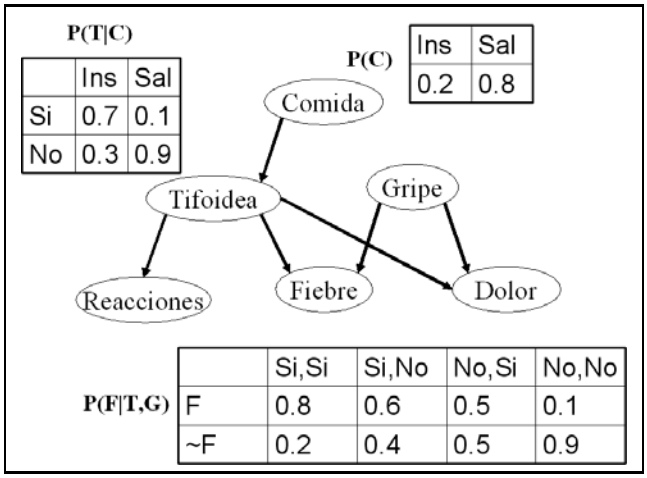


Figura 3. Ejemplo de una red bayesiana. Los nodos representan variables aleatorias y los arcos relaciones de dependencia. Se muestran las tablas de probabilidad condicional de algunas de las variables de la red bayesiana: probabilidad *a priori* de Comida, $P(C)$; probabilidad de Tifoidea dada Comida, $P(T \mid C)$; y probabilidad de Fiebre dada Tifoidea y Gripe, $P(F \mid T, G)$. En este ejemplo se asume que todas las variables son binarias.

Complementa la definición de una red bayesiana las probabilidades condicionales de cada variable dados sus padres:

- Nodos raíz: vector de probabilidades marginales.
- Otros nodos: matriz de probabilidades condicionales dados sus padres.

Aplicando la regla de la cadena y las independencias condicionales, se puede verificar que con dichas probabilidades se puede calcular la probabilidad conjunta

de todas las variables; como el producto de las probabilidades de cada variable (X_i) dados sus padres ($Pa(X_i)$):

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i|Pa(X_i)) \tag{2}$$

La Figura 3 ilustra algunas de las matrices de probabilidad asociadas al ejemplo de red bayesiana.

Redes bayesianas dinámicas Las redes bayesianas, en principio, representan el estado de las variables en un cierto momento en el tiempo. Para representar procesos dinámicos existe una extensión a estos modelos conocida como *red bayesiana dinámica* (RBD) [26]. Una RBD consiste en una representación de los estados del proceso en cierto tiempo (red estática) y las relaciones temporales entre dichos procesos (red de transición). Para las RBDs generalmente se hacen las siguientes suposiciones:

- Proceso markoviano. El estado actual sólo depende del estado anterior (sólo hay arcos entre tiempos consecutivos).
- Proceso estacionario en el tiempo. Las probabilidades condicionales en el modelo no cambian con el tiempo.

Lo anterior implica que podemos definir una red bayesiana dinámica en base a dos componentes: (i) una red base estática que se repite en cada periodo, de acuerdo a cierto intervalo de tiempo predefinido; y (ii) una red de transición entre etapas consecutivas (dada la propiedad markoviana). Un ejemplo de una RBD se muestra en la Figura 4.

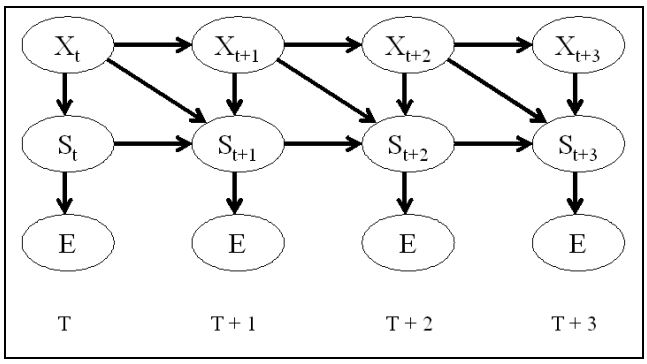


Figura 4. Ejemplo de una red bayesiana dinámica. Se muestra la estructura base que se repite en cuatro etapas temporales, así como las relaciones de dependencia entre etapas.

Estos modelos, en particular las redes bayesianas dinámicas, permiten representar en una forma mucho más compacta las funciones de transición en MDPs, como veremos en la siguiente sección.

5.2. MDPs Factorizados

Al aumentar el número de estados en un MDP, crecen las tablas para representar las funciones de transición, las cuales se pueden volver inmanejables. Por ejemplo, si tenemos un MDP con 1000 estados (algo común en muchas aplicaciones), la tabla de transición, por acción, tendría un millón (1000×1000) de parámetros (probabilidades). Una forma de reducir esta complejidad es mediante el uso de redes bayesianas dinámicas, en lo que se conoce como *MDPs factorizados* [4].

En los MDPs factorizados, el estado de descompone en un conjunto de variables o factores. Por ejemplo, en vez de representar el estado de un robot *mensajero* mediante una sóla variable, S, con un gran número de valores; podemos descomponerla en n variables, $x_1, x_2, \ldots, x_n$, donde cada una representa un aspecto del estado (posición del robot, si tiene un mensaje que entregar, posición de la meta, etc.). Entonces el modelo de transición se representa usando RBDs. Se tiene una RBD con dos etapas temporales por acción, que especifican las variables en el tiempo t y en el tiempo $t+1$. La estructura de la RBD establece las dependencias entre las variables. Se tiene asociada una tabla de probabilidad por variable en $t+1$, que establece la probabilidad condicional de la variable $x_i(t+1)$ dados sus padres, $x_{1..k}(t)$, asumiendo que tiene k padres. Un ejemplo de un modelo factorizado se muestra en la figura 5. La representación factorizada de las funciones de transición implica, generalmente, un importante ahorro de espacio, ya que en la práctica, tanto las funciones de transición como las de recompensa tienden a depender de pocas variables, resultando en modelos *sencillos*.

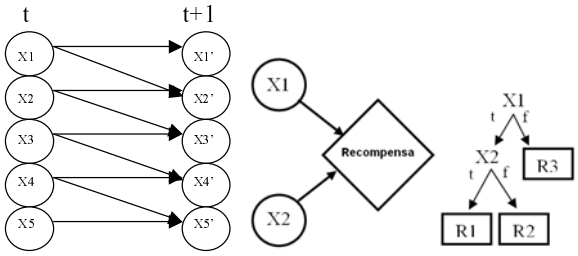


Figura 5. MDP factorizado. Izquierda: función de transición representada mediante una RBD, para un MDP con 5 variables de estado. Centro: la función de recompensa sólo depende de dos de las variables de estado. Derecha: función de recompensa como un árbol de decisión binario.

Al descomponer el estado en un conjunto de variables, también es posible representar la función de recompensa en forma factorizada. La función de recompensa se representa usando árboles de decisión, donde cada nodo en el árbol corresponde a una de las variables de estado, y las hojas a los diferentes valores de recompensa. La figura 5 ilustra un ejemplo de una función de recompensa factorizada.

En base a estas representaciones factorizadas se han desarrollado extensiones de los algoritmos básicos de solución, que son mucho más eficientes al operar directamente sobre las respresntaciones estructuradas. Dos ejemplo son el método de *iteración de política estructurado* [5] y *SPUDD* [18]. SPUDD además de usar una representación factorizada, simplifica aún más el modelo al representar las tablas de probabilidad condicional mediante grafos llamados *ADDs*.

5.3. Agregación de estados

Otra alternativa para reducir la complejidad es reducir el número de estados, agrupando estados con propiedades similares (por ejemplo, que tengan la misma función de valor o política). Si dos o más estados tienen la misma recompensa y probabilidades de transición a otros estados, se dice que son *estados equivalentes*, y se pueden fusionar en un sólo estado sin pérdida de información. Al agregar estados de esta forma, se mantiene una solución óptima del MDP. Sin embargo, en la práctica, generalmente la reducción de estados es relativamente pequeña, por lo que se han planetado otras alternativas. Éstas agrupan estados aunque no sean equivalentes, pero si *parecidos*, con lo que se sacrifica optimalidad pero se gana en eficiencia, con una solución aproximada.

MDPs cualitativos Se han propuesto varios esquemas de agregación de estados [4], veremos uno de ellos, basado en *estados cualitativos* (Q) [32]. Además de permitir reducir el número de estados en MDPs discretos, este método permite resolver MDPs continuos. El método se basa en una representación cualitativa de los estados, donde un estado cualitativo, q_i, es un grupo de estados (o una partición del espacio en el caso de MDPs continuos) que tienen la misma recompensa inmediata.

La solución de un MDP cualitativo se realiza en dos fases [32]:

Abstracción: El espacio de estados se divide en un conjunto de estados cualitativos, $q_1, \ldots, q_n$, donde cada uno tiene la misma recompensa. En el caso de un MDP discreto simplemente se agrupan todos los estados vecinos que tengan la misma recompensa. En el caso de un MDP continuo, esta partición se puede aprender mediante una exploración del ambiente, particionando el espacio de estados en secciones (hiper–rectángulos) de igual recompensa. El espacio de estados cualitativos se representa mediante un árbol de decisión, denominado *q-tree*. La figura 6 muestra un ejemplo sencillo en dos dimensiones de un espacio de estados cualitativos y el *q-tree* correspondiente. Este MDP cualitativo se resuelve usando métodos clásicos como iteración de valor.

Refinamiento: La agrupación incial puede ser demasiado *abstracta*, obteniendo una solución sub–óptima. Para ello se plantea una segunda fase de refinamiento, en la cual algunos estados se dividen, basado en la varianza en utilidad respecto a los estados vecinos. Para ello se selecciona el estado cualitativo con mayor varianza en valor (diferencia del valor con los estados vecinos), y se parte en dos, resolviendo el nuevo MDP. Si la política resultante difiere de la anterior, se mantiene la partición, sino se descarta dicha partición y se marca ese estado. Este procedimiento se repite recursivamente hasta que ya no haya estado sin marcar o se llegue a un tamaño mínimo de estado (de acuerdo a la aplicación). La figura 7 ilustra el proceso de refinamiento para el ejemplo anterior.

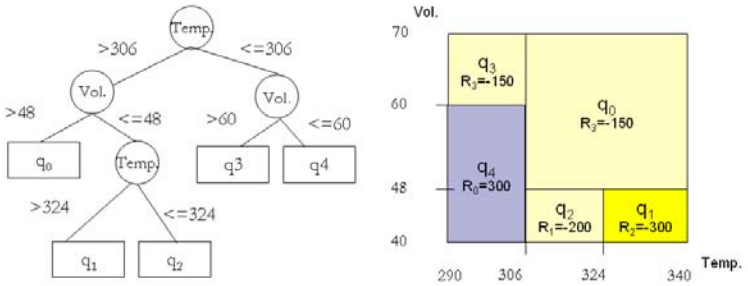


Figura 6. Ejemplo de una partición cualitativa en dos dimensiones, donde las variables de estado son Volumen y Temperatura. Izquierda: $q-tree$, las ramas son restricciones y las hojas los estados cualitativos. Derecha: partición del espacio bidimensional en 5 estados.

Aunque este esquema no garantiza una solución óptima, se ha encontrado experimentalmente que se obtienen soluciones aproximadamente óptimas con ahorros significativos en espacio y tiempo. En la sección 7 se ilustra este método en una aplicación para el control de plantas eléctricas.

5.4. Aprendizaje por Refuerzo Relacional

A pesar del trabajo que se ha hecho en aproximaciones de funciones (e.g., [7]), abstracciones jerárquicas (e.g., [11]) y temporales (e.g., [36]) y representaciones factorizadas(e.g., [21]), se sigue investigando en cómo tratar de forma efectiva espacios grandes de estados, cómo incorporar conocimiento del dominio y cómo transferir políticas aprendidas a nuevos problemas similares.

La mayoría de los trabajos utilizan una representación proposicional que sigue sin escalar adecuadamente en dominios que pueden ser definidos de manera más

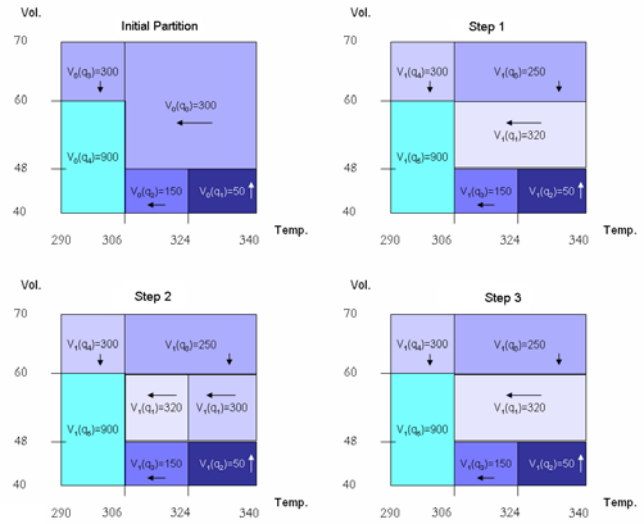


Figura 7. Varias etapas en el proceso de refinamiento para el ejemplo del MDP cualitativo en dos dimensiones. A partir de la partición inicial, se divide un estado cualitativo en dos (Step 1). Después se divide uno de los nuevos estados (Step 2). Al resolver el nuevo MDP se encuentra que la política no varía, por lo que se vuelven a unir en un estado (Step 3).

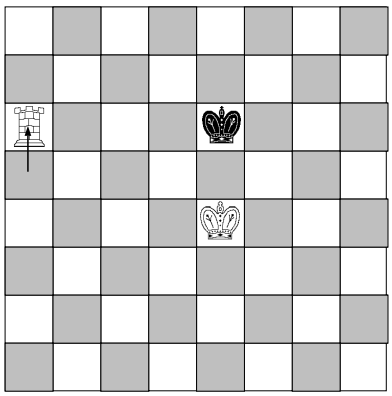


Figura 8. Obligando al rey contrario a moverse hacia una orilla.

natural en términos de objetos y relaciones. Para ilustrar esto, supongamos que queremos aprender una política para jugar un final de juego sencillo en ajedrez. Inclusive para un final sencillo como Rey-Torre vs. Rey (RTR), existen más de 175,000 posiciones legales donde no existe un jaque. El número de posibles acciones por estado en general es de 22 (8 para el rey y 14 para la torre), lo cual nos da cerca de 4 millones de posibles pares estado-acción. Aprender directamente en esta representación esta fuera del alcance de computadoras estándar, sin embargo, existen muchos estados que representan escencialmente lo mismo, en el sentido que comparten las mismas relaciones. Para un jugador de ajedrez la posición exacta de las piezas no es tan importante como las relaciones existentes entre las piezas para decidir que mover (ver [8,10]). Por ejemplo, en RTR cuando los dos reyes estan en oposición (misma fila o renglón con un cuadro entre ellos), la torre divide a los reyes y la torre está alejada al menos dos cuadros del rey contrario (i.e., puede hacer jaque en forma segura), una buena jugada es hacer jaque y forzar al rey a moverse hacia una orilla (ver Figura 8). Esta acción es aplicable a todas las posiciones del juego que cumplen con estas relaciones, lo cual captura más de 1,000 posiciones para una torre blanca. De hecho se puede aplicar para tableros de diferente tamaño.

La idea de usar una representación relacional es usar un conjunto reducido de estados abstractos que representan un conjunto de relaciones (e.g., en_oposición, torre_divide, etc.), un conjunto de acciones en términos de esas relaciones (e.g., If en_oposición And torre_divide Then jaque) y aprender qué acción usar para cada estado abstracto. La ventaja de una representación relacional es que puede contener variables y por lo tanto representar a muchas instancias particulares, lo cual reduce considerablemente el conjunto de estados y acciones, simplificando el proceso de aprendizaje. Por otro lado, es posible transferir las políticas aprendidas en esta representación abstracta a otras instancias del problema general.

En este capítulo seguimos la caracterización originalmente propuesta en [24]. Un MDP relacional, está definido de la misma forma por una tupla $M =< S_R, A_R, \Phi_R, R_R >$. En este caso, S_R es un conjunto de estados, cada uno definido por una conjunción de predicados en lógica de primer orden. $A_R(S_R)$ es un conjunto de acciones cuya representación es parecida a operadores tipo STRIPS. Φ_R es la función de transición que opera sobre estados, acciones y estados resultantes, todos ellos relacionales. Si existen varios posibles estados relaciones por cada acción se tienen que definir transiciones parecidas a operadores STRIPS probabilísticos. R_R es una función de recompensa que se define como en un MDP tradicional, dando un número real a cada estado $R_R : S \rightarrow \mathcal{R}$, solo que ahora se le da la misma recompensa a todas las instancias del estado s_R. A diferencia de un MDP tradicional, el espacio de estados esta implícitamente definido.

Una vez definido un MPD relacional podemos definir también algoritmos para aprender una política óptima. En este capítulo solo describimos cómo aplicar Q-learning en esta representación, sin embargo, lo mismo puede aplicarse a otros algoritmos. El Algoritmo 12 contiene el pseudo–código para el algoritmo rQ-learning. El algoritmo es muy parecido a Q-learning, con la diferencia que los

estados (S) y las acciones (A) están representados en forma relacional. El algoritmo sigue tomando acciones primitivas (a) y moviendose en estados primitivos (s), pero el aprendizaje se realiza sobre los pares estado - acción relacionales. En este esquema particular, de un estado primitivo se obtienen primero las relaciones que definen el estado relacional. En ajedrez, de la posición de las piezas del tablero se obtendrían relaciones entre las piezas para definir el estado relacional.

Algoritmo 12 El algoritmo rQ-learning.

Initialize $Q(S, A)$ arbitrarily
(where S is an *r-state* and A is an *r-action*)
for all episodio **do**
  Inicializa s
  $S \leftarrow rels(s)$ {conjunto de relaciones en estado s}
  for all paso del episodio (hasta que s sea terminal) **do**
    Selecciona A de S usando una política no determinística (e.g., ϵ-greedy)
    Selecciona aleatoriamente una acción a aplicable en A
    Toma acción a, observa r, s'
    $S' \leftarrow rels(s')$
    $Q(S, A) \leftarrow Q(S, A) + \alpha(r + \gamma max_{A'} Q(S', A') - Q(S, A))$
    $S \leftarrow S'$
  end for
end for

Esta representación relacional se ha probado en laberintos, el problema del Taxi, en el mundo de los bloques, en ajedrez para aprender a jugar un final de juego sencillo y para aprender a volar un avión (como se verá en la sección 7.1).

Lo bueno de aprender una política relacional es que se puede usar en diferentes instancias del problema o para problemas parecidos. La política aprendida en el dominio del Taxi (ver figura 13) usando una representación relacional, se uso como política en un espacio más grande, con nuevos destinos, cuyas posiciones son diferentes, e inclusive con paredes horizontales. En este caso, la misma política se puede aplicar y dar resultados óptimos. La figura 9 ilustra dos caminos tomados por el taxi para ir a un punto a recoger a un pasajero y llevarlo a otro punto seleccionado aleatoriamente.

6. Técnicas de Descomposición

Otra clase de técnicas de para reducir la complejidad de MDPs se base en el principio de "divide y vencerás". La idea es tratar de dividir el problema en sub–problemas que puedan ser resueltos en forma independiente, y después de alguna manera combinar las soluciones para obtener una solución global aproximada. Existen básicamente dos enfoques para descomponer un MDP:

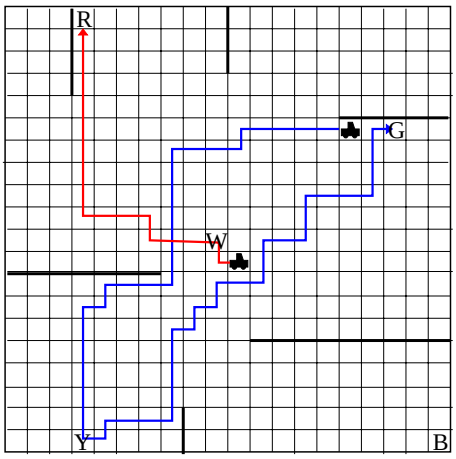


Figura 9. Dos caminos seguidos en dos instancias de un problema de Taxi aumentado.

1. Jerárquico. Se descompone el MDP en una serie de subprocesos, submetas o subrutinas, de forma que para obtener la solución global, se tienen que resolver los subprocesos, típicamente en forma secuencial. Por ejemplo, para que un robot vaya de un cuarto a otro en un edificio, el problema se puede separar en 3 sub–MDPs: salir del cuarto, ir por el pasillo al otro cuarto, y entrar al cuarto.
2. Concurrente. Se divide el MDP en un conjunto de subprocesos donde cada uno contempla un aspecto del problema, de forma que en conjunto lo resuelven en forma concurrente. Por ejemplo, en navegación robótica se podría tener un MDP que guía al robot hacia la meta y otro que evade obstáculos; al conjuntar las soluciones el robot va a la meta evitando obstáculos.

Estas técnicas sacrifican, generalmente, optimalidad para poder resolver problemas mucho más complejos. A continuación analizaremos ambos enfoques.

6.1. Decomposición jerárquica

Debido a que el espacio de solución al resolver un MDP tradicional crece de manera exponencial con el número de variables, se han propuesto esquemas que descomponen el problema original en sub-problemas. Las principales ventajas de esta descomposición son:

- Poder escalar los MDP a problemas más complejos.
- El poder compartir o re-utilizar soluciones de subtareas, en la solución del mismo problema o inclusive de problemas similares, transfiriendo las políticas aprendidas en las subtareas.

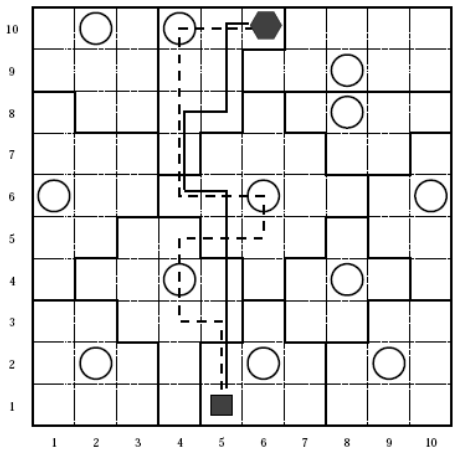


Figura 10. La figura muestra una descomposición de un espacio siguiendo un criterio de Voronoi, puntos característicos de cada región (círculos) y la trayectoria entre dos puntos pasando por puntos característicos.

Por ejemplo, supongamos que se tiene una rejilla de 10 por 10, como se ilustra en la figura 10. En este problema existen 10,000 combinaciones de posiciones del agente y la meta (mostrados con el hexágono y cuadrado obscuros). Este implica que para resolver este problema para cualquier estado inicial y final se requieren 10,000 valores V o aproximadamente 40,000 valores Q, asumiendo que se tienen cuatro movimientos posibles (norte, sur, este y oeste). Una opción es partir en espacio en regiones, seleccionando un conjunto de puntos característicos (los círculos mostrados en la figura) y asignándoles las celdas más cercanas siguiendo un criterio de Voronoi. Con esto se puede imponer una restricción en la política siguiendo el siguiente esquema:

1. Ir del punto de inicio a la posición del punto característico de su región,
2. Ir de un punto característico de una región al punto característico de la región que tiene la meta, siguiendo los puntos característicos de las regiones intermedias,
3. Ir del punto característico de la región con el estado meta a la meta.

Lo que se tiene que aprender es como ir de cada celda a su punto característico, como ir entre puntos caracteríticos, y como ir de un punto característico a cualquier celda de su región. Para el ejemplo anterior, esto requiere 6,070 valores Q (comparados con los 40,000 originales). Esta reducción se debe a que muchas de las subtareas son compartidas por muchas combinaciones de estados inicial y final. Sin embargo, estan restricciones pueden perder la optimalidad.

En general, en las estrategias de descomposición se siguen dos enfoques:

- Una descomposición del espacio de estados y recompensas, pero no de las acciones.
- Descomposición de las acciones, en donde cada agente es responsable de parte de las acciones.

En este capítulo solo vamos a revisar algunos de los trabajos más representativos de MPDs jerárquicos, como *Options*, HAM, y MAX-Q, que caen en el primer enfoque, y descomposición concurrente, que puede incluir el segundo enfoque.

6.2. Options o Macro-Actions

Sutton, Precup, y Singh en [37] extienden el modelo clásico de MDPs e incluyen "options" o macro acciones que son políticas que se siguen durante cierto tiempo. Una macro acción tiene asociada una política, un criterio de terminación y una región en el espacio de estados (la macro-acción puede empezar su ejecución en cualquier estado de la región). Al introducir macro-acciones el modelo se convierte en un proceso de decisión semi-markoviano o SMDP, por sus siglas en inglés.

La figura 11 muestra un ejemplo. La región de iniciación de la macro-acción es la parte izquierda de la figura, la política es la mostrada en la figura y la condición de terminación es terminar con probabilidadad de 1 en cualquier estado fuera del cuarto y 0 dentro del cuarto.

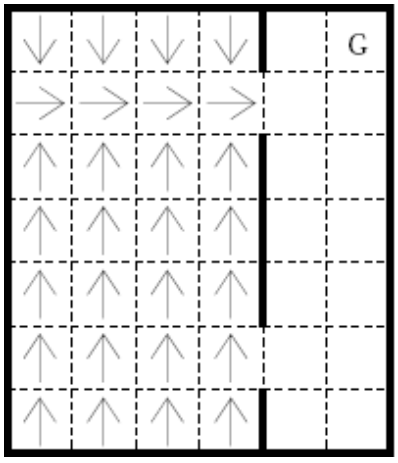


Figura 11. Se muestran dos regiones conectadas. Si el agente se encuentra en cualquier estado de la izquieda, se inicia la macro-acción siguiendo la política ilustrada en la figura y que tiene como objetivo salir de la región.

Una forma de aprender una política sobre las macro acciones es como sigue:

- En s selecciona macro-acción a y siguela
- Observa el estado resultante s', la recompensa r y el número de pasos N.
- Realiza la siguiente actualización:

$$Q(a,s) := (1-\alpha)Q(s,a) + \alpha[r + \gamma^N max_{a'} Q(s',a')]$$

donde N es el número de pasos. Si $\gamma = 1$ se convierte en Q-learning tradicional.

La política óptima puede no estar representada por una combinación de macro acciones, por ejemplo, la macro acción *sal-por-la-puerta-más-cercana* es subóptima para los estados que están una posición arriba de la salida inferior. Esto es, si el estado meta está más cercano a la puerta inferior, el agente recorrería estados de más.

6.3. HAMs

Parr y Russell proponen lo que llaman máquinas jerárquicas abstractas o HAMs por sus siglas en inglés. Una HAM se compone de un conjunto de máquinas de estados, una función de transición y una función de inicio que determina el estado inicial de la máquina [27]. Los estados en las máquinas son de cuatro tipos: (i) *action*, en el cual se ejecuta una acción, (ii) *call*, llama a otra máquina, (iii) *choice* selecciona en forma no determinista el siguiente estado, y (iv) *stop* regresa el control al estado *call* previo. Con esta técnica se resuelve el problema mostrado en la figura 12, en donde se aprenden políticas parciales para: *ir por pasillo* quien llama a *rebotar en pared* y *retroceder*. *Rebotar en pared* llama a *hacia pared* y *seguir pared*, etc.

Una HAM se puede ver como un programa y de hecho Andre y Russell propusieron un lenguaje de programación para HAMs (PHAM) que incorpora al lenguaje LISP [1].

6.4. MAXQ

Dietterich [12] propone un esquema jerárquico llamado MAXQ bajo la suposición que para un programador es relativamente fácil identificar submetas y subtareas para lograr esas submetas. MAXQ descompone el MDP original en una jerarquía de MDPs y la función de valor del MDP original se descompone en una combinación aditiva de las funciones de valor de los MDPs más pequeños. Cada subtarea tiene asociado un conjunto de estados, un predicado de terminación y una función de recompensa. La descomposición jerárquica define un grafo dirigido y al resolver el nodo raíz se resuelve el problema completo. Se obtiene una política para cada subtarea, la cual consiste en acciones primitivas o en llamar a una subtarea. Esto permite reutilizar las políticas de subtareas en otros problemas. Se puede probar que MAXQ obtiene la política óptima consistente con la descomposición.

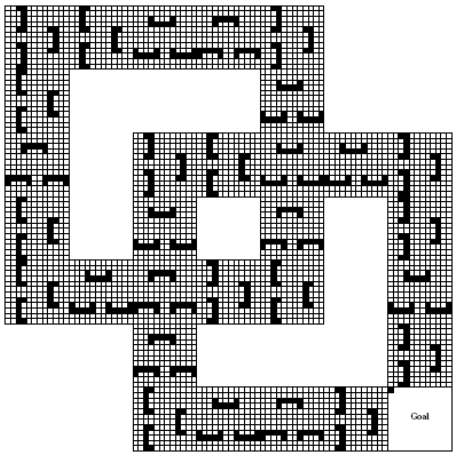


Figura 12. Se muestra un dominio que tiene cierta regularidad que es aprovechada por una HAM para solucionarlo eficientemente.

MAXQ también introduce una abstración de estados, en donde para cada subtarea se tiene un subconjunto de variables reduciendo el espacio significativamente. Por ejemplo en el problema del Taxi mostrado en la figura 13 el objetivo es recoger a un pasajero en uno de los estados marcados con una letra y dejarlo en algún destino (de nuevo marcado con una letra). La parte derecha de la figura muestra un grafo de dependencias en donde la navegación depende sólo de la posición del taxi y el destino y no si se va a recoger o dejar a un pasajero. Por otro lado el completar la tarea depende del origen y destino del pasajero más que en la localización del taxi. Estas abstracciones simplifican la solución de las subtareas.

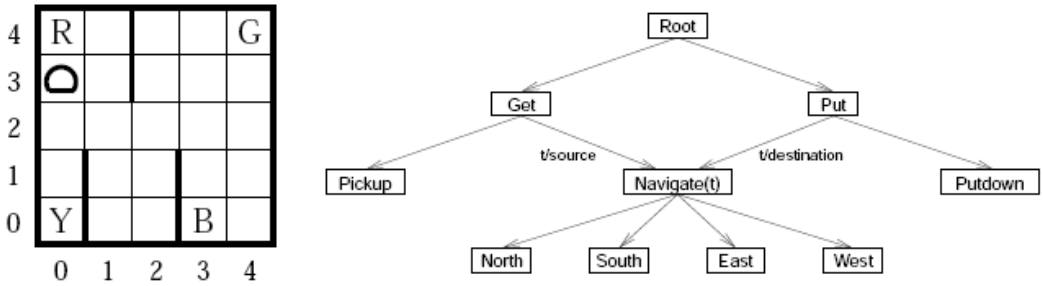


Figura 13. La figura de la izquierda muestra el dominio del Taxi donde las letras indican posiciones para recoger o dejar un pasajero. El grafo de la derecha representa las dependencias entre diferentes posibles sub-tareas a realizar.

Existen diferentes formas de como dividir un problema en sub-problemas, resolver cada sub-problema y combinar sus soluciones para resolver el problema original. Algunos otros enfoques propuestos para MDPs jerárquicos están descritos en [4,2].

6.5. Decomposición Concurrente

La descomposición concurrente divide el problema entre varios agentes, cada uno ve un aspecto del problema, y entre todos resuelven el problema global operando en forma concurrente. Analizaremos aquí dos enfoques: (i) MDPs paralelos (PMDPs), (ii) MDPs multi–seccionados (MS-MDPs).

MDPs paralelos Los MDPs paralelos [35] consisten de un conjunto de MDPs, donde cada uno ve un aspecto del mismo problema, obteniendo la política óptima desde cierto punto de vista. Cada MDP tiene, en principio, el mismo espacio de estados y acciones, pero diferente función de recompensa. Esto asume que la parte del problema que ve cada uno es *relativamente* independiente de los otros. Se tiene un árbitro que coordina a los diferentes MDPs, recibiendo sus posibles acciones con su valor Q, y seleccionando aquella acción que de el mayor valor combinado (asumiendo una combinación aditiva), que es la que se ejecuta.

Formalmente un PMDP es un conjunto de K procesos, $P_1, P_2, ..., P_k$, donde cada P_i es un MDP. Todos los MDPs tienen el mismo conjunto de estados, acciones y funciones de transición: $S_1 = S_2 = ... = S_k$; $A_1 = A_2 = ... = A_k$; $\Phi_1(a, s, s') = \Phi_2(a, s, s') = ... = \Phi_k(a, s, s')$. Cada MDP tiene una función diferente de recompensa, $R_1, R_2, ..., R_k$. La recompensa total, RT, es la suma de las recompensas individuales: $RT = R_1 + R_2 + ... + R_k$.

Por ejemplo, consideremos un robot simulado en una rejilla (ver figura 14). El robot tienen que ir de su posición actual a la meta, y al mismo tiempo evadir obstáculos. Definimos dos MDPs: (i) a *Navegador*, para ir a la meta, y (ii) *Evasor*, para evadir los obstáculos. Ambos tienen el mismo espacio estados (cada celda en la rejilla) y acciones (arriba, abajo, izquierda o derecha); pero una recompensa diferente. El *Navegador* obtiene una recompensa positiva cuando llega a la meta, mientras que el *Evasor* tiene una recompensa negativa cuando choca con un obstáculo.

Si asumimos que las funciones de recompensa y valor son aditivas, entonces la pólitica óptima es aquella que maximize la suma de los valores Q; esto es: $\pi^*_I(s) = argmax_a \sum_{i \in K} Q^*_i(s, a)$, donde $Q^*_i(s, a)$ es el Q óptimo que se obtiene al resolver cada MDP individualmente. En base a esto podemos definir el algoritmo 13 para resolver un PMDP.

La figura 15 muestra otro ejemplo de un PMDP en el mundo de rejillas, con dos agentes, un *Navegador* y un *Evasor*. En este caso la meta está en la celda $5, 5$ (abajo a la izquierda), y observamos que el PMDP obtiene la solución correcta, indicada por la flecha en cada celda.

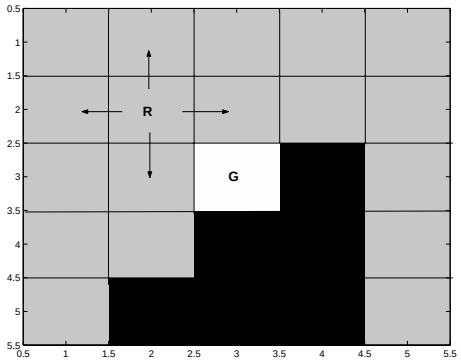


Figura 14. Ejemplo de un robot simulado en una rejilla. El ambiente consiste de celdas que pueden ser libres (gris), obstáculos (negro) o la meta (blanco). El robot (R) puede moverse a cada una de las 4 celdas vecinas como se muestra en la figura.

Algoritmo 13 Solución de un MDP paralelo.

Resolver cada MDP mediante cualquier método de solución básico.
Obtener el valor Q_i óptimo por estado–acción:

$$Q_i^*(s,a) = \{r_i(s,a) + \gamma \Sigma_{s' \in S} \Phi(a,s,s') V_i^*(s')\}$$

Calcular el valor óptimo global de Q: $Q^*(s,a) = \Sigma_{i=1}^{K} Q_i^*(s,a)$
Obtener la política óptima: $\pi^*(s) = argmax_a\{\Sigma_{i=1}^{K} Q_i^*(s,a)\}$

Los PMDPs no reducen directamente el espacio de estados y acciones, pero facilitan la solución de problemas complejos, al poder separar diferentes aspectos del mismo; ya sea si se construye el modelo por un experto y luego se resuleve, o mediante aprendizaje por refuerzo. También ayudan a la abstracción de estados, lo que pudiera llevar a una reducción en el tamaño del problema.

MDPs multi-seccionados Otro enfoque relacionado, llamado MDPs *multi-seccionados* [14], si considera una reducción en el número de estados y acciones. Se asume que se tiene una tarea que se puede dividir en varias sub–tareas, de forma que cada una implica diferentes acciones, y, en general, no hay conflicto entre ellas. Por ejemplo, en un robot de servicio que interactua con personas, se podría dividir el problema en: (i) navegador, permite al robot desplazarse y localizarse en el ambiente, (ii) interacción por voz, realiza la interacción mediante voz con las personas, y (iii) interacción por ademanes, permite la interacción gestual entre el robot y los usuarios. Cada uno de estos aspectos considera una parte del espacio de estados, y acciones diferentes; aunque todos se requiren para realizar la tarea, operan en cierta forma *independiente*.

Un MS-MDP es un conjunto de N MDPs, que comparten el mismo espacio de estados y recompensas (objetivo), pero tienen diferentes conjuntos de acciones. Se asume que las acciones de cada MDP no tienen *conflictos* con los otros pro-

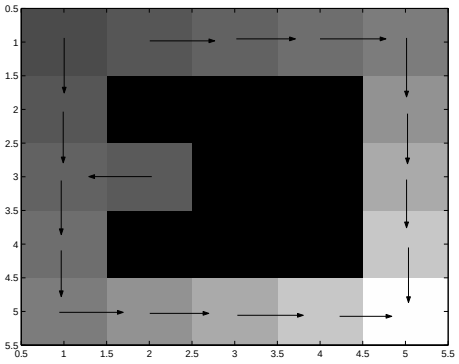


Figura 15. Función de valor y política para una rejilla con la meta en $5, 5$. Para cada celda libre se muestra el valor (como un tono de gris, más claro indica mayor valor) y la acción óptima (flecha) obtenidas al resolver el PMDP.

cesos, de forma que cada uno de los MDPs puede ejecutar sus acciones (basadas en la política óptima) en forma concurrente con los demás. Considerando una representación factorizada del espacio de estados, cada MDP sólo necesita incluir las variables de estado que son relevantes para sus acciones y recompensa, lo que puede resultar en una reducción considerable del espacio de estados–acciones. Esto implica que cada MDP, P_i, sólo tendrá un subconjunto de las variables de estado, que conforman su espacio de estados local, S_i. No se consideran explícitamente los efectos de acciones combinadas.

La solución de un MS-MDP incluye dos fases. En la fase de diseño, fuera de línea, cada MDP se define y se resuelve independientemente, obteniendo la política óptima para cada sub–tarea. En la fase de ejecución, en línea, las políticas de todos los MDPs se ejecutan concurrentemente, seleccionando la acción de acuerdo al estado actual y a la política de cada uno. No hay una coordinación explícta entre los MDPs, ésta se realiza implícitamente mediante el vector de estados común (puede haber algunas variables de estado que estén en el espacio de dos o más de los MDPs).

Como mencionamos anteriormente, se asume que no hay conflictos entre acciones; donde un conflicto es alguna restricción que no permite la ejecución de dos acciones en forma simultánea. Por ejemplo, para el caso del robot de servicio, si el robot tiene una sola cámara que utiliza para navegar y reconocer personas, podría haber conflicto si al mismo tiempo se quieren hacer las dos cosas. Actualmente el MS-MDP da prioridades a las sub–tareas, de forma que en caso de conflicto se ejecutá solamente la acción del MDP de mayor prioridad. En general, el problema de que hacer en caso de conflictos sigue abierto, si se considera encontrar la solución óptima desde un punto de vista global. En la sección 7 se ilustra la aplicación de MS-MDPs para coordinar las acciones de un robot mensajero.

7. Aplicaciones

Para ilustrar la aplicación práctica de MDPs y aprendizaje por refuerzo, en esta sección describimos tres aplicaciones, que ejemplifican diferentes técnicas que hemos analizado en este capítulo:

- Aprendiendo a volar. Mediante una representación relacional y aprendizaje por refuerzo se logra controlar un avión.
- Control de una planta eléctrica. Se utiliza una representación cualitativa de un MDP para controlar un proceso complejo dentro de una planta eléctrica.
- Homer: el robot mensajero. A través de MDPs multi-seccionados se coordina un robot de servicio para que lleve mensajes entre personas.

7.1. Aprendiendo a Volar

Supongamos que queremos aprender a controlar un avión de un simulador de alta fidelidad. Esta tarea la podemos formular como un problema de aprendizaje por refuerzo, donde queremos aprender cuál es la mejor acción a realizar en cada estado para lograr la máxima recompensa acumulada reflejando un vuelo exitoso. Sin embargo, este dominio tiene típicamente entre 20 y 30 variables, la mayoría continuas que describen el movimiento del avión en un espacio tridimensional potencialmente "infinito".

Para esta aplicación se uso un modelo de alta fidelidad de un avión acrobático Pilatus PC-9. El PC-9 es un avión extremadamente rápido y maniobrable que se usa para el entrenamiento de pilotos. El modelo fué proporcionado por la DSTO (Australian Defense Science and Technology Organization) y construído usando datos de un tunel de viendo y del desempeño del avión en vuelo.

En el avión se pueden controlar los alerones, elevadores, impulso, los *flaps* y el tren de aterrizaje. En este trabajo solo se considera como controlar los alerones y elevadores con lo cual se puede volar el avión en vuelo y que son las dos partes más difíciles de aprender. Se asume en los experimentos que se tiene un impulso constante, los *flaps* planos, el tren de aterrizaje retraído y que el avión está en vuelo. Se añade turbulencia durante el proceso de aprendizaje como un cambio aleatorio en los componentes de velocidad del avión con un desplazamiento máximo de 10 pies/seg. en la posición vertical y 5 pies/seg. en la dirección horizontal.

La salida del simulador incluye información de la posición, velocidad, orientación, los cambios en orientación e inclinación vertical y horizontal, y la posición de objetos, tales como edificios, que aparecen en el campo visual. Una misión de vuelo se especifica como una secuencia de puntos por lo que el avión debe pasar con una tolerancia de ± 50 pies, osea que el avión debe de pasar por una ventana de 100 x 100 pies^2 centrada alrededor del punto.

Este problema es difícil para aprendizaje por refuerzo, inclusive con una discretización burda de los parámetros[1]. Además, si se logra aprender una política, ésta solo sirve para cumplir una misión particular y se tendría que aprender una nueva política por cada misión nueva. Lo que nos gustaría es aprender una sola política que se pueda usar para cualquier misión bajo condiciones de turbulencia. Para esto, usamos una representación relacional para representar la posición relativa del avión con otros objetos y con la meta.

Decidimos dividir el problema en dos: (i) movimientos para controlar la inclinación vertical o elevación y (ii) movimientos para controlar la inclinación horizontal y dirección del avión. Para el control de la elevación, los estados se caracterizaron con predicados que determinan la distancia y elevación a la meta. Para el control de los alerones, además de la distancia a la meta se usaron predicados para determinar la orientación a la meta, la inclinación del avión y la tendencia seguida por el avión en su inclinación. Las acciones se pueden aprender directamente de trazas de pilotos usando lo que se conoce como *behavioural cloning* o clonación.

La clonación aprende reglas de control a partir de trazas de expertos humanos, e.g., [33,22,6]. Se sigue un esquema relativamente simple: De la información de una traza de vuelo, por cada estado se evaluan los predicados que aplican y se obtiene la acción realizada. Si la acción de control es una instancia de una acción anterior, no se hace nada, de otro forma se crea una nueva acción con una conjunción de los predicados que se cumplieron en el estado. Este esquema también puede usarse en forma incremental, incorporando nuevas acciones en cualquier momento (ver también [23] para una propuesta similar en ajedrez).

En total se aprendieron 407 acciones, 359 para alerón (de un total de 1,125 posibles) y 48 para elevación (de un total de 75 posibles) después de cinco trazas de vuelo realizadas por un usuario y de 20 trazas seguidas en un esquema de exploración para visitar situaciones no vistas en las trazas originales. Con esto solo se aprenden un subconjunto de las acciones potenciales (una tercera parte) lo cual simplifica el proceso de aprendizaje por refuerzo con un promedio de 1.6 acciones por cada estado de alerón y 3.2 por estado de elevación.

Una vez que se inducen las acciones se usa rQ-learning para aprender una política que permita volar el avión. En todos los experimentos los valores Q se inicializaron a -1, $\epsilon = 0{,}1, \gamma = 0{,}9, \alpha = 0{,}1$, y $\lambda = 0{,}9$ (dado que usamos trazas de elegibilidad). Los experimentos se hicieron con ocho metas en un recorrido como se muestra en la figura 16. Si la distancia del avión se incrementa con respecto a la meta actual durante 20 pasos de tiempo, se asume que se paso de la meta y se le asigna la siguiente meta.

La política aprendida después de 1,500 vuelos es capaz de volar la misión completa en forma exitosa. Se probó su robustez bajo diferentes condiciones de

[1] Por ejemplo, considerando un espacio bastante pequeño de 10 km^2 con 250m. de altura. Una discretización burda de $500 \times 500 \times 50$ m. con 5 posibles valores para la orientación e inclinaciones verticales y horizontales, y 5 posibles acciones por estado, nos da 1,250,000 pares estado-acción.

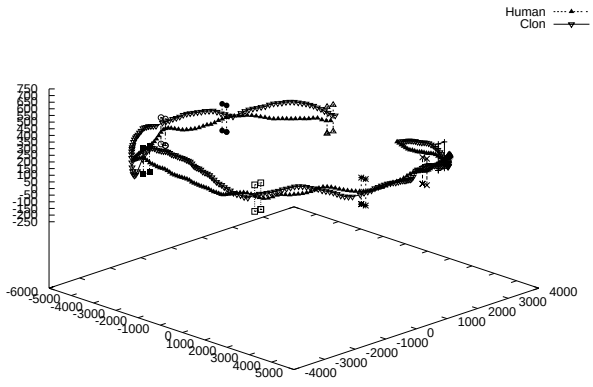


Figura 16. Traza de un humano y seguida por la política aprendida con aprendizaje por refuerzo y clonación en la misión de ocho metas y con el máximo nivel de turbulencia.

turbulencia. La figura 16 muestra una traza de un humano en esta misión y la traza seguida por la política aprendida. La política aprendida se probó en otra misión de cuatro metas. La intención era probar maniobras no vistas antes. La nueva misión incluyó: una vuelta a la derecha[2], una vuelta cerrada a la izquierda subiendo no vista antes, otra vuelta rápida a la derecha y una vuelta cerrada de bajada a la derecha.

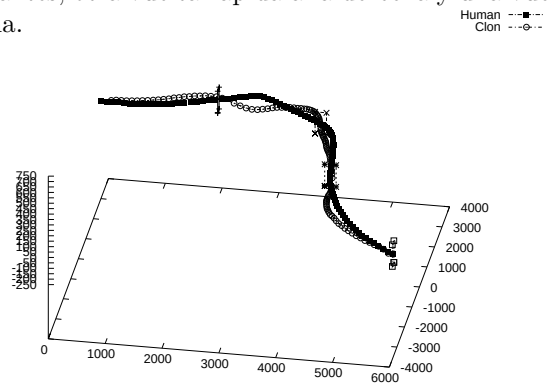


Figura 17. Traza de vuelo de un humano y de la política aprendida de la primera misión en una nueva misión de cuatro metas.

La figura 17 muestra una traza de un humano y de la política aprendida, en una nueva misión con un valor máximo de turbulencia. Claramente, la política aprendida puede volar en forma razonable una misión completamente nueva.

[2] La misión de entrenamiento tiene solo vueltas a la izquierda.

7.2. Control de una Planta Eléctrica

Una planta de generación de energía eléctrica de ciclo combinado calienta vapor de agua que alimenta una turbina de gas y luego recupera parte de esa energía para alimentar a una turbina de vapor y generar más electricidad. Un generador de vapor, usando un sistema de recuperación de calor, es una proceso capaz de recuperar la energía sobrante de los gases de una turbina de gas, y usarlos para generar vapor. Sus componentes principales son: el domo, la pared de agua, la bomba de recirculación y los quemadores. Por otro lado, los elementos de control asociados a la operación son: la válvula de agua de alimentación, la válvula de combustible de los quemadores, la válvula de vapor principal, la válvula de paso y la válvula de gas. Un diagrama simplificado de una planta eléctrica se muestra en la figura 18.

Durante la operación normal de la planta, el sistema de control regula el nivel de vapor del domo. Sin embargo, cuando existe un rechazo de carga parcial o total, los sistemas de control tradicionales no son capaces de estabilizar el nivel del domo. En este caso, se crea un incremento en el nivel de agua debido al fuerte desplazamiento de agua en el domo. El sistema de control reacciona cerrando la válvula de alimentación de agua. Algo parecido sucede cuando existe una alta demanda repentina de vapor. Bajo estas circustancias, la intervención de un operador humano es necesaria para ayudar al sistema de control a tomar las acciones adecuadas para salir del transitorio. Una solución práctica es usar recomendaciones de un asistente de operador inteligente que le diga las mejores acciones a tomar para corregir el problema. Para esto se modelo el problema como un MDP [31].

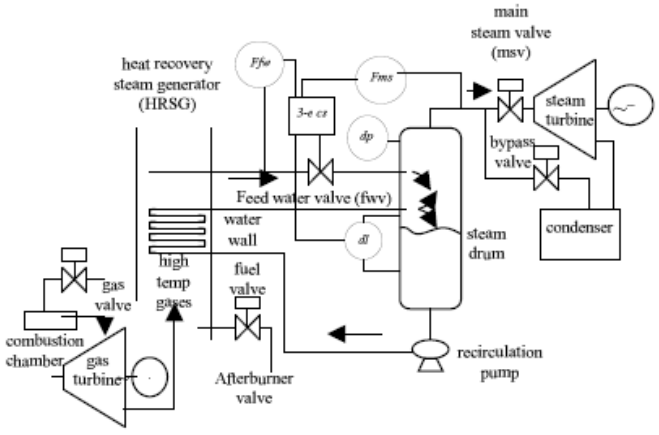


Figura 18. Diagrama de una planta de energía eléctrica de ciclo combinado.

El conjunto de estados en el MDP se obtiene directamente de las variables de operación del generador de vapor: el flujo de agua de alimentación (Ffw), el flujo principal de vapor (Fms), la presión del domo (Pd) y la generación de potencia (g), como las variables a controlar y un disturbio (d) como evento exógeno. Inicialmente se considera el "rechazo de carga" como el disturbio. Los valores de estas variables representan el espacio de los posibles estados del sistema.

Para la operación óptima de la planta, existen ciertas relaciones entre las variables que se deben de cumplir, especificadas por la curva de operación recomendada. Por ejemplo, la curva deseada para la presión del domo y el flujo principal de vapor se muestra en la figura 19.

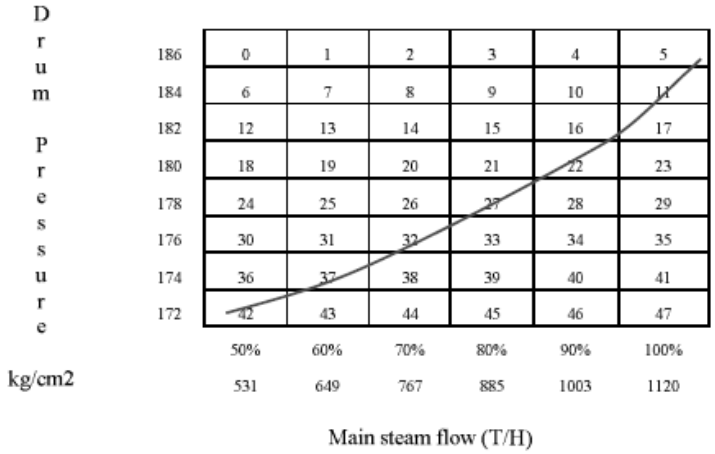


Figura 19. Curva de operación deseada para una planta de ciclo combinado. La curva establece las relaciones *óptimas* de operación entre la presión del domo y el flujo de vapor.

La función de recompensa para el MDP se basa en la curva de operación óptima. Los estados en la curva reciben refuerzo positivo y el resto de los estados un refuerzo negativo. El objetivo entonces del MDP es encontrar la política óptima para llevar a la planta a un estado dentro de la curva de operación deseada.

El conjunto de acciones se basa en abrir y cerrar las válvulas de agua de alimentación (fwv) y de vapor principal (msv). Se asume que estas válvulas regulan el flujo de agua de alimentación y el flujo principal de vapor.

Para resolver este problema se utilizó la representación cualitativa de MDPs [32] descrita en la sección 5.3. Para esto se aprendió un MDP cualitativo a partir de datos generados en el simulador. Primeramente, se generaron datos simulados con información de las variables de estado y recompensas de acuerdo a las curvas de operación óptima de la planta. A partir de estos datos se aprendió un árbol

de decisión para predecir el valor de recompensa en términos de las variables de estado usando la implementación de C4.5 [30] de Weka [40]. El árbol de decisión se transformó en árbol cualitativo en donde cada hoja del árbol es asociada a un valor cualitativo. Cada rama del árbol representa un estado cualitativo. Los datos iniciales se expresan en términos de los rangos encontrados en el árbol para estas variables, y el resto de las variables que no están incluidas en el árbol se discretizan. Los datos se ordenan para representar transiciones de valores entre un tiempo t y un siguiente tiempo $t + 1$. A partir de estos datos secuenciales se aprende una red bayesiana dinámica de dos etapas por cada acción del problema, en esta caso abrir y cerrar válvulas, usando el algoritmo de aprendizaje bayesiano K2 [9] implentado en Elvira [17].

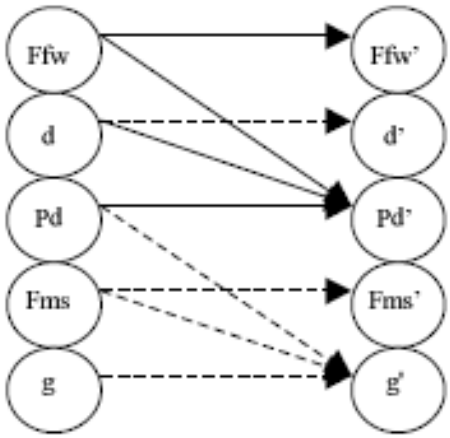


Figura 20. Red bayesiana dinámica que representa la función de transición para las acciones abrir o cerrar la válvula de de agua de alimentación.

La figura 20 muestra la red bayesiana dinámica que representa el modelo de transición para las acciones de abrir y cerrar la válvula de agua de alimentación. Las líneas sólidas representan relaciones entre nodos relevantes y las líneas punteadas representan aquellos nodos que no tienen efecto durante la misma acción.

Al transformar el espacio continuo inicial a una representación de alto nivel, el problema puede ser resuelto usando técnicas estandar de solución de MDPs, tales como iteración de valor, para obtener la política óptima. La figura 21 muestra una partición de estados automática y la política encontrada para una versión simplificada del problema con 5 acciones y 5 variables. Esta política fué validada por un experto humano, encontrandose satisfactoria. Se están realizando pruebas con un simulador para validarla experimentalmente.

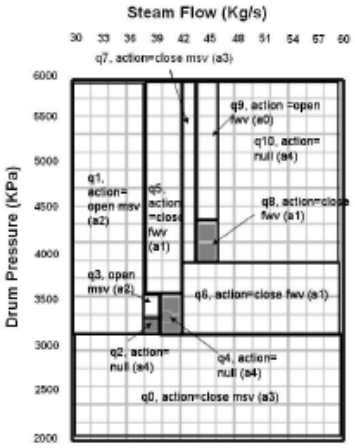


Figura 21. Partición cualitativa del espacio de estados, proyectándolo a las dos variables que tienen que ver con la recompensa. Para cada estado cualitativo, q_1 a q_10 se indica la acción óptima obtenida al resolver el MDP cualitativo.

7.3. Homer: El robot mensajero

Homer [15], ver figura 22, es un robot móvil basado en la plataforma *Real World Interface B-14* que tiene un sólo sensor: una cámara estéreo (Point Grey Research Bumblebee^{TM}).

El objetivo de Homer es llevar mensajes entre personas en un ambiente de interiores, por ejemplo entre oficinas en una empresa. Para ello cuenta con un mapa previo del ambiente, mediante el cual puede localizarse y planear su ruta; además de que sabe en que lugar se encuentra, normalmente, cada persona. Tiene la capacidad de interactuar con las personas mediante síntesis y reconocimiento de voz, interpretando comandos sencillos y guardando los mensajes que se desean enviar. Cuenta con una *cara* simulada, que le permite expresar diferentes "emociones"(alegría, tristeza, enojo y neutro), ver figura 22(b), para comunicarse con las personas.

El software de Homer está basado en una arquitectura de capas mediante comportamientos, de forma que cada una de las capacidades de Homer está contenida en un módulo. Los módulos o comportamientos de Homer incluyen: navegación, [25], localización, [25,34], detección de caras, [16], generación de expresiones faciales, planeación, [13], reconocimiento de voz, [16], detección de voz, y monitor de recursos. Mediante la combinación de estos comportamientos Homer realiza la tarea de entrega de mensajes.

Para coordinar los módulos se plantea el uso de un MDP, teniendo como objetivo el recibir y entregar mensajes, de forma que Homer reciba recompensas positivas. De esta forma al resolver el MDP, se obtienen las acciones que debe realizar Homer en cada momento (en función del estado) para llegar al objetivo.

(a) (b)

Figura 22. (a) Homer, el robot mensajero interactuando con una persona. (b) Un acercamiento de la *cabeza* de Homer.

Pero al considerar todos los posibles estados y acciones en este problema, el espacio crece a aproximadamente $10,000,000$ estados–acciones, por lo que se vuelve imposible resolverlo como un sólo MDP. Para esto, se utiliza el enfoque de MDPs multi–seccionados [14], dividiendo la tarea en un conjunto de sub–tareas que puedan ser resueltas en forma independiente, y ejecutadas concurrentemente. Utilizando MS-MDPs, el MDP más complejo tiene $30,000$ estados–acciones.

La tarea de entrega de mensajes se puede dividir en 3 sub–tareas, cada una controlada por un MDP. El **N**avegador controla la navegación y localización del robot. El manejador de **D**iálogos controla la interacción con las personas mediante voz. El **G**enerador de gestos controla las expresiones realizadas mediante la cara animada para la interacción con las personas. Cada MDP considera sólo las variables relevantes para la sub–tarea dentro del espacio de estados de Homer, y controla varios de los módulos mediante sus acciones. El estado completo consiste de 13 variables que se muestran en la tabla 1. Para cada variable se indica cual(es) de los MDPs la consideran. En la tabla 2 se listan las accions de cada MDP, y los módulos que las implementan.

Los 3 MDPs se resolvieron utilizando el sistema SPUDD (ver sección 5), generandose la política óptima para cada uno. Para coordinar la tarea se ejecutan concurrentemente las 3 políticas mediante un manejador, que simplemente lee las variables de estado, y selecciona la acción óptima para cada sub–tarea de acuerdo a la política. Durante la ejecución de las acciones se evitan conflictos potenciales, simplemente dando prioridad al navegador.

Homer fue evaluado experimentalmente realizando tareas de recepción y entrega de mensajes entre varias personas en el Laboratorio de Inteligencia Computacional en la Universidad de la Columbia Británica en Canadá [14]. Homer logró realizar exitosamente la tarea en varias ocaciones. La tabla 3 ilustra varias etapas durante la ejecución de la tarea de recepción y entrega de un mensaje.

Variable	MDP
Tiene mensaje	N,D,G
Nombre receptor	N,D,G
Nombre del que envía	N,D,G
En la meta	N,D,G
Tiene meta	N
Meta inalcanzable	N
Receptor inalcanzable	N
Batería baja	N
Localización incierta	N
Voz escuchada (speech)	D,G
Persona cerca	D,G
Llamaron a Homer	D,G
Yes/No	D,G

Cuadro 1. Variables de estado para Homer. Para cada una se indican los MDPs que la incluye.

MDP	acciones	módulos
Navegador	explorar	navegación
	navegar	navegación
	localizarse	localización
	obten nueva meta	location generator
	va a casa	navegación
	espera	navegación
Diálogo	pregunta	síntesis de voz
	confirma	síntesis de voz
	entrega mensaje	síntesis de voz
Gestos	neutral	generación de gestos
	feliz	generación de gestos
	triste	generación de gestos
	enojado	generación de gestos

Cuadro 2. Los MDPs que coordinan a Homer, y sus acciones. Para cada acción se indica el módulo que la implementa.

Cuadro 3. Ejemplo de una corrida para entregar un mensaje. Para cada estapa se muestra: (i) una imagen representativa, (ii) variables significativas, (iii) acciones que toma cada MDP (**N**avegador, **D**iálogo, **G**estos), y una breve descripción.

imagen	variables cambiadas	acciones	descripción
	Persona cerca = true Voz escuchada = true	**N**: - **D**: pregunta **G**: feliz	Se acerca persona. HOMER inicia conversación preguntandole su nombre y sonriendo.
	tiene_mensaje = true Tiene meta = true Voz escuchada = false Nombre envía = 4 Nombre Receptor = 1	**N**: navega **D**: - **G**: neutral	HOMER recive mensaje y destinatario y obtiene el destino de su módulo de planificación. El navegador empieza a moverlo hacia el destino.
	Persona cerca=false Loc. incierta = true	**N**: localizarse **D**: - **G**: -	Durante la navegación, la posición de HOMER se volvió incierta. El MDP se navegación se localiza. Los estados de los otros dos MDPs no se ven afectados.
	en_destino=true Voz escuchada=true yes/no=yes	**N**: espera **D**: entrega mensaje **G**: feliz	HOMER llego a su destino, detectó una persona y confirmó que era el receptor (a través de *yes/no*). Entrega mensaje y sonrie.
	tiene_mensaje = false En la meta=false Batería baja=true	**N**: ve a casa **D**: - **G**: -	Las baterías de HOMER se detectan bajas, el MDP navegador regresa a HOMER a casa.

8. Conclusiones

Un MDP modela un problema de decisión sequencial en donde el sistema evoluciona en el tiempo y es controlado por un agente. La dinámica del sistema está determinada por una función de transición de probabilidad que mapea estados y acciones a otros estados. El problema fundamental en MDPs es encontrar una política óptima; es decir, aquella que maximiza la recompensa que espera recibir el agente a largo plazo. La popularidad de los MDPs radica en su capacidad de poder decidir cual es la mejor acción a realizar en cada estado para lograr cierta meta, en condiciones inciertas.

En este capítulo hemos revisado los conceptos básicos de los procesos de decisión de Markov y descrito las principales técnicas de solución. Existen diferentes técnicas para solucionar MDPs dependiendo de si se conoce un modelo del ambiente o no se conoce. Independientemente de esto, la obtención de una solución en tiempo razonable se ve fuertemente afectada por el tamaño del espacio de estados y acciones posibles. Para esto se han planteado propuestas como factorizaciones, abstracciones y descomposiciones del problema, que buscan escalar las técnicas de solución a problemas más complejos. Estos avances han servido para controlar elevadores, jugar *backgammon*, controlar aviones y helicópteros, controlar plantas de generación de electricidad, servir como planificadores para robots, seleccionar portafolios financieros, y control de inventarios, entre otros.

El área de MDPs ha mostrado un fuerte desarrollo en los últimos años y se espera que se siga trabajando en extensiones; en particular en cómo combinar varias soluciones de MDPs diferentes un una solución global, en como re-utilizar las soluciones obtenidas otros problemas similares, y en aplicaciones a dominios más complejos. Adicionalmente, se están desarrollando técnicas aproximadas para resolver POMDPs, en los cuales existe incertidumbre respecto al estado, por lo que se vuelven mucho más complejos.

9. Ejercicios propuestos

1. Dada la red bayesiana de la figura 23 (todas las variables son binarias): (a) Completa las matrices de probabilidad que definan a la red bayesiana, (b) Identifica las relaciones de independencia condicional que se representan en el grafo, (c) Obten la probabilidad conjunta aplicando la regla de la cadena. Probabilidades:
   $P(a1) = 0{,}9$, $P(b1 \mid a1) = 0.8$, $P(b1 \mid a2) = 0{,}3$, $P(c1 \mid b1) = 0{,}9$, $P(c1 \mid b2) = 0{,}6$, $P(d1 \mid a1) = 0{,}6$, $P(d1 \mid a2) = 0{,}2$
2. Dado el ejemplo de la figura 1, completar la definición del modelo del MDP, incluyendo los estados, acciones, función de recompensa y función de transición.
3. Resolver el MDP del problema anterior mediante: (a) iteración de valor, (b) iteración de política.

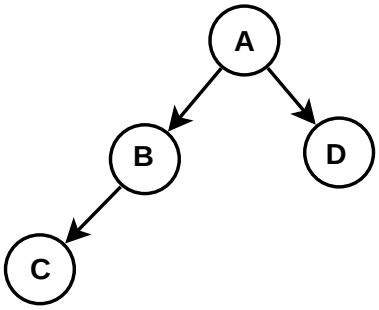


Figura 23. Red Bayesiana con cuatro variables binarias.

4. Resolver el MDP del problema anterior con una técnica de Monte Carlo asumiendo una política fija tomada de tu solución del problema anterior. Resolver el problema usando una estrategia Monte Carlo con una política inicial exploratoria y una estrategia para mejorar políticas. Explicar las diferencias (ventajas y desventajas) de las dos técnicas.
5. Resolver ahora el MDP anterior mediante TD(0), SARSA y Q-learning. Explica cómo se comparan estas técnicas.
6. En base a las soluciones de los tres problemas anteriores explica las ventajas y desventajas de resolver un MDP usando técnicas de programación dinámica, técnicas de Monte Carlo y técnicas de aprendizaje por refuerzo.
7. Resolver el problema anterior usando trazas de elegibilidad. Explicar cómo se compara en cuanto a tiempo de covergencia con la soluciones obtenidas usando los métodos de aprendizaje por refuerzo.
8. Resolver el problema anterior usando Dyna-Q. Explicar cómo se compara con respecto a técnicas de aprendizaje por refuerzo y a térnicas de programación dinámica.
9. Resuelve el ejemplo de la figura 1 como un MDP paralelo: (a) Especifica un MDP para navegar que no considera obstáculos, sólo las metas. (b) Especifica un MDP que evade obstáculos, pero que no conoce la posición de la meta. (c) Obten la solución de cada uno de los dos MDPs anteriores. (d) Combina las soluciones de ambos MDPs en base a sus funciones Q para obtener la política global (e) Compara la poítica obtenida con la solución del problema como un solo MDP del problema anterior.
10. Para el ejemplo del MDP factorizado de la figura 5, asume que todos las variables son binarias (tienen dos valores, F y T): (a) Especifica las tablas de probabilidad condicional de cada variable en $t+1$ de forma que satisfagan los axiomas de probabilidad. (b) Especifica la función de recompensa en base a $X1$ y $X2$ (como se muestra en la figura) en forma de tabla y en forma de árbol de decisión. (c) En base a las tablas de probabilidad establecidas en el inciso (a), calcula la probabilidad de las variables en $t+1$ si el estado en t es: $X1 = F$, $X2 = T$, $X3 = T$, $X4 = F$, $X5 = T$.

Referencias

1. D. Andre and S. Russell. Programmable reinforcement learning agents. In *Advances in Neural Information Processing Systems*, volume 13, 2000.
2. Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003.
3. R.E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, 1957.
4. C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artifical Intelligence Research*, 11:1–94, 1999.
5. Craig Boutilier, Richard Dearden, and Moise's Goldszmidt. Exploiting structure in policy construction. In Chris Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1111, San Francisco, 1995. Morgan Kaufmann.
6. I. Bratko, T. Urbančič, and C. Sammut. Behavioural cloning: phenomena, results and problems. automated systems based on human skill. In *IFAC Symposium*, Berlin, 1998.
7. D. Chapman and L. Kaelbling. Input generalization in delayed reinforcement learning: an algorithm and performance comparison. In *Proc. of the International Joint Conference on Artificial Intelligence*, pages 726–731, San Francisco, 1991. Morgan Kaufmann.
8. N. Charness. Human chess skill. In P.W. Frey, editor, *Chess skill in man and machine*, pages 35–53. Springer-Verlag, 1977.
9. G.F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–348, 1992.
10. A. de Groot. *Thought and Choice in Chess.* Mouton, The Hague, 1965.
11. T. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.
12. T.G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.
13. P. Elinas, J. Hoey, D. Lahey, J. Montgomery, D. Murray, S. Se, and J.J. Little. Waiting with Jose, a vision based mobile robot. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA '02)*, Washington, D.C., May 2002.
14. P. Elinas, L.E. Sucar, J. Hoey, and A. Reyes. A decision-theoretic approach for task coordination in mobile robots. In *Proc. IEEE Robot Human Interaction (RoMan)*, Japan, September 2004.
15. Pantelis Elinas, Jesse Hoey, and James J. Little. Human oriented messenger robot. In *Proc. of AAAI Spring Symposium on Human Interaction with Autonomous Systems*, Stanford, CA, March 2003.
16. Pantelis Elinas and James J. Little. A robot control architecture for guiding a vision-based mobile robot. In *Proc. of AAAI Spring Symposium in Intelligent Distributed and Embedded Systems*, Stanford, CA, March 2002.
17. The Elvira Consortium. Elvira: An environment for creating and using probabilistic graphical models. In *Proceedings of the First European Workshop on Probabilistic graphical models (PGM'02)*, pages 1–11, Cuenca, Spain, 2002.
18. J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of International Conference on Uncertainty in Artificial Intelligence (UAI '99)*, Stockholm, 1999.

19. R. A. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, Mass., U.S.A., 1960.
20. L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2), 1998.
21. L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2), 1998.
22. D. Michie, M. Bain, and J. Hayes-Michie. Cognitive models from subcognitive skills. In M. Grimble, J. McGhee, and P. Mowforth, editors, *Knowledge-Based Systems in Industrial Control*, pages 71–90. Peter Peregrinus, 1990.
23. E. Morales. On learning how to play. In H.J. van den Herik and J.W.H.M. Uiterwijk, editors, *Advances in Computer Chess 8*, pages 235–250. Universiteit Maastricht, The Netherlands, 1997.
24. E. Morales. Scaling up reinforcement learning with a relational representation. In *Proc. of the Workshop on Adaptability in Multi-agent Systems (AORC-20 03)*, pages 15–26, 2003.
25. D. Murray and J. Little. Using real-time stereo vision for mobile robot navigation. *Autonomous Robots*, 8:161–171, 2000.
26. A.E. Nicholson and J.M. Brady. Dynamic belief networks for discrete monitoring. *IEEE Trans. on Systems, Man, and Cybernetics*, 24(11):1593–1610, 1994.
27. Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1997.
28. J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann, San Francisco, CA., 1988.
29. M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, New York, NY., 1994.
30. J. R. Quinlan. *C 4.5: Programs for machine learning.* The Morgan Kaufmann Series in Machine Learning, San Mateo, CA: Morgan Kaufmann, —c1993, 1993.
31. A. Reyes, P. H. Ibarguengoytia, and L. E. Sucar. Power plant operator assistant: An industrial application of factored MDPs. *MICAI 2004: Advances in Artificial Intelligence*, pages 565–573, 2004.
32. A. Reyes, L. E. Sucar, E. Morales, and Pablo H. Ibarguengoytia. Abstraction and refinement for solving Markov Decision Processes. In *Workshop on Probabilistic Graphical Models PGM-2006*, pages 263–270, Chezch Republic, 2006.
33. C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In *Proc. of the Ninth International Conference on Machine Learning*, pages 385–393. Morgan Kaufmann, 1992.
34. S. Se, D. Lowe, and J.J. Little. Mobile robot localization and mapping with uncertainty using scale-invariant landmarks. *International Journal of Robotics Research*, 21(8):735–758, August 2002.
35. L.E. Sucar. Parallel markov decision processes. In *2nd European Workshop on Probabilistic Graphical Models*, pages 201–208, Holland, 2004.
36. R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.
37. R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

38. G. Theocharous, K. Rohanimanesh, and S. Mahadevan. Learning hierarchical partially observable Markov decision process models for robot navigation. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA '01)*, Seoul, Korea, May 2001.
39. S. Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 21(4):93–109, 2000.
40. Ian H. Witten and Eibe Frank. *Data mining: practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

Capítulo 10.

Representación y razonamiento temporal en Inteligencia Artificial

1. Introducción

La noción del tiempo es omnipresente y vital en la modelación de los fenómenos naturales y actividades humanas. En *Las confesiones* escritas entre 397 A. D. y 398 A. D. (Warner, 1963) por San Agustín de Hipona, filósofo genio y teólogo, se tratan de manera elocuente y apasionada las preguntas espirituales eternas, que conmovían mentes y corazones de los seres humanos pensativos desde los inicios de los tiempos. Particularmente, el 11vo libro de *Las confesiones* propone reflexiones sobre el génesis y la búsqueda del significado del concepto, tiempo; planteando las siguientes preguntas/problemas interesantes:

¿Quién puede explicar fácil y brevemente al tiempo?

¿Quién puede comprender al tiempo, aun en su pensamiento, de tal forma que pronuncie una palabra sobre él?

¿Qué es lo que mencionamos en un discurso con más familiaridad y conocimiento que, el tiempo?

Entendemos, cuando hablamos de tiempo;
entendemos también, cuando escuchamos a otros hablar del tiempo.

¿Entonces, qué es el tiempo?

Si nadie me pregunta, lo sé;
Pero si deseo explicarlo a alguien que me pregunta, no lo sé.

Generalmente, el hablar, representar y razonar de manera temporal juegan un papel fundamental y es cada vez más importante en el área de Ciencias de la Computación, y particularmente, en el dominio de la Inteligencia Artificial. Una de las habilidades temporales más simples, y al mismo tiempo más importantes, es el manejo de la información relacionada con el tiempo que involucra preguntas como, por ejemplo, "¿cuándo fue recogido el auto del garaje?", "¿qué sucedió después de que la tienda fue cerrada?", "¿hasta cuándo se encontraba el sospechoso fuera de su casa?" y así sucesivamente. De hecho, parece que el tiempo desempeña el capítulo de una referencia universal común— todo está relacionado por una referencia temporal, aunque las referencias temporales pueden tener diversas formas:

- *Entidades temporales absolutas*, por ejemplo, "11 AM del 26 de agosto del 1996", "6 PM", y "de 8 AM hasta 5 PM", que refieren a elementos explícitos del tiempo;
- *Entidades temporales relativas*, por ejemplo, "durante el tiempo que el conductor se encontraba en su oficina" y "después de la medianoche", se refieren a los elementos temporales que se conocen solamente por sus relaciones temporales relativas con otros elementos temporales, los cuales a su vez, pueden ser absolutos o relativos;
- *Duraciones temporales absolutas*, por ejemplo, "45 minutos" y "16 horas", que refieren a cierta cantidad de granularidad temporal;
- *Duraciones temporales relativas*, por ejemplo, "menos de 3 horas" y "más de 6 años pero menos de 10 años", que refieren a una cierta, pero indeterminada cantidad de granularidad temporal.

El problema de razonar con información temporal mezclando estas formas tiene dos sentidos:

- ¿Cómo representar los diferentes tipos de conocimiento temporal?
- ¿Cómo desarrollar un método confiable de inferencia basado en esta representación?

El capítulo se organiza de la manera siguiente: En la sección 2, trataremos algunos problemas fundamentales relacionados con teorías y modelos temporales. Se presentan las consideraciones ontológicas con respecto a primitivas temporales y a relaciones de orden. En la sección 3, se introduce los tres enfoques principales de la representación de la información temporal en la Inteligencia Artificial. Una amplia gama de meta-predicados para asignar valores booleanos a los elementos temporales, y una clasificación detallada de proposiciones temporales se presentan en las secciones 4 y 5 respectivamente. La sección 6 contiene algunos ejemplos interesantes de la representación temporal y el razonamiento temporal. Finalmente, la sección 7 proporciona un breve resumen.

2. Teorías y modelos temporales

Todos los sistemas computacionales que manejan proposiciones relacionados con el tiempo adoptan una teoría (explícita o implícita) del mismo. Se requiere que esta teoría satisfaga nuestras ideas intuitivas relacionadas con el tiempo, de modo que podríamos decir que el mundo real es un modelo donde se aplica esta teoría. Por consiguiente, siguiendo las ideas de Suppes (1961) y Funk (1983), es necesario que las afirmaciones de la teoría correspondan a situaciones verdaderas del mundo real.

2.1 Primitivas temporales

Durante muchos años ha habido una discusión en la literatura de la computación, particularmente en el dominio de la Inteligencia Artificial, sobre el problema de, qué clase de objetos deben considerarse como las partes elementales de tiempo (Ma and Hayes, 2006). Por un lado, los puntos temporales son necesarios para modelar fenómenos temporales tanto en la teoría como en la práctica. Por ejemplo, es intuitivo y conveniente asociar eventos puntuales, tales como "la velocidad del cohete que ha sido lanzado al aire llegó a ser cero cuando alcanzó la cima" y "la base de datos se actualizó a las 0:00 de la medianoche", etc., con los puntos instantáneos más que los intervalos que tienen alguna duración. Por otro lado, los intervalos de tiempo también son necesarios para representar los fenómenos temporales que tienen cierta duración positiva, por ejemplo, "el cohete que ha sido lanzado al aire cayó de la cima a la tierra", "él corrió 3 millas ayer por la mañana", y así sucesivamente.

2.1.1 Sistemas temporales basados en puntos

Una estructura típica del tiempo basada en puntos es un ordenamiento (P, $\leq$), donde P es un conjunto de puntos, y $\leq$ es una relación que (parcialmente o totalmente) ordena P. Los sistemas basados en puntos, los intervalos se pueden definir como objetos temporales derivados: como un conjunto de puntos (McDermott, 1982; van Benthem, 1983), o como pares ordenados de puntos (Bruce, 1972; Shoham, 1987; Ladkin, 1987, y 1992; Halpern y Shoham, 1991). Sin embargo, algunos investigadores argumentan que definir intervalos como objetos derivados de puntos pueden presentar el problema conocido como *Problema de División Instantánea* (van Benthem, 1983; Allen, 1983; Galton 1990; Vila 1994; Ma and Knight, 2003); de hecho, es un rompecabezas antiguo el intentar representar, lo que sucede con el punto en la frontera que divide dos intervalos sucesivos. Por ejemplo, considere una afirmación relacionada con el fuego, como se cita en (van Benthem, 1983):

El fuego que había estado ardiendo después se consumió (el fuego se extinguió).

Intuitivamente, podemos asumir que dos estados son posibles, es decir, "*el fuego ardía*" y "*el fuego no ardía*" en dos intervalos sucesivos basados en puntos, digamos, $<p_1, p>$ y $<p, p_2>$, respectivamente; entonces la pregunta se convierte en "*¿el fuego ardía o no ardía en el punto p?*" Esto, en términos de la naturaleza de los intervalos "abiertos" o "cerrados" basados en puntos involucrados, resulta ser la cuestión, ¿cuál de los dos intervalos sucesivos, es decir, $<p_1, p>$ y $<p, p_2>$, es cerrado/abierto en el punto de la división p? Virtualmente, existen cuatro casos posibles:

(a) El fuego ardía más que lo que no ardía en el punto p;
(b) El fuego no ardía más que lo que ardía en el punto p;
(c) El fuego ardía y no ardía en el punto p;
(d) El fuego no ardía ni no ardía en el punto p.

Mientras que (c) y (d) son absurdos, puesto que violan la ley de la contradicción y la ley del tercero excluido, respectivamente (van Benthem, 1983), la elección entre (a) y (b) es arbitraria y artificial. De hecho, no tenemos una mejor razón, desde el punto de vista filosófico, para decir que el fuego ardía ni para decir que no ardía en el punto instantáneo de la división, este enfoque arbitrario ha sido criticado como injustificable y por lo tanto, insatisfactorio (van Benthem, 1983; Allen, 1983; Galton, 1990; Vila, 1994).

2.1.2 Sistemas temporales basados en intervalos

La estructura basada en puntos de tiempo ha sido desafiada por muchos investigadores que creen que los intervalos de tiempo sirven mejor para representar conocimiento temporal de sentido común, especialmente en el dominio de la lingüística y de la inteligencia artificial. Por lo que, los intervalos deben tomarse como una primitiva temporal, mientras que los puntos se pueden construir como un estado derivado, por ejemplo, "máximo anidamiento" de intervalos que compartan una intersección común (Whitehead, 1929), o como "lugares de reunión" de intervalos (Allen and Hayes, 1985; 1989; Hayes and Allen, 1987; Galton, 1990). Por ejemplo, la teoría temporal de Allen (Allen, 1981; 1983; 1984) es un ejemplo representativo del enfoque basado en intervalos, que postula un conjunto de intervalos como entidades temporales primitivas. Sobre los intervalos del tiempo, Allen introduce trece relaciones temporales, incluyendo "Se encuentra junto (*meets*)", "Encontrado junto con (*met_by*)", "Igual (*equal*)", "Antes (*before*)", "Después (*after*)", "Traslapar (*overlaps*)", "Traslapado_por (*overlapped_by*)", "Empieza (*starts*)", "Empezado_por (*starts_by*)", "Durante (*during*)", "Contiene (*contains*)", "Termina (*finishes*)" y "Terminado_por (*finished_by*)", que se puede derivar de la relación "Se encuentra junto (*meets*)" (Allen and Hayes, 1989).

Como afirma Allen en sus trabajos (Allen, 1981; 1983; 1984), el enfoque basado en intervalos evita la molesta pregunta de, sí o no, un punto dado es parte, o es miembro de un intervalo dado, y por tanto, puede superar con éxito los rompecabezas como el Problema de la División Instantánea. El punto de vista de Allen es que nada puede ser verdadero en un punto, porque un punto no es una entidad en la cual las cosas suceden o son verdaderas. Sin embargo, por un lado, como demuestra Galton (Galton, 1990) en su análisis crítico de la lógica de intervalos de Allen (Allen, 1984), una teoría de tiempo basada solamente en intervalos es inadecuada para razonar correctamente sobre cambios continuos. Por otra parte, los fenómenos instantáneos existen en el mundo real y por lo tanto, el concepto de punto es necesario para una referencia temporal general. Por ejemplo, considere el siguiente escenario (Ma and Knight, 2003):

Una bola fue lanzada al aire del este al oeste.

Por sentido común, el estado en que la bola estaba en el este y debajo de su cima (máximo) fue seguido inmediatamente por el estado de que la bola estaba en su cima (máximo), y después, fue seguido inmediatamente por el estado de que la bola estaba en el oeste y debajo de su cima. También, el tiempo durante el cual la bola estaba en su máximo —ni al este del máximo ni al oeste del máximo—, debe ser un punto con duración cero, más que cualquier intervalo o momento (Allen and Hayes, 1989), no importa que tan pequeño sea. De hecho, durante el proceso de movimiento de la bola (verticalmente), la velocidad de la bola llega a ser cero solamente cuando la bola está en su máximo.

2.1.3 Sistemas temporales basados en Puntos/Intervalos

Para superar las limitaciones de los enfoques basados solamente en puntos o intervalos conservando sus correspondientes ventajas, se ha introducido un tercer enfoque (Vilain, 1982; Vilain and Kautz, 1986; Galton, 1990; Ma and Knight, 1994), que toma puntos e intervalos como primitivas para hacer referencia temporal a fenómenos instantáneos con duración cero, y a fenómenos periódicos que tienen una cierta duración positiva, respectivamente. Para tener un enfoque generalizado, en este capítulo, tomaremos la teoría de tiempo propuesta en (Ma and Knight, 1994) como base para la descripción temporal, en la cual los puntos y los intervalos se tratan como primitivas: los puntos no tienen que ser definidos como los límites de intervalos y los intervalos no tienen que ser construidos de puntos.

La teoría de tiempo, T, toma una clase no vacía de elementos primitivos de tiempo; una relación primitiva de orden *Meets* ("*se encuentra junto*" en español) sobre los elementos de tiempo, una función *Dur* (es decir, *duración*) entre elementos de tiempo y los números reales no negativos. El sistema básico de axiomas para la tríada (T, Meets, Dur) es la siguiente:

T1. $\forall t_1,t_2,t_3,t_4\ (\text{Meets}(t_1, t_2) \wedge \text{Meets}(t_1, t_3) \wedge \text{Meets}(t_4, t_2) \Rightarrow \text{Meets}(t_4, t_3))$

Es decir, si un elemento de tiempo se encuentra junto con otros dos elementos de tiempo, entonces cualquier elemento de tiempo que se encuentra junto con uno de esos dos elementos también debe encontrarse junto con el primer elemento de tiempo. Este axioma se basa realmente en la intuición de que el "lugar" donde dos elementos de tiempo se encuentran es único y se asocia con los propios elementos de tiempo (Allen and Hayes, 1989).

T2. $\forall t\ \exists t_1,t_2\,(\text{Meets}(t_1, t) \wedge \text{Meets}(t, t_2))$

Es decir, cada elemento de tiempo tiene al menos un precursor inmediato, así como al menos un sucesor inmediato.

T3. $\forall t_1,t_2,t_3,t_4\,(\text{Meets}(t_1, t_2) \wedge \text{Meets}(t_3, t_4) \Rightarrow$
$\text{Meets}(t_1, t_4)\ \nabla\ \exists t'(\text{Meets}(t_1, t') \wedge \text{Meets}(t', t_4))\ \nabla\ \exists t''(\text{Meets}(t_3, t'') \wedge \text{Meets}(t'', t_2)))$

Donde ∇ significa "o exclusivo, XOR". Es decir, cualquier par de lugares de encuentro o son idénticos o existe al menos un elemento de tiempo que se encuentra entre estos lugares de encuentro, si ésos no son idénticos.

T4. $\forall t_1,t_2,t_3,t_4\ (\text{Meets}(t_3, t_1) \wedge \text{Meets}(t_1, t_4) \wedge \text{Meets}(t_3, t_2) \wedge \text{Meets}(t_2, t_4)\ ) \Rightarrow t_1 = t_2)$

Es decir, el elemento de tiempo entre cualquier par de lugares de encuentro es único.

NOTA. En este capítulo, para cualquier par de elementos de tiempo adyacentes, t_1 y t_2 tales que $\text{Meets}(t_1, t_2)$, nosotros utilizaremos la notación $t_1 \oplus t_2$ para denotar su unión ordenada. La existencia de esta unión ordenada de cualquier par de elementos de tiempo adyacentes es garantizada por los axiomas T2 y T3, mientras que su unicidad es garantizada por el axioma T4.

T5. $\forall t_1\ ,\ t_2\ (\text{Meets}(t_1, t_2) \Rightarrow \text{Dur}(t_1) > 0 \vee \text{Dur}(t_2) > 0)$

Es decir, elementos con duración cero no pueden estar juntos (encontrarse).

T6. $\forall t_1\ t_2\ (\text{Meets}(t_1, t_2) \Rightarrow\ \text{Dur}(t_1 \oplus t_2) = \text{Dur}(t_1) + \text{Dur}(t_2))$

Es decir, la operación de "unión ordenada" sobre los elementos de tiempo es consistente con la operación convencional de "adición" sobre la función de asignación de duración, es decir, Dur.

NOTA. En la teoría de tiempo *T* introducida aquí, adoptamos los siguientes resultados de la teoría de los números reales:

(r1) El conjunto de los números reales es ordenado totalmente por la relación "≤" (menor-que-o-igual-a), y donde ">" es la relación "mayor-que", que no es (≤).

(r2) "+" es el operador de la adición convencional sobre los números reales (no negativos).

En términos de la relación *Meets* (*Se encuentra junto*), otras relaciones de orden exclusiva sobre elementos de tiempo pueden ser derivados como se muestra abajo:

$\text{Equal}(t_1, t_2) \Leftrightarrow \exists t', t''(\text{Meets}(t', t_1) \wedge \text{Meets}(t', t_2) \wedge \text{Meets}(t_1, t'') \wedge \text{Meets}(t_2, t''))$
$\text{Before}(t_1, t_2) \Leftrightarrow \exists t(\text{Meets}(t_1, t) \wedge \text{Meets}(t, t_2))$
$\text{Overlaps}(t_1, t_2) \Leftrightarrow \exists t, t_3, t_4(t_1 = t_3 \oplus t \wedge t_2 = t \oplus t_4)$
$\text{Starts}(t_1, t_2) \Leftrightarrow \exists t(t_2 = t_1 \oplus t)$
$\text{During}(t_1, t_2) \Leftrightarrow \exists t_3, t_4(t_2 = t_3 \oplus t_1 \oplus t_4)$
$\text{Finishes}(t_1, t_2) \Leftrightarrow \exists t(t_2 = t \oplus t_1)$
$\text{After}(t_1, t_2) \Leftrightarrow \text{Before}(t_2, t_1)$
$\text{Overlapped-by}(t_1, t_2) \Leftrightarrow \text{Overlaps}(t_2, t_1)$
$\text{Started-by}(t_1, t_2) \Leftrightarrow \text{Starts}(t_2, t_1)$
$\text{Contains}(t_1, t_2) \Leftrightarrow \text{During}(t_2, t_1)$
$\text{Finished-by}(t_1, t_2) \Leftrightarrow \text{Finishes}(t_2, t_1)$,
$\text{Met-by}(t_1, t_2) \Leftrightarrow \text{Meets}(t_2, t_1)$

Por un lado, el sistema completo de las trece relaciones de orden exclusiva posibles (las doce relaciones más la relación *Meets*) entre cualquier par de elementos de tiempo se puede caracterizar simplemente por un solo axioma:

$\forall t_1, t_2(\text{Equal}(t_1, t_2) \vee \text{Before}(t_1, t_2) \vee \text{After}(t_1, t_2) \vee \text{Meets}(t_1, t_2)$
$\vee \text{Met-by}(t_1, t_2) \vee \text{Overlaps}(t_1, t_2) \vee \text{Overlapped-by}(t_1, t_2) \vee \text{Starts}(t_1, t_2)$
$\vee \text{Started-by}(t_1, t_2) \vee \text{During}(t_1, t_2) \vee \text{Contains}(t_1, t_2) \vee \text{Finishes}(t_1, t_2)$
$\vee \text{Finished-by}(t_1, t_2))$

Por otro lado, la exclusividad de estas trece relaciones de orden necesita ser caracterizado por 78 axiomas (${}^2C_{13} = 13!\ /2!\ 11!$) de la siguiente forma:

$\forall t_1, t_2(\neg\text{Relation1}(t_1, t_2) \vee \neg\text{Relation2}(t_1, t_2))$

donde *Relation1* y *Relation2* son dos relaciones distintas de las trece relaciones mencionadas.

Para conveniencia de la expresión, añadiremos la relación no-exclusiva de Allen "*In*" ("*en*" en español), que se define solamente para los intervalos (Allen, 1984). Para manejar intervalos y puntos de tiempo, introduciremos otra relación temporal, "Part" (*parte* en español), como se define abajo:

$\text{In}(t1, t2) \Leftrightarrow \text{Starts}(t1, t2) \vee \text{During}(t1, t2) \vee \text{Finishes}(t1, t2)$

$\text{Part}(t1, t2) \Leftrightarrow \text{Equal}(t1, t2) \vee \text{In}(t_1, t_2)$

2.2 Glosario ontológico de los elementos temporales

De manera similar a los conceptos y a las terminologías convencionales de la teoría de los números reales, basados en la teoría de tiempo T caracterizada más arriba, presentamos un glosario ontológico referente a los elementos de tiempo:

G1. El elemento de tiempo t es un punto, si Dur(t) = 0.

G2. El elemento de tiempo t es un intervalo, si Dur(t) $>$ 0.

G3. El intervalo i es partido, si existen dos elementos de tiempo t_1 y t_2 tales que $i = t_1 \oplus t_2$; de otra forma, el intervalo i es no-partible, es decir, un momento (Allen and Hayes, 1989).

G4. El intervalo i es un subintervalo del intervalo j, si la Part(i, j).

G5. El intervalo i es un subintervalo propio del intervalo j, si In(i, j).

G6. El intervalo i es un subintervalo propio estricto del intervalo j, si In(i, j) $\wedge$ Dur(i) $<$ Dur(j).

G7. El punto p es un punto interno del intervalo i, si In(p, i).

G8. El punto p es un punto interno del intervalo i, si During(p, i).

G9. El punto p es el punto final izquierdo del intervalo i, si Starts(p, i).

G10. El punto p es el punto final derecho del intervalo i, si Finishes(p, i).

G11. El punto p es el punto de conexión izquierda del intervalo i, si Starts(p, i) $\vee$ Meets(p, i).

G12. El punto p es el punto de conexión derecha del intervalo i, si Finishes(p, i) $\vee$ Met-by(i, p).

G13. El intervalo i es abierto por la izquierda en el punto p, si Meets(p, i).

G14. El intervalo i es abierto por la derecha en el punto p, si Meets(i, p).

G15. El intervalo i es cerrado por la izquierda en el punto p, si Starts(p, i).

G16. El intervalo i es cerrado por la derecha en el punto p, si Finishes(p, i).

Por definición, un punto interno de un intervalo dado es el punto final izquierdo o el punto del final derecho, o el punto interno del intervalo. Además, un intervalo es cerrado por la izquierda en un punto, si y solamente si, este punto es el punto final izquierdo de ese intervalo. Similarmente, un intervalo es cerrado por la derecha en un punto, si y solamente si, este punto es el punto final derecho de ese intervalo.

Es importante observar que, un punto final izquierdo de un intervalo es también el punto de conexión izquierda de ese intervalo; un punto de conexión

izquierda no es necesariamente el punto final izquierdo. De manera semejante, mientras que un punto final derecho de un intervalo es también el punto de conexión derecha de ese intervalo; un punto de conexión derecha no es necesariamente el punto final derecho.

2.3 Clasificación de relaciones de orden temporal

Cuando un par de elementos de tiempo, t_1 y t_2, se especifican como un punto y un punto, un punto y un intervalo, un intervalo y un punto, y un intervalo y un intervalo, entonces, respectivamente, todas las relaciones de orden exclusiva entre t_1 y t_2 se puede clasificar en los cuatro grupos siguientes, a los que llamaremos Restricción de la Relación Temporal (*Temporal Relation Constraint*, *TRC*):

- Relaciones de orden que relacionan un punto con un punto: {*Equal, Before, After*}
- Relaciones de orden que relacionan un intervalo con un intervalo: {*Equal, Before, After, Meets, Met-by, Overlaps, Overlapped-by, Starts, Started-by, During, Contains, Finishes, Finished-by*}
- Relaciones de orden que relacionan un punto con un intervalo: {*Before, After, Meets, Met-by, Starts, During, Finishes*}
- Relaciones de orden que relacionan un intervalo con un punto: {*Before, After, Meets, Met-by, Started-by, Contains, Finished-by*}

Está claro que estos cuatro grupos de relaciones temporales cualitativas son similares a los introducidos en (Vilain, 1982; Vilain and Kautz, 1986; Terenziani and Torasso, 1995). La única diferencia es que en la teoría T del tiempo descrita aquí, al relacionar un punto con un intervalo (o un intervalo con un punto), las relaciones temporales "*Meet*" y "*Met-by*" son distintas de las relaciones temporales "*Starts*" y "*Finishes*" (o "*Started-by*" y "*Finished-by*"), respectivamente. Este tipo de distinciones no se presenta en otras teorías de tiempo. De hecho, en los sistemas del tiempo propuestas por Vilain (1982), Vilain and Kautz (1986), y Terenziani and Torasso (1995), aunque un punto puede "comenzar (*starts*)"/"terminar (*finishes*)" un intervalo y un intervalo puede ser "empezado-por (started-by)"/"terminado-por (finished-by)" un punto, las relaciones "*Meet*" y "*Met-by*" no se definen como relaciones temporales válidas entre un punto y un intervalo, o entre un intervalo y un punto.

Sin embargo, es importante comentar que la distinción entre la proposición "punto p se encuentra junto con (*meets*) el intervalo i", es decir, *Meets*(p, i), y la proposición "punto p comienza (*starts*) el intervalo i", es decir, *Starts*(p, i), es extremadamente sutil. Antes que nada, las proposiciones son mutuamente exclusivas, es decir, si se cumple *Meets*(p, i) entonces *Starts*(p, i) no se cumple. De hecho, por definición, *Meets*(p, i) significa que el punto p es uno

de los precursores inmediatos del intervalo i, es decir, que el punto p es "anterior" que i, y no hay otros elementos de tiempo que se encuentre entre p e i; y *Starts*(p, i) indica que el punto p es una parte que empieza el intervalo i (realmente, en este caso, es el punto de terminación/principio por la izquierda). Por lo que, *Meets*(p, i) implica que el intervalo i no incluye el punto p, mientras que *Starts*(p, i) implica que el intervalo i incluye este punto. En otras palabras, *Meets*(p, i) implica que el intervalo i es abierto por la izquierda en el punto p, mientras que *Starts*(p, i) implica que el intervalo i es cerrado por la izquierda en el punto p. Además, de *Meets*(p, i), podemos formar un nuevo intervalo como una unión ordenada de p y de i, es decir, $p \oplus i$. Vale la pena precisar que el intervalo i y el intervalo $p \oplus i$ son dos intervalos distintos, aunque $Dur\ (p \oplus i) = Dur\ (p) + Dur\ (i) = Dur\ (i)$ ya que $Dur\ (p) = 0$. Aun más, tenemos aquí una relación importante:

$$Meets(p, i) \Leftrightarrow Starts(p, p \oplus i).$$

Es decir, el intervalo i es abierto por la izquierda en el punto p si y solamente si el intervalo $p \oplus i$ es cerrado por la izquierda en el punto p. Las mismas consideraciones aplican a la relación entre *Finishes*(p, i) y *Meets*(i, p).

Según lo mencionado anteriormente, ya que la teoría de tiempo presentada aquí trata a los puntos y los intervalos como primitivas de manera similar, los puntos no tienen que ser definidos como límites de los intervalos y, del mismo modo, los intervalos no tienen que ser construidos usando puntos. De hecho, los intervalos pueden encontrarse sin ningún punto entre ellos, dentro de ellos, o limitándolos. Por ejemplo, podemos tener el conocimiento temporal siguiente:

$$\text{Meets}(i_1, i_2) \wedge \text{Meets}(i_1, i_3) \wedge \text{Meets}(i_2, i_4)$$

Eso quiere decir que el intervalo i_1 "se encuentra junto con" los intervalos i_2 e i_3, y el intervalo i_2 "se encuentra con" el intervalo i_4, sin la necesidad de tener conocimiento sobre el punto cualquiera.

Sin embargo, la teoría de tiempo permite los casos donde los puntos pueden encontrarse en medio, estar dentro, o comenzar/terminar los intervalos. Esta clase de conocimiento se puede expresar en términos de tales relaciones como "*Meets*", "*Starts*", "*During*", etc. Por ejemplo:

$$\text{Meets}(i_5, p) \wedge \text{Meets}(p, i_6) \wedge \text{Starts}(p, i_7)$$

Lo que quiere decir que el intervalo i_5 "se encuentra junto con (*meets*)" el punto p, que en su turno "se encuentra junto con (*meets*)" el intervalo i_6 y "comienza (*starts*)" el intervalo i_7. Basándose en estos hechos, si denotamos la unión ordenada de los tres elementos de tiempo adyacentes i_5, p e i_6 como i, es decir, $i = i_5 \oplus p \oplus i_6$, entonces tenemos *During*(p, i). En otras palabras, podemos decir que el punto p se puede considerarse "dentro" del intervalo i (es decir, p es un punto interno del intervalo i). Además, podemos deducir que el

intervalo i_5 es abierto por la derecha en p, i_6 es abierto por la izquierda en p, y el intervalo i_7 es cerrado por la izquierda en p.

3. Enfoques temporales

En las últimas tres décadas, las lógicas temporales se han reconocido por tener importancia en una variedad de áreas en el dominio de la Inteligencia Artificial. Según la clasificación de Shoham (1987) y Vila (1994),y hablando de manera general, hay tres maneras principales para introducir el concepto de tiempo en lógica:

(1) lógicas de primer orden con parámetros temporales,
(2) lógicas temporales modales, y
(3) lógicas temporales materializadas (*reified*).

3.1 Método que utiliza parámetros temporales

El método de los parámetros temporales (TA = *temporal arguments*) (Haugh, 1987) simplemente incluye tiempos como parámetros adicionales a las funciones y a los predicados del lenguaje de primer orden. Por ejemplo, usando el método de TA, se puede representar el hecho por el cual "Luz_5 es encendido en el momento t": $ON(Luz\_5, t\neg)$. TA es una manera natural (y puede ser la más simple) de introducir tiempo en un formalismo lógico. De hecho, como el tratamiento clásico del tiempo en matemáticas y física, tal enfoque de primer orden fue introducido originalmente en filosofía de Russell (1903) al principio del siglo XX. En Inteligencia Artificial, el cálculo situacional (*Situation Calculus*) de McCarthy y Hayes (1969), y la lógica temporal no materializada de Bacchus *et al.* (1991) son ejemplos de sistemas que utilizan el método de argumentos temporales. El defecto principal del método de TA es que, como el tiempo está representado apenas como argumento(s) de los predicados, no se introduce ningún estatus especial para el tiempo: ni conceptual, ni notacional. Por lo tanto, no es suficientemente expresivo hablar de las generalidades del aspecto temporal de aserciones. Por ejemplo, usando el método de TA, no se puede expresar conocimiento de sentido común tal como "los efectos no pueden preceder a sus causas" (Shoham, 1987).

3.2 Lógicas temporales modales

Las lógicas temporales modales (MTL) es un enfoque *relativista* hacia el tiempo al contrario con el enfoque absolutista de TA (Vila, 1994). En el contexto temporal, las lógicas temporales modales reinterpretan

semánticamente los mundos posibles clásicos (Kripke, 1963), haciendo que cada mundo posible represente un tiempo diferente. El concepto de tiempo se acomoda al extender el cálculo proposicional o cálculo de predicados para incluir operadores temporales modales tales como $F\phi$, $P\phi$, $G\phi$ y $H\phi$, representando en la fórmula ϕ "será verdad", "era verdad", "será siempre verdad", y "era siempre verdad", respectivamente. *MTL* también se llama la lógica temporal en filosofía (Rescher and Urquhart, 1971; McArthur, 1976), y fue construida inicialmente en los principios de los años 50 por Prior (1955). La primera aplicación de *MTL* en razonamiento sobre programas aparece en el sistema de Pnueli (Pnueli, 1977). En Gabbay (1989), se presenta una lógica temporal ejecutable llamada la lógica *US*, la cual está compuesta por una lógica clásica de primer orden con la adición de operadores modales *desde* y *hasta*, junto con un operador de punto fijo. La lógica se basa en los números naturales que representan el flujo del tiempo y se puede utilizar para la especificación y el control del comportamiento de procesos en el tiempo. Se muestra que la lógica *US* es completamente expresiva para un modelo histórico de los datos, en el mismo sentido que la lógica de primer orden es para un modelo no-temporal de los datos. Como un hecho, la mayoría de las lógicas temporales modales interpretan fórmulas sobre los puntos temporales. La única excepción que aparece en la literatura es la lógica modal proposicional de Halpern y Shoham (1991) que se basan en intervalos del tiempo.

3.3 Lógicas temporales materializadas

Como enfoque alternativo, las lógicas temporales materializadas (*reified*, en inglés) interpretan proposiciones estándar de algún lenguaje inicial (por ejemplo, la lógica de primer orden clásica o la lógica modal) como objetos que denotan términos proposicionales. Los sistemas representativos que incluyen las proposiciones substanciales son de McDermott (1982), de Allen (1984), de Shoham (1987), de Reichgelt (1989), de Galton (1990), y de Ma y Knight (1996). En lógica temporal substancial, los términos proposicionales se relacionan con los objetos temporales u otros términos proposicionales a través de una clase adicional de "meta-predicados" (Ma and Knight, 1996) (o "predicados globales", como los llamó Shoham (1987) una vez), tales como SE_CUMPLE (o VERDAD), OCURRE, CAUSA, etc. Por ejemplo, en una lógica materializada, se puede utilizar SE_CUMPLE(*01/09/1982*, VIVO (*pavo*)) representar la aserción "El pavo está vivo el 1ro de septiembre del 1982", mientras que al usar el método de argumentos temporales, se puede expresar lo mismo como VIVO(*pavo*, *01/09/1982*), o al usar el estilo de la lógica temporal modal: *M, 01/09/1982* VIVO(*pavo*), donde *M* es la interpretación en cuestión.

3.4 Lógicas materializadas vs. lógicas no-materializadas

En los últimos veinticinco años, las lógicas materializadas fueron aceptadas ampliamente en el dominio de Inteligencia Artificial. Sin embargo, también se usan otros enfoques que intentan sobreponerse a algunas deficiencias de la lógica materializada temporal. De hecho, los méritos relativos de las lógicas materializadas vs. las lógicas no-materializadas (*reified* vs. *non-reified*, en inglés) dependerán de la aplicación particular.

3.4.1 Teoría de Bacchus *et al.*

En la investigación publicada en la revista *AI journal*, Bacchus *et al.* (1991) proponen una lógica temporal no-materializada, denotada como *BTK*, siguiendo las ideas del método de argumentos temporales. Se observa que, desde el punto de vista práctico, tenemos menos experiencia con el razonamiento automatizado en formalismos materializados. Además, después de comparar su enfoque no-materializado con algunas teorías materializadas propuestas previamente incluyendo las de Allen (1984), de Shoham (1987) y de Lifschitz (1987), Bacchus *et al.* afirman que las lógicas temporales materializadas tienen la desventaja de ser más complejas al expresar aserciones sobre algunos objetos dados con respecto a diversos periodos de tiempo. Por ejemplo, en *BTK*, se puede representar la aserción "el presidente de los E.E.U.U. en 1962 murió en 1963" simplemente como:

MORIR(1963, PRESIDENTE (1962, E.E.U.U.))

donde los símbolos *PRESIDENTE* y MORIR se tratan respectivamente como el símbolo de la función y el símbolo del predicado en *BTK*.

Por el contrario, para expresar tal aserción en lógica de Shoham (1987), es necesario recurrir al uso más incómodo de la cuantificación:

$\forall x$(VERDADERO(1962, 1963, PRESIDENTE(E.E.U.U.) = x)
$\Rightarrow$ VERDADERO(1963, 1964, MORIR(x)))

Esto se aplica también a la lógica substancial de intervalos de Allen (1984):

$\forall x$(HOLDS(ES(PRESIDENTE(E.E.U.U.), x)), 1962)
$\Rightarrow$ OCCURING(MORIR(x), 1963))

NOTA. En éste, sería más intuitivo si utilizamos "*HOLDS_IN* (*SE_CUMPLE_EN* en español)" de Galton (Galton 1990) en vez de "*OCCURING* (*OCURRE* en español)" de Allen.

También, Bacchus *et al.*, argumentan que para las teorías temporales materializadas tales como la de Allen, si permitimos la lógica materializada de fórmulas no-atómicas, las cosas pueden volverse muy complicadas. De hecho, necesitaríamos axiomas para especificar la igualdad de ciertos términos proposicionales obviamente iguales, por ejemplo, axiomas como:

$\forall$*pro1,pro2(*AND*(pro1, pro2)* $\equiv$ AND*(pro2, pro1))*

Sin embargo, como señalan Bacchus *et al.* (1991), los conectores lógicos se pueden aplicar fuera del predicado HOLDS (SE_CUMPLE), puesto que la mayoría de las aplicaciones no requieren la lógica substancial más que las proposiciones atómicas. Por ejemplo, en vez de escribir

HOLDS(AND(*pro1*, *pro2*), *t*)

podemos escribir

HOLDS(*pro1*, *t*) $\wedge$ HOLDS(*pro2*, *t*)

Sin embargo, es interesante tomar esto en cuenta, y éste es el enfoque tomado por Allen. Véase (H.3) en (Allen, 1984).

3.4.2 Argumentos de Galton

En su artículo presentado en *IJCAI-91*, Galton (1991) argumenta que la lógica materializada temporal es filosóficamente sospechosa y técnicamente innecesaria. Especialmente, él examina las teorías materializadas de Allen (Allen, 1984) y de Shoham (Shoham, 1987), y demuestra cómo se puede desmaterializar. De hecho, una manera no-materializada para las fórmulas de Allen como, por ejemplo:

OCCUR (KISS (Juan, Maria), $\text{Time}_{\text{KISS}}$)

será

KISS (Juan, Maria, $\text{Time}_{\text{KISS}}$)

Galton afirma que esta versión no-materializada tiene la ventaja de mayor simplicidad y naturalidad, sin embargo, hay que pagar un cierto precio, a saber, que para expresar las proposiciones causales de Allen en el reemplazo no-materializado como el ejemplo de arriba, hay que emplear los mecanismos sintácticos que van más allá de los límites de la lógica de primer orden. Esto, seguramente es un costo inaceptable de lo no-materializado.

Para resolver este problema, alternativamente, adoptando los mecanismos de Davidson (1967; 1969) de lógica materializada del evento de tipo en lugar de lógica materializada de evento ejemplar de tipo, Galton propone un procedimiento para convertir la lógica materializada de tipos en lógica materializada de ejemplos de tipos, según lo ejemplificado por parte del cálculo de eventos de Kowalski y Sergot (1986). La idea es simplemente introducir un tipo adicional de elementos, es decir, ejemplos de tipos, en el lenguaje de primer orden para relacionar expresiones no-temporales con los intervalos de tiempo particulares. Por ejemplo, OCCUR(KISS(Juan, Maria), $\text{Time}_{\text{KISS}}$) puede ser reescrito como:

$\exists e$(KISS(Juan, Maria, e) $\wedge$ OCCUR(e, $\text{Time}_{\text{KISS}}$))

Especialmente, Galton afirma que con el mecanismo de lógica materializada de ejemplos de tipo en lugar de la lógica materializada de tipo, el predicado ECAUSE de Allen se puede expresar dentro de los límites de la lógica de primer orden. De hecho, donde Allen escribe ECAUSE (e_1, t_1, e_2, t_2), Galton usa esto:

$E(e_1) \wedge E'(e_2) \wedge \text{ECAUSE}(e_1, e_2)$

Aquí, es importante precisar que, e_1 y e_2 en ECAUSE (e_1, t_1, e_2, t_2) de Allen denotan realmente los tipos de eventos, y por lo tanto corresponden a los tipos de eventos E y E', más bien que a los ejemplos de tipos de eventos e_1 y e_2 en $E(e_1) \wedge E'(e_2) \wedge \text{ECAUSE}(e_1, e_2)$ de Galton. Mientras que Allen introduce intervalos de tiempo como argumentos de ECAUSE para asegurar referencia a las ocurrencias individuales de sus tipos de eventos, la relación causal de Galton se cumple directamente entre los ejemplos de eventos. Según observa el mismo Galton, una conexión causal entre el ejemplo de evento depende generalmente de la existencia de una cierta ley causal general que relaciona sus tipos respectivos. Sin embargo, esta observación sobre causalidad no se manifiesta claramente en el trabajo de Galton. De hecho, insistiendo en definir ECAUSE como relación entre los ejemplos de eventos, ECAUSE(e_1, e_2) es tratado por Galton como abreviatura para cualquier fórmula de la forma siguiente:

$E(e_1) \wedge E'(e_2) \wedge \text{OCCUR}(e_1, i_1) \wedge \text{OCCUR}(e_2, i_1{+}1)$
$\wedge \forall i \forall e\ (E(e) \wedge \text{OCCUR}(e, i)$
$\Rightarrow \exists e'(E'(e') \wedge \text{OCCUR}(e', i_1{+}1)))$

Sin embargo, incluso con esta interpretación, todavía no está claro cómo realizar la inferencia causal, por ejemplo "si ocurrió un evento que ha causado otro evento, entonces también ocurre el evento causado", no está claro cómo expresar aserciones de sentido común como "los hechos nunca preceden a sus causas". Es importante mencionar que ambos casos, se pueden expresar convenientemente en lógicas materializadas. Por ejemplo, véase los axiomas (O.4) y (O.5) en (Allen, 1984).

3.4.3 Argumentos a favor de la materialización temporal

Como se argumenta en (Vila, 1994) y en (Ma and Knight, 2001), una lógica temporal completamente materializada permite construir proposiciones y cuantificar sobre los términos de proposiciones, y además asociarlos con los tiempos particulares. También, se puede relacionar un tiempo dado con otros tiempos. Por lo tanto, desde el punto de vista de la expresividad, el enfoque materializado temporal tiene claras ventajas al hablar de algunos aspectos importantes que sería difícil (si no imposible) de expresar en otros enfoques. De hecho:

En primer lugar, en lógicas temporales materializadas tales como la de Ma y Knight (1996), una distinción entre los objetos temporales y las construcciones lógicas no-temporales se hace explícitamente. Esta distinción permite asignar al tiempo un estatus especial. Especialmente, a diferencia de los términos no-temporales, los términos temporales se pueden relacionar con otros términos temporales, por ejemplo, ANTES(T, T'), así como a términos proposicionales, por ejemplo, HOLDS(1997, AMA(María, Juan)).

En segundo lugar, como demostraremos en las secciones 4 y 5, en las lógicas materializadas, se puede diversificar varios meta-predicados para proposiciones con valores booleanos a los elementos de tiempo e inventar diversos tipos de proposiciones temporales. Por lo que, las lógicas proporcionan el poder expresivo de hablar de las generalidades de los aspectos temporales de aserciones estándares. Por ejemplo, considere las dos aserciones no-temporales siguientes:

El arma está cargada.

y

El pavo está vivo.

Intuitivamente, cuando asociamos estas aserciones con tiempos, éstos deben satisfacer el requisito de la homogeneidad. Es decir:

- El arma está cargada en el tiempo T_1 si y solamente si está cargada en cada parte de T_1;
- El pavo está vivo en el tiempo T_2 si y solamente si el pavo está vivo en cada parte de T_2.

Por ejemplo, en términos de lógica materializada basada en intervalos de Allen, se usa la fórmula HOLDS(*t, pro*) para expresar que la aserción *pro* se cumple durante todo el periodo del tiempo *t*, donde la calidad de la *homogeneidad* se caracteriza por medio del axioma siguiente:

$$\text{HOLDS}\ (t, pro) \Leftrightarrow \forall t'(\text{PARTE}(t', t) \Rightarrow \text{HOLDS}\ (t', pro))$$

Por lo tanto, podemos expresar "*El arma está cargada el lunes*" y "*el pavo está vivo el lunes*" simplemente por:

HOLDS (Lunes, CARGADA(arma))

y

HOLDS (Lunes, VIVO(pavo))

NOTA. Aquí, por razones de simplicidad, asumimos que el término temporal *lunes* se mapea a un día particular, aunque la palabra "*lunes*" refiere normalmente a todos los días de cierta clase (Galton, 1991).

En la mayoría de las lógicas de no- materializadas tales como *BTK* de Bacchus *et al.* (1991), este conocimiento se podría expresar como:

CARGADA (*lunes, arma*)

y

VIVO (*lunes, pavo*)

Sin embargo, para expresar la homogeneidad de estas dos aserciones, se tendría que agregar dos axiomas, respectivamente:

$$\text{CARGADA}(t, Arma) \Leftrightarrow \forall t'(\text{Parte}(t', t) \Rightarrow \text{CARGADA}(t', Arma))$$

$$\text{VIVO}(t, Pavo) \Leftrightarrow \forall t'(\text{Parte}(t', t) \Rightarrow \text{VIVO}(t', Pavo))$$

Generalmente hablando, en las lógicas no- materializadas, es imposible aplicar los meta-predicados generalmente caracterizados (por ejemplo, SE_CUMPLE) a las aserciones con algunas propiedades comunes bien definidas. De hecho, para expresar que algunas aserciones satisfacen ciertos tipos de propiedades (es decir, el requisito de la homogeneidad) en lógicas de no- materializadas, se tiene que utilizar el esquema similar de axiomas para cada aserción de este tipo por separado. Esto es de hecho una desventaja importante de las lógicas de no- materializadas en comparación con las materializadas, especialmente en casos donde el número de aserciones con algunas características comunes es muy grande.

Finalmente, como demostraremos abajo en la sección 6.3, las lógicas temporales materializadas proporcionan los medios convenientes y de gran poder de expresión para representar y razonar sobre eventos, cambios y relaciones causales.

4. Amplia gama de meta predicados

En la lógica estándar de predicados, una afirmación (o proposición) es una oración con valor booleano que puede ser verdadero o falso. Sin embargo, como pasa el tiempo, el mundo puede cambiar su estado de uno a otro, causado por ciertos eventos o procesos que ocurran en un tiempo dado. Un enfoque natural a la representación y razonamiento sobre estas afirmaciones dependientes del tiempo (que llamaremos proposiciones temporales en este capítulo), incluyendo propiedades, hechos, estados, eventos, procesos y todo lo demás que tenga valores booleanos que son dependientes del tiempo, es asociarlos con los elementos de tiempo (es decir, los puntos instantáneos e intervalos de duración).

Un asunto interesante con respecto a la relación de proposiciones temporales con los elementos de tiempo es que podemos desear expresar no sólo que una proposición sea verdadera en un intervalo del tiempo, sino también que una proposición es verdadera solamente en algunas partes de un intervalo; por ejemplo, sobre algunos de sus subintervalos, en algunos de sus

puntos internos o en los puntos finales —si los hay—, y así sucesivamente. En lo que sigue en este capítulo, nosotros utilizaremos "*HOLDS*" (*SE_CUMPLE* en español) como el meta-predicado primitivo para atribuir a las proposiciones los elementos de tiempo, de modo que para cada par que consiste de una proposición *f* y un elemento de tiempo *t*, la fórmula *HOLDS*(*f, t*) representa que la proposición *f* es verdadera con relación al elemento de tiempo *t*. En términos de "*HOLDS*" (*SE_CUMPLE*), definimos otros meta-predicados derivados como sigue:

- Meta-predicado *HoldsAT* (*SE_CUMPLE_EN_PUNTO*) atribuye a una proposición un punto, afirmando que la proposición es verdadera "*en*" ese punto:

H1. $\text{HoldsAT}(f, t) \Leftrightarrow \text{Dur}(t) = 0 \wedge \text{Holds}(f, t)$

Por ejemplo: *El cohete que había sido lanzado al aire estuvo en su cima.*

- Meta-predicado *HoldsON* (*SE_CUMPLE_EN_INTERVALO*) atribuye a una proposición un intervalo, indicando que la proposición es verdadera con respecto a todas las partes de ese intervalo, incluyendo todos sus sub intervalos propios, así como todos sus puntos internos (si los hay). Es decir, *HoldsON*(*f, t*) indica que la proposición *f* es verdadera "*en todo*" el intervalo *t*:

H2. $\text{HoldsON}(f, t) \Rightarrow \forall t_1(\text{In}(t_1, t) \Rightarrow \text{Holds}(f, t_1))$

H3. $\forall t_1(\text{Part}(t_1, t) \Rightarrow \exists t_2(\text{Part}(t_2, t_1) \wedge \text{Holds}(f, t_2))) \Rightarrow \text{HoldsON}(f, t)$

Por ejemplo: *El cohete estuvo inmóvil.*

NOTA. A diferencia del enfoque de Allen en el cual la condición suficiente y necesaria para su meta-predicado "*HOLDS* (*SE_CUMPLE*)" es la misma (Allen, 1984), en este capítulo utilizamos dos axiomas, H2 y H3, para caracterizar nuestro meta-predicado *HoldsON*. La razón de esto es tomar en cuenta el caso donde está presente un momento del tiempo que no se descompone o un punto (véase abajo). Es fácil observar que el inverso de H3 se implica por H2.

- Meta-predicado $HoldsON_P$ (*SE_CUMPLE_EN_PUNTOS_DE_INTERVALO*) atribuye a una proposición un intervalo, indicando que la proposición es verdadera en todos sus puntos internos (si los hay):

H4. . $\text{HoldsON}_P(f, t) \Leftrightarrow \forall t_1(\text{Dur}(t_1) = 0 \wedge \text{In}(t_1, t) \Rightarrow \text{HoldsAT}(f, t_1))$

Por ejemplo: *El cohete, que se había estado moviendo continuamente en la misma dirección, no logró el cambio a su posición dentro de un solo instante.*

NOTA. Aquí, $HoldsON_P$ representa la idea de una proposición que es verdadera en un punto o puntos sin ser verdadera en cualquier intervalo que contiene o que es limitado por los puntos.

- Meta-predicado $HoldsON_I$ (*SE_CUMPLE_EN_SUBINTERVALOS*) atribuye a una proposición un intervalo, indicando que la proposición es verdadera en todos sus sub intervalos:

H5. $\text{HoldsON}_\text{I}(f, t) \Rightarrow \forall t'(\text{Dur}(t_1) > 0 \wedge \text{In}(t_1, t) \Rightarrow \text{Holds}(f, t_1))$

H6. $\forall t_1(\text{Dur}(t_1) > 0 \wedge \text{Part}(t_1, t) \Rightarrow \exists t_2(\text{Dur}(t_2) > 0 \wedge \text{Part}(t_2, t_1) \wedge \text{Holds}(f, t_2)))$
$\Rightarrow \text{HoldsON}_\text{I}(f, t)$

Por ejemplo: *El cohete, que se había estado moviendo continuamente en la misma dirección, logró el cambio de su posición.*

NOTA. El meta-predicado $HoldsON_I$ definido aquí es de hecho muy similar a "*Holds* (SE_CUMPLE)" de Allen (Allen, 1984) salvo que $HoldsON_I$ se puede atribuir a los intervalos que se descomponen y a los momentos del tiempo que no se descomponen, mientras que "*Holds*" de Allen es solamente aplicable a los intervalos que se descomponen pero no a los momentos de tiempo (Ma and Knight, 1994).

- Meta-predicado *HoldsIN* (*SE_CUMPLE_EN_PARTE_DE_INTERVALO*) atribuye a una proposición un intervalo, indicando que la proposición es verdadera con respecto a por lo menos a un sub intervalo propio, o por lo menos a un punto interno de ese intervalo:

H7. $\text{HoldsIN}(f, t) \Leftrightarrow \exists t_1(\text{In}(t_1, t) \Rightarrow \text{Holds}(f, t_1))$

Por ejemplo: *El cohete era inmóvil.*

- Meta-predicado $HoldsIN_P$ (*SE_CUMPLE_EN_PUNTO_DE_INTERVALO*) atribuye a una proposición un intervalo, indicando que hay por lo menos un punto interno de ese intervalo con respecto al cual la proposición es verdadera:

H8. $\text{HoldsIN}_\text{P}(f, t) \Leftrightarrow \exists t_1(\text{Dur}(t_1) = 0 \wedge \text{In}(t_1, t) \Rightarrow \text{HoldsAT}(f, t_1))$

Por ejemplo: *El cohete alcanzó su cima.*

- Meta-predicado $HoldsIN_I$ (*SE_CUMPLE_EN_SUBINTERVALO*) atribuye a una proposición un intervalo, indicando que la proposición es verdadera con respecto a por lo menos a un subintervalo propio de ese intervalo:

H9. $\text{HoldsIN}_{\text{I}}(f, t) \Leftrightarrow \exists t'(\text{Dur}\ (t_1) > 0 \wedge \text{In}(t_1, t) \Rightarrow \text{Holds}(f, t_1))$

Por ejemplo: *El cohete logró un cambio a su posición.*

Es fácil observar que, para cualquier intervalo *t*:

$\text{HoldsON}(f, t) \Leftrightarrow \text{HoldsON}_{\text{P}}(f, t) \wedge \text{HoldsON}_{\text{I}}(f, t)$
$\text{HoldsIN}_{\text{P}}(f, t) \Rightarrow \text{HoldsIN}(f, t)$
$\text{HoldsIN}_{\text{I}}(f, t) \Rightarrow \text{HoldsIN}(f, t)$

También, si el intervalo *t* se puede descomponer, tenemos:

$\text{HoldsON}_{\text{I}}(f, t) \Rightarrow \text{HoldsIN}_{\text{I}}(f, t)$

Sin embargo, los intervalos pueden existir aunque no tengan ningún punto interno, en general:

$\text{HoldsON}_{\text{P}}(f, t) =/=> \text{HoldsIN}_{\text{P}}(f, t)$

Siguiendo la notación de Galton (Galton, 1990), utilizaremos al operador "*NOT" (NO* en el español) para la negación de proposiciones, que sea distinto de la negación de los meta-predicados en cuyo caso se usará "¬". Intuitivamente, NOT(*f*) es solamente verdadera para los elementos del tiempo para los cuales *f* no es verdadera. Introduciremos el axioma siguiente para caracterizar la relación entre la negación de una proposición y la negación del meta-predicado utilizado:

H10. $\text{Holds}(\text{not}(f), t) \Leftrightarrow \forall t_1(\text{Part}(t_1, t) \Rightarrow \neg\text{Holds}(f, t_1))$

Por ejemplo: La negación de la proposición "*la luz está encendida*", es decir, "*la luz no está encendida*", es verdadera con respecto al tiempo *t* si y solamente si la proposición "*la luz está encendida*" no es verdadera, con respecto a cualquier parte de *t*.

5. Clasificación exhaustiva de proposiciones temporales

En literatura, se ha mostrado que los aspectos temporales de varias clases de proposiciones son muy diversos y complicados. De hecho, según el sentido común, tenemos lo siguiente:

- Por un lado, existen proposiciones que no se le aplican los conceptos temporales — nunca son verdaderas con respecto a cualquier tiempo y a cualesquiera punto o intervalo (por ejemplo, "*el cohete ni se está moviendo ni es inmóvil*", "*la luz está encendida y apagada al mismo tiempo*", etc.).
- Por otra parte, mientras que algunas proposiciones temporales pueden asociarse válida y significativamente tanto a los puntos del tiempo

como a los intervalos del tiempo (por ejemplo, "*el cohete era inmóvil*"), otros pueden ser solamente aplicables a los puntos del tiempo (por ejemplo, "*el cohete, que se había estado moviendo continuamente en la misma dirección, no logró cambiar su posición dentro de un solo instante.* "), o solamente aplicable a los intervalos del tiempo (por ejemplo, "*el cohete logró un cambio a su posición*"). NOTA. En lo qué sigue de este capítulo, el término "*movimiento*" será entendido como "*cambio de posición en el espacio*".

- Además, el valor booleano de una proposición temporal con respecto a algunos elementos de tiempo puede ser dependiente en su valor booleano con respecto a otros elementos de tiempo. Por ejemplo, sea F la proposición "*el cohete el cual había sido lanzado al aire se movía*". Si queremos asociar F con los puntos de tiempo, entonces, por sentido común, F es verdadera en un punto P si y solamente si hay un intervalo del tiempo I sobre el cual F es verdadera y P está dentro de ese intervalo. En otras palabras, el valor booleano de una proposición aplicable a un punto puede ser dependiente de un intervalo. Analogamente, el valor booleano de una proposición aplicable a un intervalo puede ser dependiente de un punto o dependiente de un intervalo, aunque tales clases de dependencias pueden ser innecesarias para otros tipos de proposiciones temporales (las definiciones formales de la dependencia de un punto/intervalo se darán más adelante en el capítulo).

Por lo que, los problemas pueden presentarse cuando uno combina diversos puntos de vistas sobre la estructura y hacer preguntas temporales, si algunos tipos de proposiciones temporales se pueden asociarse válida y significativamente a diversos elementos de tiempo.

Además, vale la pena precisar que, aunque en la literatura se presentan prolongados debates referentes a algunos términos/fenómenos temporales típicos, estos términos/fenómenos realmente no se han caracterizado formalmente de manera explícita. Entre éstos fenómenos se encuentran:

- *Inicio instantáneo*: un punto del tiempo en el cual una proposición se convierte en verdadera.
- *Fin instantáneo*: un punto del tiempo en el cual una proposición termina su valor Booleano.
- *División instantánea*: un punto del tiempo que divide dos intervalos de tiempo sucesivos sobre los cuales se cumplen dos proposiciones incompatibles, respectivamente.
- *Instigación*: un proceso con un inicio instantáneo.
- *Terminación*: un proceso con un fin instantáneo.
- *Cambio continuo* (*intermingling*, en inglés): la posibilidad de una proposición que cambia frecuentemente su valor booleano infinitamente dentro de un intervalo con una duración finita (Galton, 1996).

En términos de los meta-predicados introducidos en la sección anterior, proponemos una clasificación de las proposiciones temporales como sigue:

C1. La proposición f es aplicable a los conceptos temporales si

$\exists$t(Holds(f, t))

Ejemplos (es decir, son aplicables los conceptos temporales):
1. *El cohete era inmóvil.*
2. *El cohete se movía.*
3. *El cohete logró el cambio a su posición.*
4. *El cohete, que se había estado moviendo continuamente en la misma dirección, no logró el cambio a su posición dentro de un instante.*
5. *El cohete que había sido lanzado en el aire se movía hacia arriba (abajo).*
6. *El cohete que había sido lanzado en el aire cayó en la tierra.*
7. *El cohete que había sido lanzado al aire estaba en su cima (altura máxima).*
8. *El cohete alcanzó su máximo.*

Contraejemplos (es decir, **no** se aplican los conceptos temporales):
1. *El cohete ni se movía ni era inmóvil.*
2. *La luz era encendida y apagada al mismo tiempo.*

C2. La proposición f es aplicable a los puntos de tiempo si

$\exists$t(HoldsAT(f, t))

Ejemplos:
1. *El cohete era inmóvil.*
2. *El cohete, que se había estado moviendo continuamente en la misma dirección, no logró el cambio a su posición dentro de un instante.*

Contraejemplo:
El cohete logró el cambio a su posición.

C3. La proposición f es aplicable solamente a los punto de tiempo si

Holds(f, t) $\Rightarrow$ Dur(t) = 0

Ejemplo:
El cohete, que se había estado moviendo continuamente en la misma dirección, no logró el cambio a su posición dentro de un instante.

Contraejemplo:
El cohete era inmóvil.

C4. La proposición f es aplicable a los intervalos de tiempo si

$\exists$t(Dur(t) $>$ 0 $\wedge$ Holds(f, t))

Ejemplos:
1. *El cohete era inmóvil.*
2. *El cohete logró el cambio a su posición.*

Contraejemplo:
El cohete, que se había estado moviendo continuamente en la misma dirección, no logró el cambio de su posición.

C5. La proposición f es aplicable a los intervalos de tiempo solamente si

$\text{Holds}(f, t) \Rightarrow \text{Dur}(t) > 0$

Ejemplo:
El cohete logró el cambio a su posición.

Contraejemplo:
El cohete era inmóvil.

C6. La proposición f aplicable a los puntos de tiempo es dependiente de intervalos de tiempo si:

$\text{HoldsAT}(f, t) \Rightarrow \exists t_1(\text{Dur}(t_1) > 0 \wedge \text{In}(t, t_1) \wedge \text{HoldsON}_I(f, t_1))$

Ejemplo:
El cohete se movía hacia arriba.

Contraejemplo:
El cohete que había sido lanzado en el aire estaba en su cima.

C7. La proposición f aplicable a los intervalos de tiempo es dependiente de los punto del tiempo si:

$\text{Holds}(f, t) \wedge \text{Dur}(t) > 0 \Rightarrow \text{HoldsIN}_P(f, t)$

Ejemplo:
El cohete alcanzó su cima.

Contraejemplo:
El cohete logró el cambio de su posición.

NOTA: Las proposiciones dependientes de los puntos de tiempo son solamente aplicables a los intervalos de tiempo.

C8. La proposición f aplicable a los intervalos de tiempo es dependiente de los intervalos de tiempo si

$\text{Holds}(f, t) \wedge \text{Dur}(t) > 0 \Rightarrow \exists t_1(\text{Dur}(t_1) > 0 \wedge \text{In}(t_1, t) \wedge \text{Dur}(t_1) < \text{Dur}(t) \wedge \text{Holds}(f, t_1))$

Ejemplo:

El cohete, que se había estado moviendo continuamente, logró el cambio de su posición.

Contraejemplo:
El cohete, que se había estado moviendo continuamente en la misma dirección, avanzó dos millas.

C9. La proposición f es extensible en la frontera si:

$\text{Holds}(f,t) \wedge \text{Dur}(t_1) = 0 \wedge (\text{Meets}(t_1,t) \vee \text{Starts}(t_1,t) \vee \text{Finishes}(t_1,t) \vee \text{Meets}(t,t_1))$
$\Rightarrow \text{HoldsAT}(f,t_1)$

Ejemplo:
El cohete no cambió su posición.

Contraejemplo:
El cohete que había sido lanzado en el aire estaba debajo de su cima.

NOTA: En las clases siguientes (**C10 - C29**), adoptaremos directamente algunos términos de la teoría de Shoham de los tipos de proposiciones, tales como hereditario arriba, hereditario abajo, concatenable, gestalt, líquido, y sólido, etc. (Shoham, 1987).

C10. La proposición f es hereditaria-abajo-universalmente si

$\text{Holds}(f, t) \Rightarrow \text{HoldsON}(f, t))$

Ejemplos:
1. *El cohete era inmóvil.*
2. *El cohete, que se había estado moviendo continuamente en la misma dirección, avanzó menos de dos millas.*

Contraejemplo:
El cohete, que se había estado moviendo continuamente en la misma dirección, avanzó exactamente dos millas (del principio al final).

C11. La proposición f es hereditaria-arriba-universalmente si:

$\forall t_1(\text{In}(t_1, t) \Rightarrow \text{Holds}(f, t_1)) \Rightarrow \text{Holds}(f, t)$

Ejemplos:
1. *El cohete era inmóvil.*
2. *El cohete se movía hacia arriba.*

Contraejemplo:
El cohete, que se había estado moviendo continuamente en la misma dirección, avanzó menos de dos millas.

C12. La proposición f es hereditaria-abajo-sobre-puntos si

$\text{Holds(f, t)} \Rightarrow \text{HoldsON}_{\text{P}}\text{(f, t)}$

Ejemplo:
El cohete es inmóvil.

Contraejemplo:
El cohete logró el cambio a su posición.

C13. La proposición f es hereditaria-arriba-sobre-puntos si

$\text{HoldsON}_{\text{P}}\text{(f, t)} \Rightarrow \text{Holds(f, t)}$

Ejemplo:
El cohete está en el aire más bien que en la tierra.

Contraejemplo:
El cohete, que se había estado moviendo continuamente en la misma dirección, no logró el cambio a su posición dentro de un instante.

C14. La proposición f es hereditaria-abajo-sobre-intervalos si

$\text{Holds(f, t)} \Rightarrow \text{HoldsON}_{\text{I}}\text{(f, t)}$

Ejemplos:
1. *El cohete era inmóvil.*
2. *El cohete, que se había estado moviendo continuamente en la misma dirección, avanzó menos de dos millas.*

Contraejemplo:
El cohete, que se había estado moviendo continuamente en la misma dirección, avanzó exactamente dos millas (del principio al final).

C15. La proposición f es hereditaria-abajo-sobre-intervalos si:

$\forall t_1(\text{Dur}(t_1) > 0 \wedge \text{In}(t_1, \text{t}) \Rightarrow \text{Holds(f, } t_1)) \Rightarrow \text{Holds(f, t)}$

Ejemplo:
El cohete logró el cambio de su posición.

Contraejemplo:
El cohete, que se había estado moviendo continuamente en la misma dirección, avanzó menos de dos millas.

C16. La proposición f es líquida-universal-universal si es hereditaria-abajo-universal y hereditaria-arriba-universal.

Ejemplo:
El cohete es inmóvil.

Contraejemplos:

1. (en cuanto a hereditaria-abajo-universal) *El cohete, que se había estado moviendo continuamente en la misma dirección, avanzó exactamente dos millas (del principio al final*).
2. (en cuanto a hereditaria-arriba-universal) *el cohete, que se había estado moviendo continuamente en la misma dirección, avanzó menos de dos millas.*

Análogamente, damos las definiciones siguientes que se refieren a varias clases de proposiciones líquidas:

C17. La proposición f es líquida-universal-sobre-puntos si es hereditaria-abajo-universal y hereditaria-arriba-sobre-puntos.

C18. La proposición f es líquida-universal-sobre-intervalos si es hereditaria-abajo-universal y hereditaria-arriba-sobre-intervalos.

C19. La proposición f es líquida-sobre-puntos-universal si es hereditaria-arriba-universal y hereditaria-abajo-sobre-puntos.

C20. La proposición f es líquida-sobre-puntos-sobre-puntos si es hereditaria-arriba-sobre-puntos y hereditaria-abajo-sobre-puntos.

C21. La proposición f es líquida-sobre-puntos-sobre-intervalos si es hereditaria-abajo-sobre-puntos y hereditaria-arriba-sobre-intervalos.

C22. La proposición f es líquida-sobre-intervalos-universal si es hereditaria-abajo-sobre-intervalos y hereditaria-arriba-universal.

C23. La proposición f es líquida-sobre-intervalos-sobre-puntos si es hereditaria-abajo-sobre-intervalos y hereditaria-arriba-sobre-puntos.

C24. La proposición f es líquida-sobre-intervalos-sobre-intervalos si es hereditaria-arriba-sobre-intervalos y hereditaria-abajo-sobre-intervalos.

C25. La proposición f es concatenable si

$$\text{Holds}(f, t_1) \wedge \text{Holds}(f, t_2) \wedge \text{Meets}(t_1, t_2) \Rightarrow \text{Holds}(f, t_1 \oplus t_2)$$

Ejemplos:
1. *El cohete, que se había estado moviendo continuamente en la misma dirección, avanzó más de dos millas.*
2. *El cohete se movía hacia arriba.*

Contraejemplo:
El cohete, que se había estado moviendo continuamente en la misma dirección, avanzó menos de dos millas.

C26. La proposición f es gestalt si

$\text{Holds}(f, t_1) \wedge \text{In}(t_2, t_1) \Rightarrow \neg\text{Holds}(f, t_2)$

Ejemplos:
1. *Correr una milla de principio a fin.*
2. *Dibujar un círculo de principio a fin.*

Contraejemplos:
1. *El cohete es inmóvil.*
2. *Pasaron exactamente seis minutos* (Shoham, 1987).

NOTA. El término "gestalt" se usa en (Shoham, 1987) para definir un tipo de proposición que nunca se cumpla sobre dos intervalos uno de los cuales contenga el otro como el intervalo propio. Es interesante observar que las proposiciones de gestalt de Shoham son de hecho muy similares a los eventos de Allen (Allen, 1984), aparte de que los intervalos de Shoham están definidos en términos de pares de puntos, mientras que Allen los trata como primitivas. Además, el segundo contraejemplo en C26 se categoriza realmente como gestalt en categorías de Shohams debido al hecho que los valores booleanos de proposiciones en las fronteras de intervalos no están especificados (Shoham 1987). Sin embargo, es solamente gestalt-tolerante-a-puntos (véase la definición siguiente).

C27. La proposición f es gestalt-tolerante-a-puntos si

$\text{Holds}(f, t_1) \wedge \text{In}(t_2, t_1) \wedge \text{Duration}(t_2) < \text{Duration}(t_1) \Rightarrow \neg\text{Holds}(f, t_2)$

Ejemplos:
1. *Pasaron exactamente seis minutos* (Shoham, 1987).
2. *La duración del elemento de tiempo es de 5 unidades.*

Contraejemplos:
1. *El cohete es inmóvil.*
2. *El cohete, que se había estado moviendo continuamente en la misma dirección, cambió su posición.*

C28. La proposición f es sólida si

$\text{Holds}(f, t_1) \wedge \text{Overlaps}(t_2, t_1) \Rightarrow \neg\text{Holds}(f, t_2)$

Ejemplo:
El cohete inmóvil empezó a moverse a la 1:00, siguió moviéndose continuamente por seis minutos y dejó de moverse a las 1:06.

NOTA: Aquí, los puntos de frontera 1:00 y 1:06 tienen que ser inclusivos.

Contraejemplos:
El cohete es inmóvil.

C29. La proposición f es sólida-tolerante-a-puntos si:

$\text{Holds}(f, t_1) \wedge t_1 = t_1' \oplus t \wedge t_2 = t \oplus t_2' \wedge \text{Dur}(t_1') > 0 \wedge \text{Dur}(t_2') > 0 \Rightarrow \neg\text{Holds}(f, t_2)$

Ejemplos:
El cohete se estaba moviendo en la misma dirección por 6 seis minutos entre 1:00 y 1:06.

NOTA: Aquí, decimos que esta proposición es sólida-tolerante-a-puntos en lo que respecta a los momentos del tiempo 1:00 y 1:06 puesto que estos dos puntos de frontera pueden ser inclusivos o exclusivos.

Contraejemplo:
El cohete es inmóvil.

C30. La proposición f es tendencia-a-instigación si:

$\text{Holds}(f, t) \wedge \text{Dur}(t_1) > 0 \wedge \text{Starts}(t_1, t) \Rightarrow \text{Holds}(f, t_1)$

Ejemplos:
1. *El cohete se movía hacia arriba desde la tierra.*
2. *Juan iba de su oficina a casa.*

Contraejemplo:
El cohete, que se había estado moviendo continuamente en la misma dirección, avanzó dos millas.

C31. La proposición f es tendencia-a-terminación si:

$$\text{Holds}(f, t) \wedge \text{Dur}(t_1) > 0 \wedge \text{Finishes}(t_1, t) \Rightarrow \text{Holds}(f, t_1)$$

Ejemplos:
1. *El cohete alcanzaba su cima.*
2. *Juan acabó su ensayo.*

Contraejemplo:
El cohete, que se había estado moviendo continuamente en la misma dirección, avanzó dos millas.

C32. Se dice que la proposición f tiene un punto t como su inicio instantáneo si:

$$\text{Holds}(f, t_1) \wedge \exists t_2(\text{Holds}(\text{not}(f), t_2) \wedge \text{Meets}(t_2, t) \wedge \text{Meets}(t_2, t_1))$$

Ejemplo:
El punto de tiempo en el cual el cohete fue lanzado en el aire es el inicio instantáneo de la proposición en que el cohete se estaba moviendo.

C33. Se dice que la proposición f tiene el punto t como su fin instantáneo si:

$\text{Holds}(f, t_1) \wedge \exists t_2(\text{Holds}(\text{not}(f), t_2) \wedge \text{Meets}(t_1, t) \wedge \text{Meets}(t_1, t_2))$

Ejemplo:
El punto de tiempo en el cual el cohete que había sido lanzado al aire alcanzó su cima es el fin instantáneo de la proposición de que el cohete se estaba moviendo.

C34. La proposición f_1 es incompatible con la proposición f_2 si

$\text{Holds}(f_1, t) \Rightarrow \text{Holds}(\text{not}(f_2), t)$

Ejemplo:
La proposición "el fuego ardía" es incompatible con la proposición "el fuego se apagó".
NOTA: Es fácil observar que la proposición f_1 es incompatible con la proposición f_2 si y solamente si la proposición f_2 es incompatible con la proposición f_1.

C35. Se dice que dos proposiciones incompatibles f_1 y f_2 tienen el punto t como su división instantánea (instante divisor) si:

$\text{Holds}(f_1, t_1) \wedge \text{Holds}(f_2, t_2)$
$\wedge (\text{Meets}(t_2, t) \wedge \text{Meets}(t, t_1) \vee \text{Meets}(t_2, t) \wedge \text{Meets}(t_2, t_1) \vee \text{Meets}(t_2, t_1) \wedge \text{Meets}(t, t_1)$
$\vee \text{Meets}(t_1, t) \wedge \text{Meets}(t, t_2) \vee \text{Meets}(t_1, t) \wedge \text{Meets}(t_1, t_2) \vee \text{Meets}(t_1, t_2) \wedge \text{Meets}(t, t_2))$

Ejemplos:
1. *El punto de tiempo en el cual el cohete inmóvil empezó a moverse es división instantánea de la proposición en que el cohete era inmóvil y de la proposición de que el cohete se estaba moviendo.*
2. *El punto de tiempo en el cual el cohete que había sido lanzado al aire estaba en su cima es división instantánea de la proposición, que el cohete se movía hacia arriba y de la proposición, que el cohete se movía hacia abajo.*

No es sorpresa que otras categorías de proposiciones temporales pueden ser definidas. Sin embargo, terminamos aquí puesto que, como será demostrado abajo, por un lado, la nueva clasificación propuesta aquí puede incluir las categorías representativas mencionadas en los trabajos anteriores y en gran medida los supera en su expresividad; por otra parte, esta clasificación ya tiene la capacidad de caracterizar fenómenos temporales típicos y de verter la luz sobre los teoremas y rompecabezas que de otra manera serían algo misteriosos.

6. Ilustraciones de uso y ejemplos

En la sección 2.1.3 de este capítulo, introdujimos una teoría de tiempo que toma puntos e intervalos como primitivas de la misma naturaleza — ni los intervalos tienen que ser construidos de los puntos, ni los puntos se tienen que definir como concepto que limita a los intervalos. De un lado, esta teoría de tiempo extendida conserva las características atractivas de la teoría basada en intervalos de Allen (Allen, 1983), la cual evita la molesta pregunta si o no un punto dado es parte de un intervalo; por otra parte, la teoría acomoda con éxito los puntos de tiempo de una manera constante, permitiendo utilizar como referencias temporales a los fenómenos instantáneos. En el texto siguiente, ilustramos algunas aplicaciones de esta teoría de tiempo.

6.1 Representación del instante divisor

El problema de la división instantánea (*Dividing Instant Problem* (*DIP*)) fue nombrado formalmente por Van Benthem (1983) en una discusión sobre las contradicciones de los cambios que ocurren en el transcurso de tiempo. Van Benthem cita el ejemplo de un fuego que ha estado ardiendo y se extingue más adelante, y hace la pregunta de qué exactamente sucede en el instante intermedio entre los dos estados sucesivos de arder y ser extinguido. En otra descripción del problema, Allen (1983) da un ejemplo de una luz que ha sido apagada, y se enciende después al mover un interruptor. ¿Está la luz apagada o encendida en el punto del cambiar el interruptor?

Las preguntas filosóficas como el rompecabezas de la división instantánea datan de hace más de 2300 años en la era de Aristóteles (véase 234a en *Física*, *libro VI*, (Hardie and Gaye, 1930)). Como sucede con muchas preguntas filosóficas famosas, hay varias maneras de responderla. Las respuestas más comunes son:

- El *DIP* se presenta solamente cuando uno insiste en asociar una proposición a un punto de tiempo, lo que, según afirma Allen (Allen 1983; 1984), no es una entidad en la cual las cosas suceden o son verdaderas. Por lo tanto, los sistemas basados en intervalos que excluyen puntos de la ontología temporal no presentan tal problema.
- El enfoque que caracteriza todos los intervalos como medio abiertos, por ejemplo, cerrados por la izquierda y abierto por la derecha, hará que los intervalos sucesivos del tiempo se acomoden convenientemente uno al lado de otro (Maiocchi, 1992). Por lo que, si una proposición es verdadera en un intervalo del tiempo y después se convierte en falsa en el intervalo subsecuente, su valor booleano en el instante divisor de los dos intervalos sucesivos será especificado como falso. Este enfoque ha sido criticado como arbitrario/artificial (Allen, 1983, 1984; Galton,

1990), puesto que no hay ninguna razón para especificar todos los intervalos como, por ejemplo, cerrados por la izquierda y abierto por la derecha, y no abiertos por la izquierda y cerrados por la derecha. ¿Entonces, qué hacemos? Mientras una solución práctica sea aplicable a la mayoría de los casos, ¿por qué debemos incomodarnos con argumentos filosóficos?

- El *DIP* es un asunto sobre la interconexión de diversas clases de proposiciones, a saber, el problema de relacionar el valor booleano de una proposición sobre un intervalo con su valor booleano en un punto. El problema se presenta cuando se combinan diversos puntos de vista sobre la estructura temporal, y resulta que el mismo predicado, por ejemplo, "un cuerpo está en una posición particular", o "un cuerpo está en movimiento", se aplica válida y significativamente tanto a los intervalos extendidos como a los puntos sin duración (van Bentham, 1998).

En cuanto al tratamiento del problema de división instantánea, suponemos que F y G denoten dos proposiciones incompatibles que son verdaderas sobre dos intervalos sucesivos i y j, respectivamente. Según lo presentado en (Ma and Knight, 2003), hay realmente 4 casos posibles con respecto al instante que las divide, digamos p, que se encuentra entre dos estados sucesivos:

Caso 1:
Éste es el caso donde no hay conocimiento cualquiera con respecto al instante divisor. Todo lo que sabemos es que la proposición F es verdadera en el intervalo i y la proposición G es verdadera en el intervalo subsecuente j. Este caso puede tener dos interpretaciones:

Caso 1.1:
No existe conceptualmente un instante divisor; a lo más, el instante divisor se puede tomar simplemente como el "lugar de la reunión" de los dos intervalos — no es una entidad real en la cual F o G puedan tener valor. Éste es de hecho la solución de Allen al problema de la división instantánea (Allen, 1983).

Caso 1.2:
El instante divisor existe y es una entidad en la cual el valor booleano de las proposiciones F y G puede ser especificado. Sin embargo, en este caso, no tenemos ningún conocimiento con respecto al valor booleano de la proposición F ni de la proposición G en el instante p - es simplemente desconocido. Es decir, no se da ninguna suposición en cuanto a la naturaleza (cerrado o abierto) del intervalo i y del intervalo j en el punto p.

Caso 2:
Éste es el caso donde, por alguna razón relacionada con las aplicaciones específicas, existe la suposición impuesta con respecto al instante divisor p que

la proposición F se considera falsa en el punto p. Es decir, el intervalo i está abierto por la derecha en el punto p, mientras que el intervalo j está cerrado por la izquierda en el punto p (y por lo tanto, *Meets*(i, p)). Es decir, mientras el intervalo i "se encuentra (Meets)" con el intervalo j, el punto p es "encontrado (Met-by)" por el intervalo i y "comienza (Starts)" el intervalo j.

Caso 3:

Éste es el caso donde, otra vez, por alguna razón relacionada con las aplicaciones específicas, existe la suposición impuesta con respecto al instante divisor p que la proposición F se considera verdadera en el punto p. Es decir, se asume que el intervalo i está cerrado por la derecha en el punto p; y el intervalo j se asume abierto por la izquierda en el punto p (y por lo tanto, *Meets*(p, j)). Es decir, mientras el intervalo i "se encuentra (Meets)" con el intervalo j, el punto p "termina (Finishes)" el intervalo i y "se encuentra (Meets)" con el intervalo j.

Además, existe otro caso que está estrechamente relacionado con el problema de la división instantánea. Este es,

Caso 4:

Este es el caso donde, a diferencia de los tres casos anteriores donde el intervalo i se encuentra con j inmediatamente, existe un lapso con duración cero, es decir, el punto p se encuentra entre los intervalos i y j. Es decir, ambos intervalos i y j están abiertos en el punto p. Sin embargo, podemos seguir considerando p como "instante divisor" que divide los dos intervalos, cada uno de los cuales se encuentra junto con el otro casi inmediatamente (es decir, la duración del lapso entre ellos es cero).

Es fácil notar que éstos son todos los casos posibles relacionados con la aplicación del instante divisor según lo definido en C35, véase la sección 5.

Para ilustrar todos los casos/suposiciones relacionados con el *DIP*, consideramos el escenario siguiente:

La energía fue cortada; un robot era inmóvil; un coche había sido acelerado continuamente desde 0 mph hasta 100 mph; y una bola fue lanzada al aire del este al oeste. Exactamente en el momento cuando la bola alcanzó su cima (altura máxima), la energía fue reestablecida, y el indicador de la velocidad del coche alcanzó 60 mph; inmediatamente después de restablecerse la energía, el robot comenzó a moverse.

Para el razonamiento general, imponemos las suposiciones/hipótesis siguientes:

- El estado en que la energía fue "apagada" fue seguido inmediatamente por el estado de "encendido".
- El estado en que la velocidad del coche estuvo debajo de los 60 mph fue seguido inmediatamente por el estado que ya no está debajo de los 60 mph; y cuando la bola llegó en su cima, la velocidad del coche ya no estaba debajo de los 60 mph.

- El estado en que el robot no se movía fue seguido inmediatamente por el estado que el robot se movía; y en el momento cuando la bola estaba en su cima, el robot no se movía.
- El estado que la bola estaba en el este y debajo de su cima fue seguido inmediatamente por el estado en que la bola estaba en su cima, y que, en su turno, fue seguido inmediatamente por el estado en que la bola estaba en el oeste y debajo de la cima.

En los términos de la teoría basada en puntos e intervalos y de los meta predicados de tiempo introducidos en las secciones 3 y 4, respectivamente, estas suposiciones/hipótesis pueden expresarse respectivamente como:

(i)
HoldsON(PowerOff, I_1)
HoldsON(PowerOn, J_1)
Meets(I_1, J_1)
(ii)
HoldsON(SpeedBelow60, I_2)
HoldsON(SpeedNoLongerBelow60, J_2)
HoldsAT(SpeedNoLongerBelow60, P)
Meets(I_2, J_2)
Meets(I_2, P)
(iii)
HoldsON(RobotNotMoving, I_3)
HoldsON(RobotMoving, J_3)
HoldsAT(RobotNotMoving, P)
Meets(I_3, J_3)
Meets(P, J_3)
(iv)
HoldsON(EastBelow, I_4)
HoldsAT(AtApex, P)
HoldsON(WestBelow, J_4)
Meets(I_4, P)
Meets(P, J_4)

donde Duration(P) = 0, Duration(I_n) > 0 and Duration(J_n) > 0, n = 1, 2, 3, 4.

Es interesante observar que, en la situación descrita en el escenario mencionado, para el cambio de estado de la energía, se sabe que no sólo a) el estado en que la energía fue "encendida" fue seguido inmediatamente por el estado en que fue "apagada", pero también, a') es que cuando la bola alcanzó su cima, la energía cambió su estado de "apagado" a "encendido". Esta pieza adicional del conocimiento, es decir, a'), también puede ser expresada con éxito. De hecho, de la calidad de homogeneidad del predicado *HoldsON* (es decir, si una proposición es verdadera sobre un intervalo entonces es verdadera

en cualquier parte de ese intervalo), podemos deducir que hay intervalos I_1', J_1', I_4' y J_4', tales que:

A')

$\text{Equal}(I_1', I_1) \vee \text{Finishes}(I_1', I_1)$
$\text{Equal}(J_1', J_1) \vee \text{Starts}(J_1', J_1)$
$\text{Equal}(I_4', I_4) \vee \text{Finishes}(I_4', I_4)$
$\text{Equal}(J_4', J_4) \vee \text{Starts}(J_4', J_4)$
$I_1' \oplus J_1' = I_4' \oplus P \oplus J_4'$
$\text{Duration}(I_1') = \text{Duration}(I_4')$

De $I_1' \oplus J_1' = I_4' \oplus P \oplus J_4'$, obtenemos $\text{Duration}(I_1' \oplus J_1') = \text{Duration}(I_4' \oplus P \oplus J_4')$, y con $\text{Duration}(P) = 0$ y $\text{Duration}(I_1') = \text{Duration}(I_4')$, podemos deducir que $\text{Duration}(J_1') = \text{Duration}(J_4')$. Por lo tanto, A') incluye el conocimiento de que el "lugar del encuentro" de los intervalos I_1' y J_1' es el punto P. Puesto que el "lugar del encuentro" de los intervalos I_1' y J_1' es también el "lugar del encuentro" de los intervalos I_1 y J_1, por lo tanto, la pieza adicional del conocimiento a') se puede deducir de A').

Una representación gráfica de los casos mencionados se muestra en la figura siguiente (Ilustración 1):

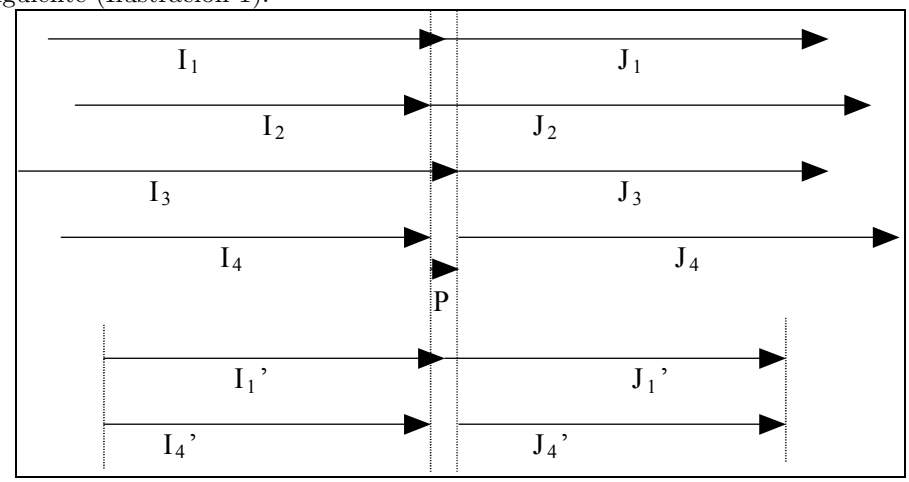


Ilustración 1. Representación gráfica de los ejemplos.

6.2 Representación temporal en narrativas

La representación y el razonamiento temporal de la Inteligencia Artificial penetran muchas áreas de aplicaciones diversas, por ejemplo, leyes, ingeniería y lenguaje natural. Particularmente, los informes de los agentes, o cualquier

estructura narrativa, implican perspectiva: la relación entre el tiempo del narrador y los eventos narrados.

Consideremos por ejemplo un escenario donde dos personas, Peter y Jack, están bajo sospecha de cometer un asesinato durante el día. Ante el tribunal, Jack y Peter declararon lo siguiente:

– Declaraciones de Peter:

"Llegué a casa junto con Jack antes de la 1 pm. Almorzamos juntos, y cuando Jack se fue puse un vídeo. El vídeo dura 2 horas. Antes de que terminara el vídeo, llegó Robert. Cuando el vídeo terminó fuimos a la estación del tren y esperamos a Jack hasta que él llego a las 4 de la tarde."

– Declaraciones de Jack:

"Peter y yo fuimos a su casa y llegamos allá antes de la 1 pm. Cuando terminamos de almorzar allí, Peter puso un vídeo, y yo me fui al supermercado. Permanecía allí entre la 1 y las 2. Después conduje a mi casa para recoger mi correo. Toma entre 1.5 a 2 horas para llegar hasta mi casa, y toma casi el mismo tiempo para llegar a la estación del tren. Llegué a la estación del tren a las 4 pm."

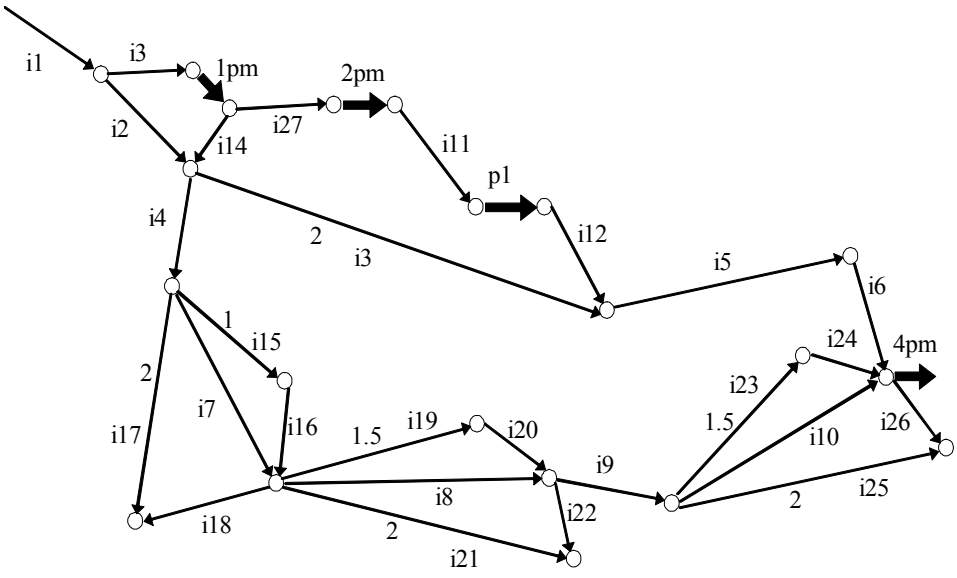


Ilustración 2. Grafo temporal del ejemplo 2.

– Además, siendo el testigo, Robert hizo sus declaraciones:

"Salí de mi casa a las 2 pm y fui a la casa de Peter. Él estaba viendo un vídeo, y esperamos hasta que se terminó. Después fuimos juntos a la estación del tren y esperamos a Jack. Jack llegó a la estación del tren a las 4 pm."

Usando la representación gráfica introducida en (Knight and Ma, 1992), podemos expresar el escenario de sus declaraciones en los términos de un grafo temporal como se muestra en Ilustración 2, donde:

i_1: el tiempo (intervalo) cuando Peter y Jack fueron a casa de Peter;

1 pm: el tiempo de referencia (punto) antes del cual ellos llegaron a casa de Peter;

i_2: el tiempo (intervalo) cuando Peter y Jack estaban almorzando;

i_3: el tiempo (intervalo) cuando Peter estaba viendo el video (Dur (i_3) = 2);

i_4: el tiempo (intervalo) cuando Jack fue al supermercado;

p_1: el tiempo (punto) cuando Robert llegó a casa de Peter;

i_5: el tiempo (intervalo) cuando Peter y Robert fueron a la estación del tren;

i_6: el tiempo (intervalo) cuando Peter y Robert estuvieron esperando a Jack en la estación del tren;

4 pm: el tiempo (punto) cuando Jack llegó a la estación del tren;

i_7: el tiempo (intervalo) cuando Jack permaneció en el supermercado ($1<$Dur (i_7) < 2);

i_8: el tiempo (intervalo) cuando Jack estuvo conduciendo a su casa ($1.5<$Dur (i_8) < 2);

i_9: el tiempo (intervalo) cuando Jack recogió correo en su casa;

i_{10}: el tiempo (intervalo) cuando Jack estuvo conduciendo a la estación del tren ($1.5 <$ Dur (i_{10}) < 2);

2 pm: el tiempo (punto) cuando Robert salió de la casa;

i_{11}: el tiempo (intervalo) cuando Robert fue a la casa de Peter;

i_{12}: el tiempo (intervalo) cuando Peter y Robert estuvieron viendo el vídeo juntos;

i_{13}, i_{14},..., i_{27}: algunos elementos relativos adicionales de tiempo que se utilizan para expresar el conocimiento relacionado con la duración, por ejemplo, con i_{19}, i_{20}, i_{21}, i_{22}, y Dur (i_{19}) = 1.5 y Dur (i_{21}) = 2, podemos expresar que $1.5 <$ Dur (i_8) < 2 (véase Ilustración 2).

Aquí, los elementos de tiempo se muestran como los arcos de un grafo (los arcos con las líneas delgadas representan los intervalos de tiempo, específicamente, i_1, i_2,..., i_{27}; y los arcos con las líneas gruesas representan los puntos de tiempo, es decir, 1 pm, 2 pm, 4 pm y p_1). La relación de un precursor inmediato Meets(t_i, t_j) es representada por el hecho que t_i se encuentre dentro del arco y t_j se encuentre fuera del arco del mismo nodo. Los arcos con pesos representan los intervalos que se sabe sus duraciones absolutas (por ejemplo, Dur (i_3) = 2). Los arcos sin pesos corresponden a los intervalos con duraciones desconocidas o relativas, donde el conocimiento de la duración relativa se puede representar por medio de algunos elementos del tiempo

relacionados con algunas duraciones absolutas conocidas (por ejemplo, i_{19}), o no relacionadas con tales duraciones (por ejemplo, i_{20}).

La condición necesaria y suficiente para que la consistencia de un grafo temporal pueda ser definida como en (Knight and Ma, 1992) es:

1) Existe una asignación de las duraciones de tiempo en el grafo, tal que, para cada circuito simple en el grafo, la suma de duraciones (calculada tomando en cuenta las direcciones) es cero;
2) Para cualquier dos elementos de tiempo adyacentes, la suma de las duraciones es mayor que cero.

Ahora, consideramos si el grafo temporal mencionado es consistente o no:

En Ilustración 3, hay tres elementos de tiempo (es decir, dos intervalos, i_{11} e i_{12}, y un punto p_1) que se encuentran entre 2 pm y 4 pm. Puesto que cada intervalo tiene una duración positiva y cada punto tiene una duración no negativa, podemos deducir que:

$$Dur(i_5) + Dur(i_6) < 2$$

Adicionalmente, ya que $Dur(i_3) = 2$, entonces

$$Dur(i_3) + Dur(i_5) + Dur(i_6) < 2 + 2 = 4$$

Sin embargo,

$$Dur(i_4) + Dur(i_7) + Dur(i_8) + Dur(i_9) + Dur(i_{10}) > 0 + 1 + 1.5 + 0 + 1.5 = 4$$

Entonces, para cada circuito simple i_3, i_5, i_6, i_{10}, i_9, i_8, i_7, i_4, como se muestra en la Ilustración 3, no existe alguna asignación de duración, tal que

$$Dur(i_3) + Dur(i_5) + Dur(i_6) = Dur(i_4) + Dur(i_7) + Dur(i_8) + Dur(i_9) + Dur(i_{10})$$

es decir,

$$Dur(i_3) + Dur(i_5) + Dur(i_6) - Dur(i_4) - Dur(i_7) - Dur(i_8) - Dur(i_9) - Dur(i_{10}) = 0$$

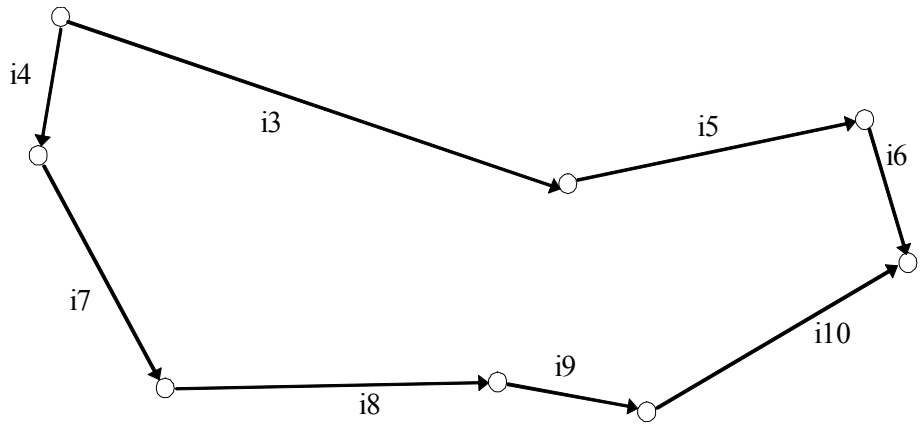


Ilustración 3. Representación temporal de las declaraciones de testigos del ejemplo 2.

Por lo tanto, la representación temporal presentada en la Ilustración 3 es inconsistente, y por tanto podemos cerciorarnos directamente que algunas afirmaciones son falsas. Suponemos que se puede comprobar que el vídeo dura realmente dos horas, entonces debe haber una cierta falsedad en las declaraciones o de Robert o de Jack. Si se puede comprobar que Robert no salió de la casa a las 2 pm, entonces Jack debe haber mentido al hacer sus declaraciones. Si no, para convencernos que sus declaraciones sean verdaderas, Jack debe demostrar que Robert salió de la casa en algún momento antes de las 2 pm.

6.3 Razonando sobre evento, cambio y causalidad

Durante el último medio siglo, muchos enfoques se han propuesto para razonar sobre los aspectos dinámicos del mundo. Generalmente hablando, en la mayoría de estos enfoques, el mundo asume en persistir en un estado dado hasta que ocurre un cierto evento en un momento específico para cambiarlo a otro estado. Por lo que, la representación y razonamiento sobre acciones/eventos y sus efectos es esencial en la modelación de los aspectos dinámicos del mundo. En este capítulo, definimos un estado del mundo (denotémoslo por *s*) en el discurso como colección de proposiciones.

Según el enfoque propuesto en (Shanahan, 1995), utilizamos *Belongs*(f, s) (*Pertenece*(f, s) en español) para denotar que la proposición *p* pertenece a la colección de proposiciones que representan el estado *s*:

(6.3.1) $s_1 = s_2 \Leftrightarrow \forall f(\text{Belongs}(f, s_1) \Leftrightarrow \text{Belongs}(f, s_2))$

Es decir, dos estados son iguales si y solamente si contienen los mismos proposiciones.

(6.3.2) $\exists s \forall f(\neg \text{Belongs}(f, s))$

Es decir, existe un estado que sea un conjunto vacío de proposiciones.

(6.3.3) $\forall s_1 f_1 \exists s_2 (\forall f_2 (\text{Belongs}(f_2, s_2) \Leftrightarrow \text{Belongs}(f_2, s_1) \vee f_1 = f_2))$

Es decir, cualquier proposición se puede agregar a un estado existente para formar un nuevo estado.

Para evitar la confusión, utilizaremos *Holds*(s, t) para denotar que el estado *s* es verdadero en un cierto momento *t*, de tal manera que:

(6.3.4) $\text{Holds}(s, t) \Leftrightarrow \forall f(\text{Belongs}(f, s) \Rightarrow \text{Holds}(f, t))$

Los conceptos de cambio y de tiempo están profundamente relacionados puesto que los cambios son causados por los eventos que ocurren en el tiempo. Para expresar que los eventos (denotados por e) ocurren, siguiendo el enfoque de Allen (Allen, 1984), utilizamos *Occur*(e, t) (*Ocurre*(e,t) en español) para denotar que el evento *e* ocurre en un cierto momento *t*, e imponemos el axioma siguiente:

(6.3.5) $\text{Occur}(e, t) \Rightarrow \forall t'(\text{In}(t', t) \Rightarrow \neg \text{Occur}(e, t'))$

Además, utilicemos la fórmula *Changes*(t_1, t, t_2, s_1, e, s_2) (*Se_ cambia* en español) para expresar la ley causal, el cual intuitivamente declara que, bajo la condición previa de que el estado s_1 se cumple en un cierto momento t_1, la ocurrencia en un cierto momento *t* del evento *e* cambiará el mundo del estado s_1 al estado s_2, que se sostiene en el momento t_2. Formalmente, imponemos el axioma siguiente sobre la causalidad para asegurarnos que si la condición previa de una ley causal se sostiene y el evento sucede, después debe aparecer el efecto causado esperado:

(6.3.6) $\text{Changes}(t_1, t, t_2, s_1, e, s_2) \wedge \text{Holds}(s_1, t_1) \wedge \text{Occur}(e, t)$
$\Rightarrow \text{Holds}(s_2, t_2)$

Para caracterizar relaciones temporales entre los eventos y sus efectos, imponemos las restricciones temporales siguientes:

(6.3.7) $\text{Changes}(t_1, t, t_2, s_1, e, s_2)$
$\Rightarrow \text{Meets}(t_1, t) \wedge (\text{Meets}(t_1, t_2) \vee \text{Before}(t_1, t_2))$

Es importante notar que el axioma (6.3.7) representa realmente lo que se llama la mayor restricción temporal general (*general temporal constraint, GTC*) (Shoham, 1987; Terenziani and Torasso, 1995). La *GTC* garantiza el cumplimiento de la aserción de sentido común: "*el principio de un efecto no puede preceder el principio de su causa*".

Hay de hecho 8 relaciones de orden temporal posibles entre t_1, t y t_2 que satisfacen (6.3.7). Éstos se presentan en Ilustración 4.

- Caso (A): donde el efecto llega a ser verdadero inmediatamente después del final del evento y sigue siendo verdadero por un cierto tiempo después del evento. Por ejemplo, el evento de poner un libro en la mesa tiene el efecto que el libro está en la mesa inmediatamente después que se termina el evento.
- Caso (B): donde el efecto se lleva a cabo solamente al mismo tiempo que el evento está en progreso. Por ejemplo, el efecto del evento de tocar la bocina de un coche, la bocina emite sonido solamente mientras se la están presionando.
- Caso (C): donde el principio del efecto coincide con el principio del evento, y el efecto termina antes que el evento termine. Por ejemplo, considere el caso donde una mina es colocada en la primera mitad de un puente. Si alguien está caminando a través del puente, él estará en el peligro solamente en la primera mitad del puente.

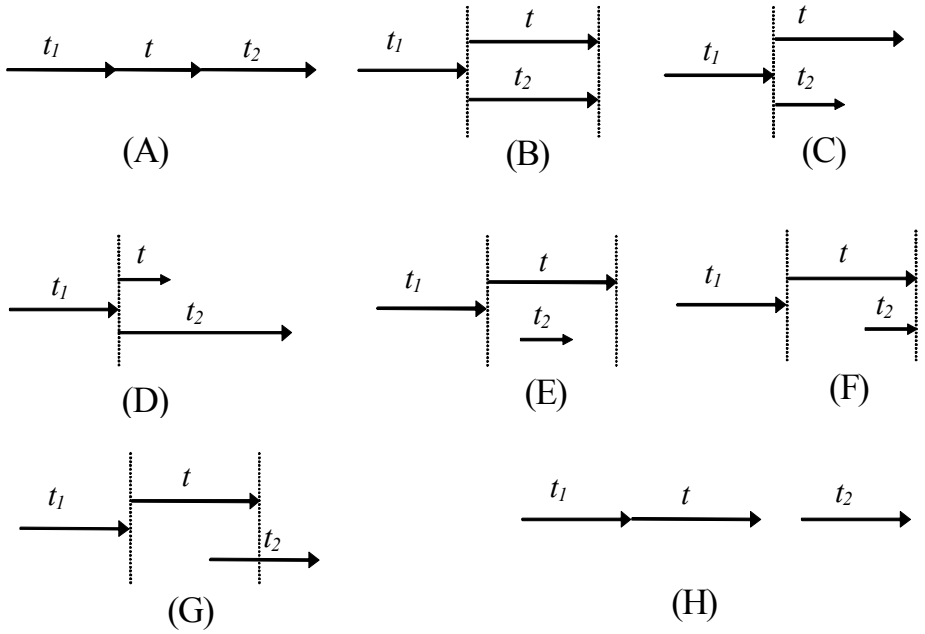


Ilustración 4. Relaciones de orden temporal posibles.

- Caso (D): donde el principio del efecto coincide con el principio del evento, y el efecto sigue presente por un cierto tiempo después del evento. Por ejemplo, el efecto del evento de tocar un timbre en la puerta (digamos, por un segundo), puede durar varios segundos (es decir, el timbre sigue sonando cuando el botón del timbre ya no está presionado).

- Caso (E): donde el efecto se mantiene solamente un cierto tiempo durante el evento. Por ejemplo, el corredor que siente un cansancio extremo durante el quincuagésimo minuto del evento de correr por cuatro horas.
- Caso (F): donde el efecto se presente durante el evento y sigue siendo presente hasta que el evento se termina. Por ejemplo, considere el evento de quitar un poco de agua de una vasija levantando un lado de la vasija. En el caso de cuando la vasija no esté llena, el efecto de que el agua sale corriendo hacia afuera solamente después de que se levante el borde de la vasija, y el agua quedará fluyendo hacia afuera hasta que el evento termine.
- Caso (G): donde el efecto se presenta durante el progreso del evento y sigue presente por un cierto tiempo después del evento. Por ejemplo, el corredor que se siente cansado por días después del trigésimo minuto del evento por correr la pista por tres horas.
- Caso (H): donde hay retraso entre el evento y su efecto. Por ejemplo, 25 segundos después de que el botón de un semáforo se presiona (para solicitar el cruce de la calle), la luz peatonal se cambia a amarillo; y después de otros 5 segundos, se cambia a verde.

Según lo mencionado, varias teorías se han propuesto para representar y razonar sobre acciones/eventos y los cambios. Sin embargo, las relaciones causales temporales entre los eventos y sus efectos según se especifica en la mayor parte de los formalismos existentes son limitadas. Una excepción es la teoría relativamente general del Tiempo y Tipos de Acciones, y de la Causalidad, introducida por Terenziani y Torasso en el medio de los años 90s (Terenziani and Torasso, 1995). En lo que sigue, demostraremos brevemente que las relaciones causales presentadas aquí son más generales que las de Terenziani y Torasso, y por su versatilidad para describir todos los casos encontrados en la literatura.

Realmente, si t y t_2 se especifican como puntos, un punto y un intervalo, un intervalo y un punto, o dos intervalos, respectivamente, aplicando la Relación de Restricción Temporal (*Temporal Relation Constraint, TRC*) según describimos en la sección 2.3, podemos obtener directamente los cuatro teoremas siguientes:

(Th1) $\text{Changes}(t_1, t, t_2, s_1, e, s_2) \wedge \text{Dur}(t) = 0 \wedge \text{Dur}(t_2) = 0$
$\Rightarrow \text{Equal}(t, t_2) \vee \text{Before}(t, t_2)$

Es decir, si el evento y el efecto se representan como los puntos, entonces el evento precede (estrictamente) al efecto, o ellos coinciden uno con el otro (es decir, suceden simultáneamente en el mismo punto del tiempo).

(Th2) $\text{Changes}(t_1, t, t_2, s1, e, s_2) \wedge \text{Dur}(t) = 0 \wedge \text{Dur}(t_2) > 0$
$\Rightarrow \text{Starts}(t, t_2) \vee \text{Meets}(t, t_2) \vee \text{Before}(t, t_2)$

Es decir, si el evento es un punto y el efecto es un intervalo, entonces el evento precede (inmediatamente o estrictamente) al efecto, o el evento coincide con el principio del efecto.

(Th3) $\text{Changes}((t_1, t, t_2, s_1, e, s_2) \wedge \text{Dur}(t) > 0 \wedge \text{Dur}(t_2) = 0$
$\Rightarrow \quad \text{Started-by}(t, t_2) \vee \text{Contains}(t, t_2)$
$\vee \text{Finished-by}(t, t_2) \vee \text{Meets}(t, t_2) \vee \text{Before}(t, t_2)$

Es decir, si el evento es un intervalo y el efecto es un punto, entonces el evento precede (inmediatamente o estrictamente) al efecto, o el efecto coincide con el principio o el final del evento, o el efecto sucede durante el evento.

(Th4) $\text{Changes}((t_1, t, t_2, s_1, e, s_2) \wedge \text{Dur}(t) > 0 \wedge \text{Dur}(t_2) > 0$
$\Rightarrow \text{Started-by}(t, t_2) \vee \text{Contains}(t, t_2) \vee \text{Finished-by}(t, t_2)$
$\vee \text{Equal}(t, t_2) \vee \text{Starts}(t, t_2) \vee \text{Overlaps}(t, t_2) \vee \text{Meets}(t, t_2)$
$\vee \text{Before}(t, t_2)$

Es decir, si el evento y el efecto son intervalos, entonces el principio del evento precede (inmediatamente o estrictamente) o coincide con el principio del efecto, donde el final del evento puede preceder (inmediatamente o estrictamente), coincidir, o ocurrir después (inmediatamente o estrictamente) del final del efecto.

Es fácil observar que (Th1) y (Th4) son equivalentes a los teoremas 4 y 1 de Terenziani y Torasso, respectivamente, mientras que (Th2) y (Th3) pueden ser vistos como la extensión al teorema 3 y teorema 2 de Terenziani y Torasso, respectivamente (Terenziani y Torasso, 1995). Esto es debido al hecho que, mientras que la relación "Meets" entre un evento representado por un punto y un efecto representado por un intervalo, y entre un evento representado por un intervalo y un efecto representado por un punto, se acomodan perfectamente en (Th2) y (Th3), los teoremas 3 y 2, respectivamente, de Terenziani y Torasso no permiten tales relaciones.

De hecho, según el teorema 3 de Terenziani y Torasso, o debe haber un hueco entre la causa representada por un punto y su efecto representado por un intervalo, o la causa representada por un punto debe coincidir con el principio del intervalo que corresponde a su efecto. En otras palabras, el intervalo donde aparece el efecto debe estar "Después de" o "Cerrado en" el punto en el cual sucede la causa. Por lo tanto, el caso donde una causa representada por un punto "*Meets* (*Se encuentra*)" con su efecto representado por un intervalo (es decir, el intervalo donde sucede el efecto es "Abierto" en el punto en el cual la causa sucede) no está permitido. Sin embargo, considere el ejemplo siguiente:

Inmediatamente después de que la energía fue encendida, un robot que había permanecido inmóvil comenzó a moverse.

Si utilizamos $s_{Stationary}$ para representar el estado que "*el robot estuvo inmóvil*", $e_{SwitchOn}$ para representar el evento "*la energía fue encendida*", y s_{Moving} para representar el efecto correspondiente al "*el robot se movía*", entonces

Changes($t_{Stationary}$, $t_{SwitchOn}$, t_{Moving}, $s_{Stationary}$, $e_{SwitchOn}$, s_{Moving})

debe ser consistente con:

Meets($t_{SwitchOn}$, t_{Moving})

Es decir, del punto "Encendido" $t_{SwitchOn}$ es seguido inmediatamente por el intervalo "Moverse", pero no se incluye en este intervalo. En otras palabras, el robot se movía inmediatamente después del punto "Encendido" $t_{SwitchOn}$, pero en el punto cuando la energía fue encendida, el robot no se movía. Obviamente, este escenario no puede expresarse usando el teorema 3 de Terenziani y Torasso.

Similarmente, en el teorema 2 de Terenziani y Torasso, el caso donde un evento representado por un intervalo "*Meets (Se encuentra)*" con su efecto representado por un punto (es decir, el intervalo donde sucede la causa es "Abierto" en el punto en el cual sucede el efecto) no está permitido. Entonces nuevamente, considere el ejemplo siguiente:

Inmediatamente después de que la bola estaba cayendo hacia abajo en el aire, la bola tocó la tierra.

Si utilizamos s_{InAir} para representar la condición previa que "*la bola estaba en cierta posición en el aire*", $e_{FallingDown}$ para representar el evento "*la bola estaba cayendo*", y $s_{TouchGround}$ para representar el efecto que "*la bola tocó la tierra*", respectivamente, entonces

Changes(t_{InAir}, $t_{FallingDown}$, $t_{TouchGround}$, s_{InAir}, $e_{FallingDown}$, $s_{TouchGround}$)

debe ser consistente con:

Meets($t_{FallingDown}$, $t_{TouchGround}$)

Es decir, el intervalo cuando la bola estaba cayendo hacia abajo, es seguido inmediatamente por el punto, cuando la bola tocó la tierra, pero no incluye este punto. En otras palabras, la bola estaba cayendo hacia abajo inmediatamente antes del instante en que tocó la tierra, pero en el momento cuando la bola tocó la tierra, la bola ya no estaba cayendo. Una vez más, tal escenario no es permitido por el teorema 2 de Terenziani y Torasso (1995).

7. Conclusiones

En este capítulo, hemos tratado algunos asuntos fundamentales relacionados con teorías y modelos temporales, y hemos presentado algunas consideraciones ontológicas relacionadas con las primitivas temporales y las

relaciones de orden. También hemos proporcionado una descripción de los tres enfoques principales de la representación de la información temporal en el dominio de la Inteligencia Artificial, y hemos introducido una gama más amplia de meta-predicados y una clasificación más fina de proposiciones temporales que en otros enfoques. Esperamos que este capítulo ayude al lector a desarrollar un interés especial en la representación y el razonamiento temporal.

8. Ejercicios propuestos

Ejercicio 1:

Exprese las siguientes frases en lenguaje natural como proposiciones de la lógica de predicados de primer orden estándar.

1. El calentador no fue encendido **en cada momento**.
2. El calentador no fue encendido **en cualquier momento**.
3. Había **algunos momentos** cuando el calentador no fue encendido.

Ejercicio 2:

Exprese las tres proposiciones del Ejercicio 1 en términos de la lógica temporal materializada según lo descrito en la Sección 3.3.

Ejercicio 3:

Demuestre

$$\text{HoldsON}(f, t) \Leftrightarrow \text{HoldsON}_P(f, t) \wedge \text{HoldsON}_I(f, t)$$

Ejercicio 4:

Por definición (C26), una proposición f es gestalt si

$$\text{Holds}(f, t_1) \wedge \text{In}(t_2, t_1) \Rightarrow \neg\text{Holds}(f, t_2).$$

Es decir, si una proposición gestalt f se cumple en un intervalo I, entonces f no se cumple en cualquier subintervalo propio de I. Usando esta propiedad, demuestre que si una proposición gestalt f se cumple sobre un intervalo I, entonces f no se cumple sobre cualquier I' tal que I es un subintervalo propio de of I'.

Ejercicio 5:

Demuestre que de

$$\text{HoldsON}(f, t) \Rightarrow \forall t_1(\text{In}(t_1, t) \Rightarrow \text{Holds}(f, t_1))$$

se puede deducir

$$\text{HoldsON}(f, t) \Rightarrow \ \forall t_1(\text{Part}(t_1, t) \Rightarrow \exists t_2(\text{Part}(t_2, t_1) \wedge \text{Holds}(f, t_2)))$$

Ejercicio 6:

Usando la definición C34, demuestre que:

La proposición f_1 es incompatible con la proposición f_2 si y solamente si la proposición f_2 es incompatible con la proposición f_1.

Ejercicio 7:

Usando ejemplo(s) para ilustrar el fenómeno de *Cambio continuo* (*Intermingling*), muestre que existe la posibilidad del cambio infinito de valor booleano de una proposición sobre un intervalo con la duración finita.

Ejercicio 8:

La Sección 5 de este capítulo contiene una categorización extensa de las proposiciones temporales. Como un descubrimiento científico, intente encontrar algunos fenómenos temporales que se puede clasificar usando este esquema de categorización, y, por lo tanto, se necesita su ampliación a otras categorías.

Referencias

[1] Allen J. F. (1981). An interval-based representation of temporal knowledge, *Proc. 7th Int. Joint Conf. on AI*, 1981, pp.221-226.

[2] Allen J. F. (1983). Maintaining Knowledge about Temporal Intervals, *communication of ACM.*, Nov. 1983, Vol.26, pp.123-154.

[3] Allen J. (1984). Towards a General Theory of Action and Time, *Artificial Intelligence*, 23, 123-154.

[4] Allen J. and Hayes J. (1985), A Common-Sense Theory of Time, *Proceedings of the 9th IJCAI*, 528-531.

[5] Allen J. F. and Hayes P. J. (1989). "Moments and Points in an Interval-based Temporal-based Logic", *Computational Intelligence*, Vol.5, No.4, pp.225-238.

[6] Bacchus F. (1991) Tenenberg J. and Koomen J. A.: "A non-substancializada temporal logic", *Artificial Intelligence*, 52, pp.87-108.

[7] van Benthem, J. (1983). *The Logic of Time*, Kluwer Academic, Dordrech.

[8] van Beek P. V. (1992). "Reasoning About Qualitative Temporal Information", *Artificial Intelligence*, 58, pp.297-326.

[9] Bruce B. C. (1972). "A Model for Temporal References and Application in a Question Answering Program", *Artificial Intelligence*, 3, pp.1-25.

[10] Davidson D. (1967). The logical form of action sentences. In Nicholas Rescher (ed) *The logic of Decision and Action.* University of Pittsburgh Press.

[11] Davidson D. (1969). The Individuation of Events. In Nicholas Rescher (ed) *Essays in Honor of Carl G. Hempel.* Dordrech, D. Reidel.

[12] Funk K. H. (1983). "Theories, Models, And Human_Machine Systems", *Mathematical Modelling*, Vol.4, pp.567-587.

[13] Gabbay, D. (1989). The Declarative Past and Executable Future. *Temporal Logic in Specification: Altrincham Workshop 1987 (Lecture Notes in Computer Science)*, 398, 409-448.
[14] Galton A. (1990). "A Critical Examination of Allen's Theory of Action and Time", *Artificial Intelligence*, 42, pp.159-188.
[15] Galton A. P. (1991). Substancializada temporal theories and how to unreify them. *Proceedings of IJCAI'91*, 1177-1182.
[16] Galton A. (1996). An Investigation of Non-intermingling Principles in Temporal Logic. *Journal of Logic and Computation*, 6(2), 267-290.
[17] Haugh B. A. (1987). Non-Standard Semantics for the Method of Temporal Arguments. *Proceedings of 10th IJCAI* **1:** 449-455.
[18] Halpern J. Y. and Shoham Y. (1991). A Proposicional Model Logic of Time Intervals. *Journal of the Association for Computing Machinery*, 38(4), 935-962.
[19] Hardie P. and Gaye K. (1930) *Physica* (The Works Of Aristotle, translated into English, editor: W. D. Ross, Volume II), The Clarendon Press, Oxford.
[20] Hayes P. and Allen J. (1987), Short time periods, *Proceedings of the 10th IJCAI*, Milan, Morgan Kaufmann Publishers, San Francisco, 981-983.
[21] Knight. B. and Ma. J. (1992). "A General Temporal Model Supporting Duration Reasoning", *AI Communication Journal*, Vol.5(2), pp.75-84.
[22] Kowalski R. A. and Sergot M. J. (1986). "A logic-based calculus of events", *New Generation Computing*, 4, pp.67-95.
[23] Kripke S. (1963). Semantical considerations on modal logic. *Acta Philosophic, Fennica*, 16**,** 83-94.
[24] Ladkin P. (1987), Models of axioms for time intervals, *Proceedings of the 6th National Conference on Artificial Intelligence*, 234-239.
[25] Ladkin P. (1992), Effective solutions of qualitative intervals constraint problems, *Artificial Intelligence*, 52, 105-124.
[26] Lifschitz V. (1987). A theory of action. *Proceedings of 10th IJCAI*, 966-972.
[27] Ma J and Hayes P. (2006). "Primitive Intervals Vs Point-Based Intervals: Rivals Or Allies?", *the Computer Journal*, Vol.49(1), 32-41.
[28] Ma J. and Knight B. (1994). "A General Temporal Theory", *The Computer Journal*, 37(2), 114-123.
[29] Ma J. and Knight B. (1996). "A Substancializada Temporal Logic", *The Computer Journal*, 39(9), 800-807.
[30] Ma J. and Knight B. (2001). "Substancializada Temporal Logics: An Overview", *Artificial Intelligence Review*, Vol.15, 189-217.
[31] Ma J. and Knight B. (2003), Representing The Dividing Instant, *the Computer Journal*, 46(2), 213-222.
[32] Maiocchi R. (1992) Automatic Deduction of Temporal Information. *ACM Transactions on Database Systems*, 4, 647-688.
[33] McArthur R. (1976). *Tense Logic*. Reidel, Dordrecht.
[34] McCarthy J. and Hayes P. J. (1969). "Some Philosophical Problems from the Standpoint of Artificial Intelligence", *Machine Intelligence*, B. Meltzer and D. Michie, (eds.), 4, Edinburgh U.p., pp.463-502.
[35] McDermott D. V. (1982). "A Temporal Logic for Reasoning about Processes and Plans", *Cognitive Science*, 6, pp.101-155.
[36] Prior A. (1955). Diodoram modalities. *Philosophical Quarterly*, 5, 205-213.
[37] Pnueli A. (1977). The temporal logic of programs. *Proceedings of 8th. IEEE Symp. On Foundations of Computer Science*: 46-67.

[38] Reichgelt H. (1989). A comparison of first-order and modal logics of time. In Jackson P. and van Harmelen H. R. F. (eds), *Logic-based knowledge representation*, 143-176.
[39] Rescher J. & Urquhart A. (1971). *Temporal Logics*. Springer Verlag.
[40] Sadri F. (1987). "Three Recent Approaches to Temporal Reasoning", *Temporal Logic and their Applications*, ed. Galton A., Academic Press, pp.121-168.
[41] Shanahan M. (1995). A Circumscriptive Calculus of Events, *Artificial Intelligence*, Vol. 77, 29-384.
[42] Shoham Y. (1987). "Temporal Logics in AI: Semantical and Ontological Considerations", *Artificial Intelligence*, 33, pp.89-104.
[43] Suppes P. (1961). "A Comparison of the Meaning and Uses of Models in Mathematics and the Empirical Sciences", in *The Concept and Role of the Model in Mathematics and Natural and Social Sciences*, D. Reidel Publishing Company.
[44] Terenziani P. and Torasso P. (1995). Time, Action-types, and Causation: an Integrated Analysis, *Computational Intelligence*, 11(3), 529-552.
[45] Vila, L. (1994), A survey on temporal Reasoning in Artificial Intelligence, *AI Communication*, 7, 4-28.
[46] Vilain M. V. (1982). "A System for Reasoning about Time", *Proc. AAAI-82*, Pittsburgh, PA. pp 197-201.
[47] Vilain M. B. and Kautz H. (1986). "Constraint Propagation Algorithms for Temporal Reasoning", *Pro. AAAI-86*, 1986, pp.377-382.
[48] Warner R. (1963). *The Confessions of St. Augustine*. New York: Penguin.
[49] Whitehead A. (1929), *Process and Reality*. Cambridge University Press, Cambridge.

Capítulo 11.

Robótica basada en el comportamiento

1. Elementos de la robótica basada en el comportamiento

Desde un punto de vista bioinspirado es importante considerar los mecanismos subyacentes en animales que que tienen que sobrevivir en ambientes dinámicos altamente cambiantes. Es por esto que una llave en la resolución de tareas desde un enfoque del tipo: *no pienses, actua*. Es por esto que a un nivel intermedio el enfoque basado en el comportamiento observa la evolución del mundo para poder disparar una serie de patrones que forman subcomportamientos. Es entonces que un mecanismo de arbitraje resulta necesario para interpretar información sensora del mundo y niveles internos del animal robot *animat* a modelar. Este capítulo aborda temas relacionados con este enfoque robótico.

1.1. Características del enfoque basado en el comportamiento

El problema de selección de acción esta relacionado con el uso compartido de recursos limitados. Donde varios subsistemas tratan de ganar control de estos recursos. Un ejemplo simple de esta situación es la que se encuentra en un centro de cómputo cuando varias computadoras tratan de usar la impresora al mismo tiempo. Es por esto que es necesario un protocolo para garantizar que la impresiones de todos los usuarios se lleven a cabo correctamente. En el problema de selección de acción es un mecanismo de selección de acción el que facilita que los subcomportamientos sean elegidos uno a la vez. Sin embargo es posible que más de un comportamiento sea elegido, como es bien sabido *todos podemos caminar y masticar chicle al mismo tiempo*. Del mismo modo es posible que dos acciones que no sean incompatible por su uso compartido de los recursos motores sean observadas al mismo tiempo. Por ahora es mejor asumir que solamente una acción puede ser realizada a la vez.

Actualmente la Inteligencia Artificial, que de algún modo alberga a la robótica es una ciencia que comparte una serie de elementos que la han convertido en un enfoque multidisciplinario. Tomando prestadas y contribuyendo de vuelta a otras

areas del conocimiento. Del mismo modo sucede con la robótica. Y por sobre todo la robótica basada en el comportamiento toma prestadas ideas provenientes de la etología, la psicología y las neurociencas por mencionar algunas. Lo que sigue a continuación es una breve introducción a algunos rasgos que estas disciplinas tienen en común.

Conducta Animal (Etología, Psicología, Neurociencias) Dentro de la conducta animal resulta importante mencionar que el problema de selección de acción es identificado dentro de la etología donde este problema se conoce como la conmutación de conductas. Generalmente los etólogos estudian a los animales en su medio ambiente de preferencia tratando de pasar inadvertidos. Sin embargo, queda a criterio del observador los subcomportamientos y los momentos en los que se activan. Resulta difícil no abundar en detalles al momento de la observación y queda siempre a criterio del observador los rasgos identificados. Cabe mencionar que los primeros desarrollos en Inteligencia Artificial se basan en gran parte en la introspección, es por esto que el humano tiende a buscar rasgos inteligentes en los animales basandose en una definición de atropomórfica de inteligencia. Por ejemplo sería natural pensar que cuando estamos hambrientos y pasamos cerca de un estímulo relacionado con saciar el hambre, esta hambre pueda incrementarse. En parte esta apreciación es correcta, pero para un observador marciano que no conociera las costumbres de la tierra, éste podría identificar la activación de la conducta *alimentarse* con la acción realizada por un humano cuando abre el refrigerador; sin embargo podría suceder que el humano solamente buscaba matar el tiempo checando que había en el refrigerador. De igual modo podría pasarle a un observador humano tratando de identificar comportamientos animales.

Es evidente que no podemos dejar al observador fuera de estas apreciaciones, de hecho en la robótica basada en el comportamiento resulta práctico considerar la modelación de un animat como el producto conjunto del observador, el ambiente, el animat y su supuesto estado interno. De este modo como menciona Dennet [1], si no podemos estudiar una iguana completa, porque no construir una iguana marciana con tres ruedas. Sin importar que esta tenga ruedas, en lugar de piernas, o una cámara CCD [1] en lugar de ojos, etc. Por lo menos a una iguana marciana la podemos diseccionar, abrir, estudiar, modificar y cualquier otro experimento que se nos ocurra con esta iguana artificial. Este es uno de los grandes enfoques que se siguen en la robótica basada en el comportamiento. Es decir hay que ver que resulta de construir animales lo más parecidos a los reales y también de ser posible tratar de incorporar mecanismos similares a los que pudieran existir dentro de los animales vivos.

Por lo tanto es necesario considerar no solamente la explicación que podemos dar del comportamiento animal por medio de la observación. Sino también de-

[1] Charge-Coupled Device, una cámara con un dispositivo de cargas (eléctricas) interconectadas, el CCD en este caso es el sensor con diminutas células fotoeléctricas que registran la imagen.

sarrollar modelos que expliquen en términos de procesamiento de la información como es que un animal podría percibir su ambiente y como en base a estas percepciones puede tomar decisiones. Un aspecto que es importante señalar es que muchos modelos de comportamiento animal, son modelos que no son implantables directamente. Por ejemplo el modelo psicohidráulico de Lorenz [2]. Por otra parte resulta importante entender y conocer como internamente el cerebro de los animales funciona. Aún más importante es poder trazar paralelos de como funciona el cerebro de animales con características comunes. Es por esto que el uso de iguanas marcianas, o cualquier otro animal que se nos ocurra. Pensemos en una rata marciana de aquí en adelante. Por ejempo el uso de ratas marcianas con cerebros de ratas marcianas nos ayuda a comprender la inteligencia derivada del comportamiento animal. Es importante señalar que no estamos tratando de hacer pasar un robot por una rata al ponerle un disfraz (Fig. 1). Estamos tratando de construir una rata robot que no solamente comparta algunas características físicas lejanamente parecidas a las de su igual real, que resuelva una tarea animal y que además tenga un selector de acciones como el de una rata real. El uso del disfraz de rata tampoco esta descartado, la idea no consiste en hacer crear únicamente la ilusión lo más real posible. Sino que lo más importante consiste en acortar la distancia, entre el animat y el animal real. Si se considera esto descabellado, piense en los torneos robóticos de futbol soccer, donde se espera que algún día hombres y máquinas puedan jugar futbol juntos.

Figura 1. El contar con un robot lo más parecido a una rata por el momento no es posible. Sin embargo, el tener una perspectiva abierta de hasta donde puede nuestro robot simular una rata, ayuda enormemente en el desarrollo de un animal robótico que cada vez se acerque más a su contraparte real.

Es importante senãlar que existe la firme certeza, fundamentada por varios tipos de experimentos de que existe un mecanismo de selección de acción en todos los vertebrados. Este mecanismo es conocido como los Ganglios Basales y una descripción de su funcionamiento y sus características excede el alcance

de este capítulo. Sin embargo, es importante conocer que una disfunción en este conjunto de estructuras esta relacionado con la enfermedad de Parkinson. En esta enfermedad existe la dificultad de realizar movimientos voluntarios complejos. En otras palabras el inicio de rutinas que definen subcomportamientos no se esta activando. Existe una implementación robótica de los Ganglios Basales, para mayor información el lector debe referirse al trabajo reportado en [3] [4] [5]. Otro punto que es importante señalar, es que la coordinación de movimientos finos se lleva a cabo en el Cerebelo. Los Ganglios Basales y el Cerebelo están muy relacionados entre sí y de alguna manera también abarcan la selección de acción. En este capítulo nos avocaremos a tomar como cierta la afirmación de la existencia de un selector en el cerebro vertebrado. Lo cual nos va a llevar a poder modelar una rata robot lo más realmente posible. A continuación abordaremos el problema de la resolución de tareas robóticas desde el punto de vista de la robótica que asume que el mejor modelo interno que un robot puede tener del mundo, es el mismo mundo en el que se desnvuelve, es decir la interpretación que percibe de este a través de sus sensores.

Conducta Robótica (Reactiva) Inicialmente en el desarrollo de la robótica se consideraba que esta dependia del desarrollo de modelos resultantes de la introspección. Donde el robot cuenta con un mecanismo de procesamiento de información para la planeación de acciones por realizar. Cabe señalar que este enfoque se basada como lo resume Maja Mataric [6] en la premisa *piensa, entonces actua*. Posteriormente hubo muchos investigadores que se plantearon la pregunta de porque no hacer otra cosa como *no pienses, reacciona*. El enfoque basado en el comportamiento es un enfoque que esta situado entre el primer enfoque conocido como deliberativo, y el segundo conocido como reactivo. Al enfoque basado en el comportamiento, Mataric lo resume como *piensa la manera que estas actuando*. Sin embargo un enfoque verdaderamente intermedio, sería uno que ofrezca una solución comprometida entre el enfoque deliberativo y el reactivo como un enfoque híbrido resumido por Mataric como *paralelamente piensa y actua independientemente*.

Existen muchos ejemplos de arquitecturas que basan su desarrollo en el enfoque reactivo. Por ejemplo, la arquitectura subsumción de Rodney Brooks [7], donde no existe una jerarquía explícitamente definida de conductas, donde una conducta puede tener mayor prioridad que otra que ya esta en ejecución. A esta característica Brooks la identifica como la sumbsumción de una conducta por otra. Por otra parte otro ejemplo es el trabajo de Pattie Maes [8], donde cada conducta esta relacionada con otra por medio de relaciones sucesor, predecesor y en conflicto. Cada conducta puede ejecutarse después de que otra sustenta la condiciones necesarias para activación de la primera. De este modo una de las conductas tendrá que esperar a que otra ponga las condiciones suficientes. Existiendo un flujo de activación desde una conducta a otra. Sin embargo existen conductas rivales cuya ejecución previene la selección de una que este dispuesta a ejecutarse. El cálculo de cuando y cual de estos módulos conductuales debe

ejecutarse, en el modelo de selección de acción de Maes se resuelve por medio diversas ponderaciones y parámetros difíciles de calibrar. Sin embargo la selección se realiza correctamentamente una vez que se han establecido estos parámetros. Es necesario resaltar que la selección de acción puede ser difícil de identificar en una propuesta puramente reactiva. En este caso la solución del problema emerge de la interacción interna y externa del modelo reactivo. A continuación este tema se aborda con mayor enfásis en su relación con el problema de conmutación de conductas.

1.2. Visión modular y emergente del modelado de conductas

La emergencia de comportamientos adecuados a través de la interacción de un modelo puede no ser la solución más viable. No importando que uno no pueda localizar el sitio exacto donde se toma una decisión particular. Tarde o temprano todos los detalles del desarrollo de un modelo, pueden escapar de nuestro control, ocasionando fantasmas en las máquinas. Visto de otra manera anomalías que puedan o no ser facilmente detectables. Sin embargo que pasa cuando estos fantasmas actuan en nuestro favor, entonces se tienen características especiales que no necesariamente fueron colocadas por un diseñador. En este sentido la evolución natural es un mejor diseãdor porque no teme explorar soluciones poco ortodoxas. En otras palabras, la emergencia de una solución resulta deseable siempre que podamos repetir el proceso por el cual se produjo. De otra manera tendríamos modelos completamente predecibles y muchos de los problemas en Inteligencia Artificial estarían ya resueltos. La emergencia de conductas es deseable, pero las conductas no se forman de la nada. Existe una serie de conocimientos, prealambrados, innatos o simplemente ya asimilados que permiten que adquiramos destreza en la resolución de tareas específicas. En este sentido la visión puramente reativa y emergente se auxilia de la existencia de patrones conductuales almacenados en algún lado bajo un modelo de selección de acción basado en el comportamiento. Además de que un modelo que tiene el mecanismo de selección separado de los módulos conductuales puede ofrecer mayor robustez y extensibilidad. Este último punto se desarolla a continuación.

1.3. Ventajas del enfoque basado en el comportamiento

Una de las principales ventajas de usar un enfoque basado en el comportamiento. Es el contar con un modelo intermedio de conductas que es capaz de reajustar sus parámetros para llevar a cabo la tarea que se esta proponiendo resolver. En este caso de usar conductas como módulos independientes que pueden ser modificables y además extensibles. Además el enfoque basado en el comportamiento permite acercarnos un poco al desarrollo de inteligencia en animales vivos. Esto facilita la construcción de animales robots. En nuestro caso es importante poder contar con robots que puedan resolver tareas que resuelven los seres vivos. Un ejemplo común de la modelación son conductas de forrajeo y

conductas sociales. Estas conductas sociales pueden ser cooperativas o competitivas. En cualquiera de los dos casos robots heterogénos compiten entre sí o se apoyan para resolver una tarea.

Una distracción típica en Inteligencia Artificial considera que la inteligencia puede formarse como una escalera. Donde a medida que subimos un peldaño incrementalmente podremos aumentar la inteligencia de nuestros seres robóticos; sin embargo esto no es del todo correcto. La evolución y la naturaleza en general trabaja por medio de inhibir aumentar o disminuir las capacidades de centros especilizados que proveen funcionamientos especificos para el desarrollo de mecanismos particulares de inteligencia. Además de que en el cerebro en general existe una gran redundancia que permite que en caso de un malfuncionamiento o una falla en partes del cerebro, este pueda recuperarse hasta donde sea posible. Es natural suponer que algunas capacidades realmente nunca se recuperan. El desarrollo de modelos bioinspirados basados en el comportamiento, permite el acercamiento a entender los mecanismos subyacentes de la inteligencia animal.

2. Elementos de la robótica evolutiva

La modelación de esquemas de resolución robótica requiere del ajuste de parámetros o incluso la modelación completa de comportamientos específicos. El uso de Redes Neuronales es una solución muy popular para el desarrollo de comportamientos de forrajeo. Incluso de comportamientos sociales, principalmente competitivas. De algún modo en este caso el ajuste de parámetros en la red neuronal son los pesos que reflejan la fuerza de las conexiones de los datos de entrada y su procesamiento interno. Este punto no se contrapone con el diseño de un mecanismo de selección de acción bajo el enfoque basado en el comportamiento. De igual manera, el ajuste de parámetros de un mecanismo de selección de acción bajo el esquema evolutivo. Dentro del enfoque generalizado de evolución artificial, existen varios enfoques. Estos son los siguientes: *Algoritmos Genéticos, Estrategias Evolutivas, Programación Evolutiva y Genética y Coevolución* [9]. Cada uno de estos enfoques presenta características particulares que ofrecen ventajas y desventajas para problemas específicos. Por otra parte el enfoque basado en un algoritmo genético puede ser visto como una solución generalizada a un problema que requiere optimización. Sin embargo, como se mencionó anteriormente la Inteligencia Artificial y de algún modo la robótica bionspirada enfocada en el comportamiento comparte este punto de vista multidisciplinario. Es por esto que la combinación de varios elementos para promever la emergencia de Inteligencia Artificial comulga con las ideas de este tipo de enfoque.

2.1. Identificación de los elementos a evolucionar

En el desarrollo de un mecanismo de selección de acción es importante identificar los elementos a evolucionar. Por una parte se ha mencionado que un

sistema robótico en parte depende del estado interno, el mundo externo y el observador. Sin embargo el observador también tiene presencia al momento del diseño de las conductas a modelar y de algunos elementos en la selección. Es decir el diseñador humano tiene que observar y elegir los aspectos a modelar. Existen elementos que necesitan ser optimizados y otros que por su naturaleza puede buscarse una solución práctica para su realización. Por ejemplo, existen conductas que requieren una repetición progresiva de movimientos específicos siempre en el mismo orden. El brazo de un robot móvil proporciona una idea clara de esto. En el caso más simple el brazo siempre baja al frente donde es posible ubicar el sitio de la recolección, entonces el brazo se cierra y si la alineación con el objeto fue correcta y no hubo movimientos ni deslizes el objeto será aprisionado en la garra robótica. Posteriormente el brazo se levanta, esta serie de acciones siempre se realiza en el mismo orden. Resulta poco práctico tratar de optimizar esta situación a menos que se tenga un control preciso del brazo y pueda recogerse objetos al vuelo, es decir en movimiento. En un caso práctico es mejor realizar siempre la misma repetición en el mismo orden y con un numero fijo de iteraciones. Son situaciones como estas donde una rutina de programación fija puede realizar siempre este trabajo de una mejor manera.

Por lo anteriormete descrito, es mejor optimizar aquellas características que ofrecen mayor viabilidad. En nuestro caso, se ha decidido optimizar las conductas y los parámetros de selección. La manera de realizar este ajuste del modelo de selección se ha realizado de manera conjunta y secuencial como se explica en los siguientes párrafos.

2.2. El diseño de la función de calidad

Un Algoritmo Genético (AG) [10] es una técnica de ascenso del gradiente, lo que significa que el AG busca en un espacio de soluciones una que sea óptima. En otras palabras si se graficara la calidad de cada configuración que ofrece una solución a un problema particular. Lo que se obtiene es un espacio convolucionado (hipersuperficie de calidad), que puede ser pensado en tres dimensiones y que esta formado por una serie de valles, picos y superficies planas (Fig. 2). Dependiendo de el tipo de problema que se tenga, podremos incrementar una ganancia o minimizar una perdida. Esto equivaldría a subir la montaña más alta o el valle más profundo (siendo este último caso un descenso del gradiente que apunta a la solución con el error mínimo). Sin embargo, la realidad es que deambular por un terreno abrupto como el que nos ofrece la hipersuperficie de calidad es un caminar errático. Por lo tanto debe existir una manera simple de encontrar el valor máximo. Una heurística o solución práctica nos sugiere caminar siempre hacia arriba en el caso de optimizar la calidad. ¿Pero que sucede sí nos encontramos en la montaña casi más alta? La respuesta es que no vamos a bajar de esta montaña por la simple razón de que la instrucción consiste en subir siempre sin retroceder.

Esta misma situación la encuentra el algoritmo genético, que para empezar no sabe en donde esta iniciandose la búsqueda dentro de la hipersuperficie de

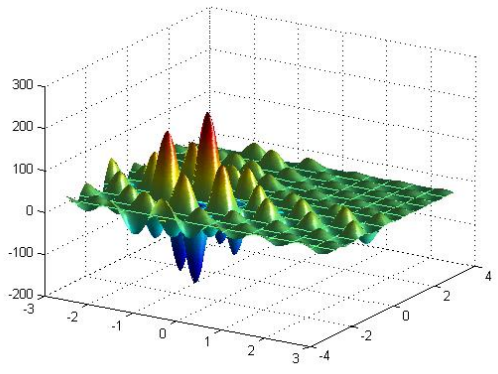


Figura 2. Una hipersuperficie de calidad generada usando Matlab © [11].

calidad. Un punto a mencionar, es que la búsqueda en una hipersuperficie de calidad equivale a buscar el valle más profundo o la montaña más alta en medio de una densa neblina. Es por esto que el GA no sabe donde se encuentra y lo único que puede distinguir es su avance paso a paso hacia una mejor solución del problema a resolver. De este modo el algoritmo genético realiza una búsqueda paralela partiendo de varios puntos elegidos inicialmente al azar. La manera de refinar el siguiente paso para identificar una solución mejor en la evolución artificial esta dada por la función de calidad. Esta función permite que los puntos iniciales seleccionados al azar comienzen a moverse en la hipersuperficie de calidad. Adicionalmente, esta función identificia características especiales que deben preservarse cuando se esta resolviendo un problema particular, por tanto otorga los puntajes más altos a aquellas soluciones más prometedoras. Como resultado si graficamos la calidad a través de las distintas generaciones se nota que algunos individuos muestran un mejor desempeño a cada paso de la evolución artificial, mientras que la población en general mejora al producir individuos cada vez más aptos (Fig. 3).

2.3. Diseño de conductas empleando evolución artificial

Un ejemplo del uso de la función de calidad en el algoritmo genético es el siguiente: Vamos a partir de una tarea simple, navegar un ambiente semiestructurado con algunos obstáculos, donde el robot tiene que recorrer la mayor parte del espacio físico al cual llamaremos arena. El robot no dispone de un esquema de navegación específica, únicamente pensemos que tiene un selector de conductas, cuyos parámetros van a ser optimizados con el uso del algoritmo genético. La evolución artificial en este caso inicia por generar múltiples individuos con valores aleatorios para todos los parámetros arreglados en un vector. Por tanto

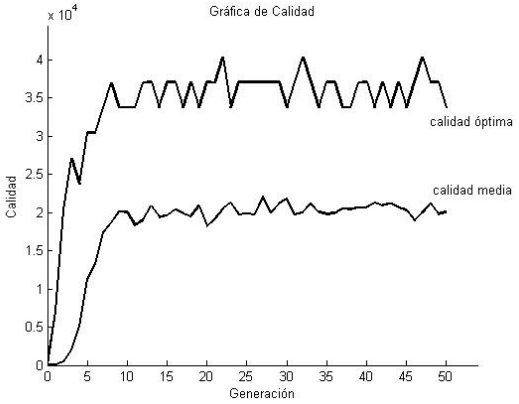
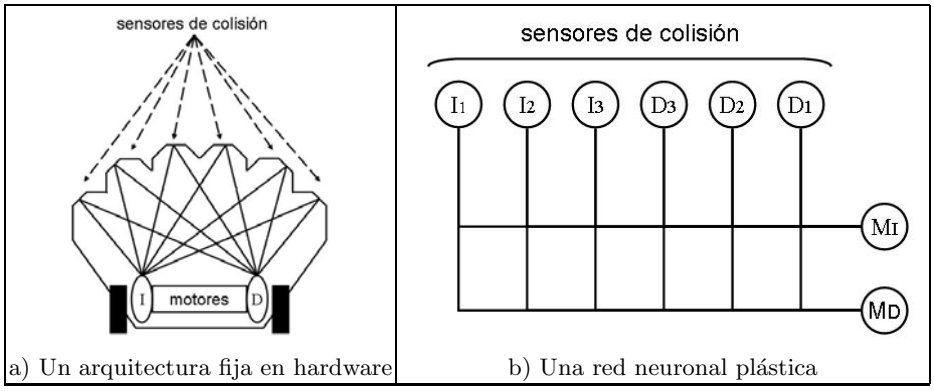


Figura 3. Se presenta una gráfica típica del resultado de aplicar una función de calidad a una poblacón usando un algoritmo genético. La calidad óptima representa aquellos individuos que mejoran notablemente a cada paso de la evolución. Mientras que la calidad media refleja el estado general de la población donde existen individuos cada vez más aptos, pero también individuos no tan aptos son producidos. A medida que la calidad aumenta con el tiempo una solución adecuada puede ser encontrada.

supongomas que tenemos una entrada de seis sensores que detectan colisiones, con los valores de estos seis sensores se hacen dos cálculos. Uno para determinar la velocidad del motor izquierdo del robot y otro para determinar la velocidad del motor derecho del motor (Fig. 4). Para cada motor, cada uno de los sensores tiene una relevancia particular, dependiendo de la situación en la que el robot se encuentre. No es difícil crearnos una imagen mental del caso cuando los tres primeros sensores a la izquierda nos indican un obstáculo, entonces si queremos que el robot gire sobre su propio eje para evitar el obstáculo. Si el obstáculo está a la izquierda el robot puede girar en cualquiera de las dos direcciones, izquierda o derecha. Sin embargo, la forma más rápida de evitar un obstáculo a la izquierda, partiendo de que no existe obstáculo a la derecha está en girar precisamente hacia el lado contrario (la derecha). De esta manera, el robot móvil que consta de dos ruedas dispuestas en el mismo eje horizontal al centro del robot puede evitar objetos que obstruyan su paso. Existe un punto adicional a considerar para que un giro óptimo ocurra el robor debería girar para un óbstaculo a la izquierda su motor izquierdo hacia adelante y el derecho hacia atrás, un giro subóptimo consiste en detener el motor derecho y avanzar únicamente el izquierdo.

Regresando a nuestro ejemplo, tenemos seis valores de entrada para el motor izquierdo y seis valores de entrada para el motor derecho. Para cada uno de estas entradas existe una conexión pesada que mide la relevancia de la información proveniente de los sensores que puede ser binaria, bipolar o oscilar en un rango determinado, por el momento vamos a considerarla binaria. Por lo tanto tenemos seis más seis pesos asociados a las entradas lo cual forma un vector de doce



a) Un arquitectura fija en hardware b) Una red neuronal plástica

Figura 4. Diseño de una red para una arquitectura robot simple

elementos. Este vector de doce elementos ofrece una solución espécifica al problema de evasión de obstáculos. La solución puede variar desde no funcional hasta la óptima. La manera de determinar el nivel de la calidad de una función va a depender de que puntaje se le otorga a un individuo con estas características. Esta medición de la calidad también equivaldría a señalar la siguiente ubicación de la calidad de un individuo en una hipersuperficie convolucionada.

Posteriormente se abordará como funcionan los operadores genéticos por el momento basta mencionar que intercambian, modifican y promueven los valores de cada individuo expresados en el arreglo de doce elementos para este caso. De modo que puedan producirse nuevos individuos. Sin embargo a cada paso de la evolución incremental se debe medir la calidad de cada individuo (Fig. 5). En este ejemplo de evasión de obstáculos la función de calidad debe medir las caractérisiticas que hacen que un individuo evite obstáculos y explore la arena. Entonces se debe premiar a aquellos individuos que muestran indicios de poder viajar hacia adelante, mientras puedan girar hacia los lados evitando obstáculos, incluso aquellos que se presentan de frente. Es esta la labor de la función de calidad que debe llevar un puntaje de cada una de las carácteristicas que permiten a un individuo optimizar el comportamiento que estamos modelando. Sin embargo, se debe recordar que la función de calidad únicamente tiene acceso a los valores de la velocidad de los motores, los cuales pueden ser un valor negativo o positivo al tiempo t para cada uno de los motores de corriente directa. Además de los valores de los sensores de colisión que son binarios como se especificó previamente.

Por lo tanto una función de calidad para la resolución de una tarea de evitar-obstáculos (basada en [12] [13]) para un comportamiento que emplea una red neuronal plástica es la siguiente:

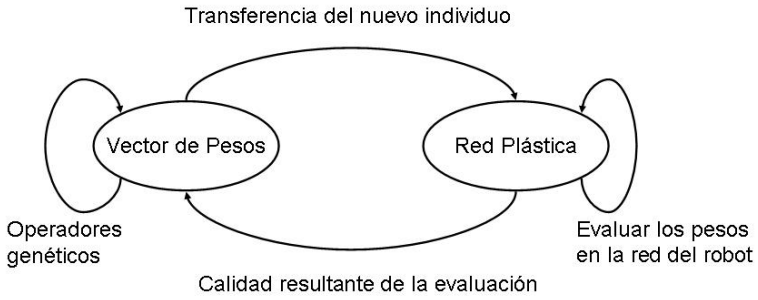


Figura 5. El algoritmo genético prueba en el robot la red neuronal plástica con los mejores pesos encontrados hasta el momento. Como resultado de la prueba, se determina la calidad de este individuo, el material genético de este individuo que corresponde con el vector de pesos de la red plástica es modificado por medio de los operadores genéticos. Este proceso se realiza iterativamente en paralelo para generar nuevos individuos que a su vez serán probados y su material genético intercambiado. El algoritmo encuentra su condición de parada cuando una solución con una calidad aceptable es encontrada.

$$f_{c1} = \sum_{i=0}^{2500} abs(vl_i)(1 - \sqrt{vd_i})(1 - colision\_pr_i) \quad (1)$$

donde para la *i-ésima* iteración de 2500 que es la duración de la evaluación de la calidad se tiene que, *vl* es la velocidad lineal en ambas ruedas (el valor absoluto de la suma de las velocidades izquierda y derecha), *vd* es la velocidad diferencial en ambas ruedas (mide la velocidad angular), finalmente *colision_pr* indica la presencia de colisión en alguno de los sensores.

Es importante señalar que una función como esta premia a aquellos individuos que son los más rápidos y que viajan en línea recta mientras evitan obstáculos y que no persisten en avanzar hacia adelante cuando un obstáculo los bloquea. En la siguiente sección se aborda el uso de los operadores genéticos y la modelación de comportamientos específicos para su uso con una arquitectura de selección de acción particular.

3. Un mecanismo centralizado de integración de información para la selección de acción

Es esta sección abordaremos la importancia de usar un mecanismo de selección de acción para resolver una tarea de forrajeo. Anteriormente se describió la arquitectura de selección de acción de Pattie Maes [8], la cual ofrece un enfoque distribuido para resolver este problema. Sin embargo, en la biología un enfoque

distribuido es poco práctico debido a necesidades metabólicas y de espacio. En otras palabras, tener un mayor número de cables requiere de mayor energía para alimentar un circuito, además de que un incremento en el número de conexiones requiere mayor espacio. Esto no es posible si nos avocamos a la hipótesis de un circuito de selección de acción en el cerebro vertebrado. A continuación se introduce una arquitectura central de selección la cual es biológicamente posible y que produce selección de acción de una manera efectiva.

3.1. CASSF, una arquitectura central de selección

Este modelo denominado una Central Arquitectura de Selección con Sensores Fusionados (CASSF) ofrece una combinación de la información proveniente de distintos sensores herogéneos entre sí [14]. Adicionalmente el uso de CASSF para la modelación bajo el esquema basado en el comportamiento presenta los medios suficientes para una selección de acción efectiva [15]. Por otra parte esta arquitectura permite que bajo una plataforma robótica se integren las percepciones sensores con el mecanismo de selección de acción para producir los comandos motores apropiados para ejecutar las conductas que resuelven la tarea de forrajeo. Este modelo centralizado construye una percepción unificada del mundo a cada paso del bucle principal de control. Adicionalmente, la percepción fusionada del mundo facilita el uso de la información de distintos sensores, los cuales forman variables perceptuales que son interpretadas y sopesadas por el mecanismo de selección de acción.

Estas variables perceptuales son empleadas para calcular la urgencia (saliencia) de un módulo conductual que puede ser ejecutable y que trata de ganar control de la planta motora. Además, los módulos contribuyen con una parte de su propia influencia al cálculo de la saliencia con un señal de estado-ocupado indicando que se esta atravesando por un periodo crítico donde no debería ocurrir una interrupción. Por lo tanto, la saliencia del módulo conductual se calcula por la suma de la información sensora del medio ambiente con la percpeción interna de un estado de ocupado. Una vez que los distintos niveles de urgencia han sido calculados, entonces el módulo conductual o la conducta con el nivel de urgencia más alto gana la competencia, así su ejecución es expresada por medio de comandos motores dirigidos a las ruedas y la pinza del robot (Fig. 6). En caso de empate se elige al azar una de las conductas en competencia. A continuación se especifica como se lleva a cabo el cálculo de la saliencia.

Para la tarea de forrajeo se van a describir cuatro variables perceptuales, las cuales son detecta_pared (e_w), detecta_pinza (e_g), detecta_cilindro (e_c), y detecta_esquina (e_r) que se codifican a partir de configuraciones específicas de los distintos sensores. Por ejemplo, generalmente una pared es indicada por más de un valor alto en tres de los seis sensores infrarojos forntales del robot. Por otra parte un objeto en la pinza puede ser detectado por medio de un sensor de bloqueo en la pinza. Otra opción es que el robot cuente con un sensor de medición de resistividad, que diferencia objetos metálicos de los no metálicos. Un

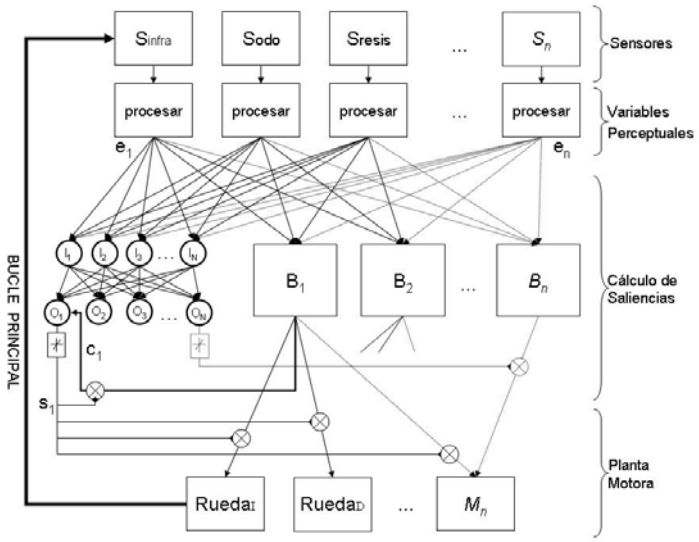


Figura 6. En el modelo CASSF, las variables perceptuales (e_i) forman la entrada para la red de decisión que calcula la saliencia. La saliencia (s_i) con el nivel más alto es seleccionada y expresada a través de los motores del robot. Se debe notar la ocurrencia de la señal de ocupado (c_1) del módulo conductual B_1 a las neuronas de salida.

cilindro pequeño generalmente es detectado por a lo más dos sensores infrarojos. Una esquina es detectada por al menos cuatro sensores de los seis infrarojos. Se debe mencionar que estas son configuraciones típicas, sin embargo existen casos específicos donde el ruido y las distintas orientaciones del robot con respecto a varios objetos presentan configuraciones difíciles de generalizar. Es por esto que el uso de una red neuronal realiza una mejor generalización de los distintos casos mas representativos a los que el robot puede enfrentarse. Este punto se trata mas adelante cuando se explica la evolución de las conductas.

Regresando al cálculo de la urgencia se tiene que las variables perceptuales forman un vector contextual el cual se construye de la siguiente manera: ($\mathbf{e} = [e_w, e_g, e_c, e_r], e_w, e_g, e_c, e_r \in \{1, 0\}$). Posteriormente, cuatro diferentes módulos conductuales regresan un estado-ocupado ($\mathbf{c}_i$) indicando sí alguna actividad seleccionable debería o no ser interrumpida. Entonces, se describe como el vector actualizado de estado-ocupado se forma: $\mathbf{c} = [c_s, c_p, c_w, c_d], c_s, c_p, c_w, c_d \in \{1, 0\}$, respectivamente para los módulos *busca-cilindro*, *recoge-cilindro*, *busca-pared*, y *deposita-cilindro*. Finalmente, la urgencia o saliencia ($\mathbf{s}_i$) se calcula por la adición del estado-ocupado pesado ($c_i \cdot w_b$) a el vector contextual pesado ($e \cdot [w_j^e]^T$).

Se debe notar que los módulos conductuales están cercanamente relacionados con patrones conductuales específicos (conductas). La mayor parte del tiempo la selección de una única conducta ocurre, sin embargo la competencia de módulos con niveles de urgencia similares hace que el modelo de selección de acción tenga que resolver competencias donde únicamente se activa un solo ganador cuya ejecución es la más relevante para la percepción actual del medio ambiente. Sin embargo, existen situaciones donde podría favorecerse la selección de más de una conducta, pero es más difícil controlar una selección múltiple. Como resultado de una selección dinámica de conductas se tiene que patrones complejos emergen y al final la conducta de forrajeo se resuelve adecuadademente. Por lo tanto este mecanismo de selección implementa un mecanismo del tipo *el que gana-toma-todo* que tiene que elegir entre los cuatro módulos conductuales disponibles. A continuación se propone una breve descripción de estos módulos: *busca-cilindro* explora aleatoriamente la arena buscando comida (cilindros) mientras evita obstáculos, *recoge-cilindro* encuentra el espacio adecuado para bajar la pinza y capturar un cilindro, *busca-pared* encuentra la pared más cercana evitando cualquier obstáculo presente; finalmente *deposita-cilindro* baja la pinza y libera un cilindro capturado.

Aún cuando el cálculo de la saliencia se ha descrito como un vector formado por cuatro saliencias de los módulos conductuales. Este mismo cálculo puede ser pensado como una red neural de decisión de dos capas, ambas de cuatro neuronas. La capa de entrada simplemente distribuye los valores perceptuales de los sensores del robot. Por otra parte la capa de salida, a través de cada una de sus neuronas pesa las variables perceptuales y cálcula la saliencia pertinente. Por tanto las neuronas en la capa de salida emplean la identidad como función de transferencia para preservar el valor original de la urgencia. A continuacióm la conducta seleccionada envía una señal de ocupado a las neuronas de salida

cuando su ejecución ha sido promovida. Una conducta seleccionada envía una copia de este estado-ocupado a las cuatro neuronas de salida. En su oportunidad cada uno de los cuatro módulos conductuales es eleccionable, por lo tanto cada módulo envía la señal de ocupado a las neuronas de salida agregando cuatro entradas más. En la siguiente sección se explica el uso de los operadores genéticos en el desarrollo de las conductas para CASSF empleando evolución artificial.

3.2. Desarrollo de conductas para CASSF

Para el desarrollo de conductas en CASSF se emplea como módulo conductual un tipo particular de redes neuronales [17]. Además de que el uso de redes neuronales con algoritmos genéticos es una solución común para la modelación robótica [16]. El uso de este enfoque requiere la eleción adecuada de la topología y los pesos de la red neuronal para controlar el robot. Es la elección de los distintos posibles valores para los pesos de la red neuronal la que produce desde individuos no-aptos hasta muy aptos. Como se mencionó anteriormente sí la calidad de la solución ofrecida por todos estos individuos se grafica tendríamos un espacio convolucionado donde el ascenso del grandiente que apunta hacia la solución óptima tiene que ser seguirse. La mejora en la calidad de las soluciones esta dada por los operadores genéticos, sin embargo se debe mencionar que no únicamente se produce un promedio de individuos de buena calidad, sino también algunos mejores individuos junto con bastantes peores también. Es esta variacón de la población la que permite que después de varias iteraciones un individuo que sea muy apto se genere.

Los operadores genéticos o evolutivos se aplican durante todo el proceso de evolución. La población inicial se forma de valores aleatorios para los pesos de las redes neuronales de los primeros individuos. Esto equivale a colocar varios puntos iniciales en el espacio convolucionado donde un valor óptimo o cuasi-óptimo puede ser encontrado (Fig. 7). La calidad de las soluciones de los individuos aleatorios iniciales es calculada, y a partir de los mejores individuos se produce una nueva generación a través de los operadores genéticos, este proceso se repite hasta que una solución adecuada es encontrada.

La *selección* busca engendrar aquellos individuos más aptos a partir del material genético de la población activa. Una vez que los individuos son seleccionados se aplican los operadores de cruzamiento y mutación. Una instancia particular de la selección, es la *selección por torneo*, en ésta se eligen los mejores individuos a partir de competencias de tamaño variable donde se eligen varios competidores al azar y los de mejor calidad son seleccionados para engendrar un nuevo individuo. Sin embargo, existen casos donde un individuo puede generar el material de un individuo completo, al ser todo su material genético copiado en la siguiente generación. Esta práctica de insertar individuos intactos en la siguiente generación se le conoce como *elitismo*. Por otra parte, el *cruzamiento* es un operador que toma el material genético de dos de los mejores individuos e intercambia su material a partir de uno o varios puntos aleatorios del vector

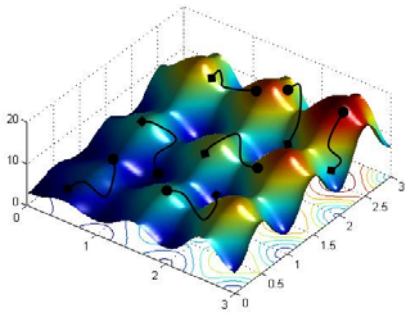


Figura 7. Se muestra una hipersuperficie de calidad con varios puntos iniciales dispuestos al azar que corresponden a los individuos de la población inicial cuyos pesos son manipulados por el algoritmo genético. Un rombo indica el punto inicial donde el individuo es dispuesto, un círculo representa su movimiento final.

de pesos de un individuo. Por último la mutación puede afectar un porcentaje del material genético de una población al alterar los valores de un elemento del arreglo genético de un solo individuo. Un ejemplo consiste en producir un valor aleatorio solamente para uno de los genes o elementos del arreglo genético, es decir el cambio de un valor dado en el contenido de un individuo por un valor al azar.

Es importante señalar que el material genético en ocasiones requiere una cierta codificaicón y decodificación para poder ser manipulado. Sin embargo es frecuente usar una codificación directa donde el arreglo genético es manipulado en su estado en bruto [18]. Para el desarrollo de dos conductas que describiremos a continuación se hizo uso de un simulador robótico y un robot Khepera. Como se explicará más adelante existen ventajas de este enfoque. Anteriormente se explicó el desarrollo de una función de calidad, por tanto para el desarrollo de las conductas *busca-pared* y *busca-cilindro* se requiere la elección de una función que produzca el comportamiento que es requerido. Otro punto que es importante señalar es que los comportamiento secuenciales que están siempre formados por el mismo conjunto de iteraciones no son los candidatos más adecuados para la evolución. Este es el caso de las conductas *levanta-cilindro* y *suelta-cilindro* que también se explican en este apartado.

Para la topología del soporte neuronal de las conductas a evolucionar se seleccionó una red del tipo perceptrón completamente conectada con alimentación delantera, multicapa, sin conexiones recurrentes. El perceptrón [19] es un modelo neuronal básico que se explica en un curso introductorio de redes neuronales. Normalmente este modelo consiste de tres capas, una de entrada que distribuye las entradas dadas, una intermedia que hace un preprocesmaiento de las entradas y una capa de salida que define con un valor superior al de las otras neuronas la clasificación de un patrón de salida. Puede existir un preprocesaiento en las

entradas y un procesamiento de las salidas para ajustarlas al problema que se esta tratando de resolver. Debe mencionarse que una red neuronal de este tipo es un gran clasificador que con ciertas limitaciones puede identificar grupos específicos a los que pertenece una entrada (por ejemplo una clasifiación de tipos de líneas inclinadas). Para usos específicos este modelo neuronal puede no ser el más adecuado, sin embargo es un buen comienzo para resolver un problema de clasificación. Una solución típica para entrenar los pesos de esta red consiste en un usar un algoritmo de retropropagación de error, que consiste en tener un grupo de entradas identificadas cuya salida es conocida de antemano (aprendizaje supervisado). Este algoritmo tratara de minimizar el error en la clasificación de las entradas con salidas conocidas a través de varias iteraciones. Dos puntos importantes acerca del uso de este tipo de redes neuronales, es que la única manera de saber si un problema es clasificables es sí en realidad lo es y el otro punto es que la elección de la toplogía de la red neuronal muchas veces esta dada a partir de la experiencia con el problema a resolver.

Regresando al diseño de nuestras conductas, nuestro perceptrón tiene seis neuronas de entradas conectadas a cinco neuronas en la capa intermedia, las cuales a su vez están conectadas a las dos neuronas de la capa de salida (controlan los motores izquierdo y derecho). Los seis valores que alimentan la capa de entrada corresponden con los seis sensores frontales del robot Khepera. Para facilitar la variación de valores entre el simulador y el robot real, los cuales están sujetos a mucho ruido. Entonces se decidio usar un umbral de colisión $th_c = 800$ que facilita la binarización de los sensores reales y simulados. La salida continua bipolar de la red neuronal es escalda a los ± 20 valores que controlan los motores de corriente directa del robot.

El algoritmo genético almacena los valores de un controlador en un cromosoma que codifica directamente la conducta busca-pared que es una forma de evasión de obstáculos como se muestra adelante. Para esta conducta el cromosoma esta formado por $(6 * 5) + (5 * 2) = 40$ pesos, en el vector $\mathbf{ch_x}$. Para este cromosoma inicialmente se se generan valores aleatorios, $\mathbf{ch}_{xi}$, $-K_w < \mathbf{ch}_{xi} < K_w$ con $K_w = 7{,}5$. Se generan $n = 100$ cromosomas individuales para formar la población G_0. La calidad de cada uno de los individuos es evaluada por alrededor 25 segundos en el simulador y entonces los operadores evolutivos son aplicados. Los dos mejores individuos son copiados en la siguiente generación a manera de *elitismo*. La *selección por torneo* es empleada para cada una de las $(n/2) - 1$ competencias locales que producen dos padres para engendrar un nuevo individuo empleando un solo punto aleatorio de *cruzamiento* con una probabilidad de 0.5. A continuación la población entera se ve afectada con una *mutación* con una probabilidad de 0.01. Cada uno de los individuos de la nueva descendencia son nuevamente evaluados durante 25 segundos.

La conducta *busca-pared* es una instancia de la conducta de evasión de obstáculos debido a que una pared debe ser encontrada sin colisionar con ningún obstáculo. Cuando una pared es encontrada el mecanismo de selección de acción detiene la ejecución de la conducta. La ecuación emplea contiene una formula

similar a 1 y esta basada en [20]. Por otra parte, la conducta *busca-cilindro* es una instancia de la conducta de evasión de obstáculos combinada con un detector de cilindros. Para esta segunda conducta, el robot tiene que ser posicionado exactamente enfrente de un cilindro para que el robot pueda moverse hacia atrás y colectar adecuadamente el cilindro. La conducta busca-cilindro se obtiene como el resultado de usar evolución en los pesos de la red neural. Estos pesos se condifican en en el vector $\mathbf{ch}_x$ de 40 elementos. Para esta conducta la función de calidad es la siguiente:

$$f_{c2} = f_{c1} + K_1 \cdot ccerca + K_2 \cdot cenfrente \tag{2}$$

donde en una formula como esta se seleccionan los individuos que evitan obstáculos y localizan cilindros posicionados alrededor de la parte frontal del cuerpo del robot (*ccerca*), capaz de orientar el robot en una posición donde el cilindro pueda moverse para recoger el cilindro (*cenfrente*). Las constantes K_1 y K_2, $K_1 < K_2$, se usan para recompensar el posicionamiento adecuado del robot frente a un cilindro.

La manipulación de la pinza en el robot requiere un número fijo de iteraciones que hacen que la definicón de una función de calidad requiera una evolución por etapas. Además existe un problema con la detección de un cilindro cuando este va a ser recolectado. La principal razón de la confusión de la recolección de un cilindro en el robot Khepera es la siguiente. Cuando un cilindro es detectado y posicionado exactamente enfrente de cuerpo del robot, los dos sensores más frontales proporcionan una lectura muy alta. Cuando el robot se aleja para poder bajar la garra robótica, estas lecturas caen a niveles muy bajos, es como sí el cilindro desapareciera. Posteriormente al mover el brazo robótico hacia abajo, hace que el travesaño que une cada una de las garras de la pinza sea confundida con una barrera como sí hubiera una pared enfrente. En el último paso, el cilindro no es detectado correctamente hasta que el brazo esta en lo alto sujetando el cilindro. Todas estas inconveniencias hacen que el emplear las lecturas de los sensores en la pinza para determinar la utilidad de esta acción a través de una función de calidad sea una tarea muy difícil de llevar a cabo con evolución.

Para la conducta *recoge-cilindro* se muestra el pseudocódigo que lleva a cabo esta tarea:

```
paso = 0
repite
 si paso == 0; entonces mueve_robot_atras, paso = 1
 si paso == 1; entonces abrir_la_pinza, paso = 2
 si paso == 2; entonces bajar_el_brazo, paso = 3
 si paso == 3; entonces cerrar_pinza, paso = 4
 si paso == 4; entonces levanta_brazo, paso = -1
hasta que paso == -1
```

La conducta *deposita cilindro* esta definida como se muestra a continuacón.

```
paso = 0
repite
 si paso == 0; entonces bajar_el_brazo, paso = 1
 si paso == 1; entonces abrir_pinza, paso = 2
 si paso == 2; entonces levanta_brazo, paso = -1
hasta que paso == -1
```

Se ha mostrado la definición de módulos conductuales heterogeneos, en los cuales la interacción entre el diseñador humano es evidente. En la siguiente sección se muestra como puede reducirse el numero de las decisiones tomadas por el humano.

3.3. Reduciendo las decisiones de un diseñador humano

El número de decisiones tomadas por el diseñador humano pueden reducirse al usar extensivamente el algoritmo genético [21]. En otras palabras es posible evolucionar al mismo tiempo varias caracteristicas por medio de coevolución [22]. Sin embargo es necesario resaltar que al optimizar el algoritmo genético optimiza en espacio y tiempo. Lo que significa que es posible notar que el orden de las conductas como lo ha percibido el diseñador humano puede cambiar, o incluso algunos módulos conductuales especializados pueden ser descartados. Aún más algunos módulos pueden ser generalizados de modo que estos puedan emplearse para distintas acciones. La coevolución se va explicar en el siguiente párrafo pero para aclarar el punto anterior se proporciona el siguiente ejemplo. Supongamos que queremos evolucionar al mismo tiempo las conductas relacionadas con el forrajeo, así como los parámetros de selección del mecanismo de selección de acción. En este caso si no se especifica adecuadamente la función de calidad y las condiciones en que se pueden o no activar los módulos conductuales. Entonces para cuatro conductas que son *busca_cilindro, recoge_cilindro, busca_pared, deposita_cilindro* puede ocurrir que deposita_cilindro sea descartada debido a que esta conducta es difícil de terminar completamente porque por diseño del robot las lecturas de los sensores en la pinza para detectar la presencia del cilindro son poco confiables cuando el brazo esta en movimiento.

Los anterior significa que ante la poca confiabilidad de los sensores en movimiento, entonces un cilindro podría parecer ausente cuando el brazo se esta moviendo constantemente. Por otra parte recoge_cilindro es activada continuamente porque como una medida de precaución, la pinza siempre es abierta y el brazo sacudido para asegurar que por la misma razón de que no es confiable detectar la presencia del cilindro en la pinza, entonces es necesario asegurarse que no se esta sosteniendo un cilindro antes de recoger otro. En este caso el algoritmo genético con la coevolución generaliza el uso de la conducta *recoge_cilindro* para hacer ambas cosas recoger y soltar el cilindro. Debido a que el patrón de forrajeo consta de varias etapas, la selección de *libera_cilindro* ocurre a una etapa muy posterior cuando ya han sido calibrados los parámetros de selección que activan la ejecución de las conductas. Es por esto que modelar comportamientos

secuenciales, aún en este caso que dependen de la selección de distintos comportamientos complica un poco las cosas con la optimización por evolución. Se pueden poner restricciones en la activación de los módulos pero entonces limitamos al algoritmo genético de descubrir nuevas soluciones no previstas por el diseñador humano. Por lo tanto la coevolución con el uso de robótica reactiva requiere una solución comprometida donde el diseñador especifique la función de calidad y los comportamientos con la suficiente flexibilidad para que el algoritmo genético encuentre mejores soluciones y se trate de cubrir el comportamiento observado por el humano. A continuación se explica el uso de coevolución en el diseo de dos comportamiento, pero de igual modo puede aplicare la coevolución al modelado de selección y comportamientos o también de la selección de dos entidades cooperativas o en competencia.

Para la coevolución de los comportamientos *busca_cilindro, busca_pared* se usaron los dos cromosomas descritos anteriormente unidos en un solo cromosoma usando codificación directa. Por lo tanto se tienen 80 pesos para el vector $\mathbf{ch_x}$. La coevolución se realiza de la misma manera que la evolución artificial, en este caso con el uso de un algoritmo genético. De este modo los operadores genéticos se aplican como se explicó anteriormente. La formula para la coevolución de los pesos (basada en [23]) para las conductas busca_cilindro y busca_pared es la siguiente

$$f_{cv} = (K_1 \cdot ep) + (K_2 \cdot bcl) + (K_3 \cdot rcfactor) + (K_4 \cdot lbfactor) \qquad (3)$$

La coevolución de los pesos de los patrones conductuales se llevo a cabo usando en la formula (f_{cv}) las constantes K_1, K_2 , K_3 and K_4 con $K_1 < K_2 < K_3 < K_4$ para la selección de aquellos individuos que encuentran paredes en la arena (ep). Sin embargo, la función de calidad premia mayormente a aquellos individuos que localizan cilindros (bcl), su recolección dentro del arena ($rcfactor$) y su liberación cerca de las paredes ($blfactor$).

En la siguiente sección se presenta de manera concisa algunos ejemplos de resolución de tareas usando selección de acción. Por lo tanto se especifica la plataforma a emplear, la cual es un robot Khepera. Posteriormente se señala la importancia de usar un robot real contra un simulador o el uso de ambos en la evolución artificial. Posteriormente se muestra dos ejemplos, uno de una tarea de forrajeo y otro de una tarea social competitiva, ambos haciendo uso de la selección de acción.

4. Resolución de tareas por medio de selección de acción

El uso de evolución artificial con la robótica basada en el comportamiento no son las únicas soluciones para resolver el problema de selección de acción. En esta sección se presentan dos ejemplos que combinan la selección de acción con la robótica reactiva. Sin embargo antes es necesario dar una breve explicación del la plataforma robótica que hemos seleccionado para nuestros experimentos.

El uso de una plataforma comercial como esta requiere del uso de un simuladores robótico para el desarrollo de prototipos rápidos e incluso de la evolución artificial. Existen ventajas sobre el uso de un robot real contra el uso de un robot simulado. A continuación se explica como el uso de ambos nos proporcionan las herramientas necesarias para la modelación de animales robot.

4.1. El robot Khepera

Hasta el momento existen tres versiones del robot Khepera [24], siendo las dos primeras muy parecidas y las que presentan mayores similaridades. Por lo tanto a continuación cuando se menciona el robot Khepera en el texto se hae referencia a las primeras dos versiones a menos que se especifique lo contrario, entonces se hace referencia a la tercera versión. El robot Khepera ha sido nombrado como una de las formas del Dios del Sol Egipcio *Re*, la cual en algunas ocasiones es identificada como un escarabajo. Por lo tanto el Khepera móvil es un escarabajo que en lugar de patas para su locomoción emplea un par de ruedas dispuestas a ambos lados del centro del robot. Dos balines de teflón permite que en caso de que el robot se balance hacia adelante o hacia atras, este se mantenga regularmente en una posición que le permite desplazarse libremente. El rozamiento es despreciable en un robot tan pequeño como este que cuenta con siete centímetros de diámetro en su segunda versión (Fig. 8). Dos motores de corriente directa alimentan a las ruedas izquierdas y derecha para permitir una velocidad máxima de cincuenta centímetros sobre segundo a su máxima potencia con una autonomía con carga completa de alrededor de una hora.

Figura 8. Un robot Khepera uno, equipado con una pinza y una cámara inalámbrica. Un robot como este puede emplearse en tareas de forrajeo. El uso de la pinza es opcional para experimentos sociales.

Este robot principalmente detecta obstáculos por medio de un anillo de sensores infrarojos distribuidos alrededor del robot, con seis sensores en su parte

frontal y dos en su parte trasera. El robot puede ser controlado de dos maneras principales, una a través de una especie de control remoto usando una interfaz RS232 serial y cualquier programa que sea capaz de conectarse de este modo al robot. Incluso usando algunas de las bibliotecas que se proporcionan en la página del fabricante del robot para algunos programas como Matlab © y LabView ©. A pesar de que la comunicación serial es la opción preferida la mayor parte del tiempo, existe una forma de descargar un programa en versión nativa del controlador del Khepera para proporcionar con la batería previamente cargada una verdadera autonomía. Básicamente el Khepera esta formado por un emparedado de circuitos electrónicos donde se contienen los motores, engranes, las baterías y circuitos para el control del Khepera. Esta es la versión base, la cual esta más desnuda en su primera versión y en la segunda una carcaza de un material plástico resguarda con mayor efectividad las entrañas del escarabajo robot.

Existen diversas torretas de extensón para aumentar las capacidades del Khepera. Las torretas varían desde una pinza, distintos tipos de cámaras CCD incluyendo una versión inalámbrica y una con una visión de 360 grados; otras torretas permiten una comunicación por radio y existe una torreta que permite el desarrollo de una torreta hecha a la medida. Para nuestros experimentos el uso de la pinza y de la cámara de color son indispensables. En esta sección se explica brevemente la pinza y posteriormente se explica el uso de la cámara. La pinza tiene dos grados de libertad con dos motores que facilitan su apertura y su movimiento hacia el frente y hacia atrás. Este pequeño brazo robot se monta encima de la versión básica del robot siendo entonces que los dos sensores de la pinza pueden funcionar, uno es una barrera óptica que detecta cuando un haz de luz invisible ha sido obstruido, el otro detecta la resistividad de los objectos que la pinza sostiene.

La tercera versión del robot Khepera es de mayor tamaño, ocupando un diámetro de trece centímetros con una velocidad similar a sus versiones previas pero con una autonomía de hasta ocho horas. Esta versión cuenta con un anillo de ocho infrarojos para la detección de obstáculos, dos sensores infrarojos que apuntan al piso para el seguimiento de líneas y cinco sonares frontales para una detección de objetos a mayor distancia. La comunicación con el robot es serial y por medio de red inalámbrica WiFi [2]. Existe una pinza que puede agregarse al robot Khepera tres, la cual tiene un mayor número de grados de libertad. Para mayor información acerca de los distintos tipos del robot Khepera referirse al sitio del fabricante *K-Team* [3]. A continuación enunciaremos algunas ventajas y desventajas en la preferencia de un robot simulado sobre un robot real para nuestros experimentos.

[2] WiFi es un conjunto de estndares para redes inalámbricas basados en las especificaciones IEEE 802.11, fue creado para ser utilizado en redes locales inalámbricas

[3] http://www.k-team.com con acceso válido hasta el momento de la publicación de este libro

4.2. Importancia del uso de un simulador robótico y un robot real

Existen diversas ventajas para preferir el uso del robot real sobre un simulador. A continuación se enuncian brevemente algunas de ellas. El mayor argumento a favor del uso de un robot real estriba en el hecho de que los simuladores recrean hasta cierto punto el comportamiento del robot real. Sin embargo existen muchos pequeños detalles a los que hay que enfrentarse cuando se trabaja con un robot físico. Por ejemplo cambios en las luces afectan la detección de los infrarrojos, por otra parte el comportamiento de estos varía de un robot a uno simulado. Este es el caso de los objetos que no son detectables hasta que se esta casi enfrente de ellos, lo cual no ocurre en el robot simulado. Por otra parte se tienen desventajas como que en los simuladores, algunos objetos son atravesados como si se tratara de fantasmas, además si un objeto es derribado por accidente no se comporta como un objeto real. Son estos casos que presentan una serie de limitantes y oportunidades para el desarollo de mecanismos de control en tiempo real, donde las respuestas a la percpción del mundo se basan en la confiabiidad de lo que el robot percibe. Pero por otra parte, en un robot real la batería se descarga, y si se conduce demasiado rapido hacia una pared puede estrellarse y dañarse. Además de que en el caso de que se tenga un solo robot este debe compartirse y el uso del simulador facilita el desarrollo de pruebas hasta que se tiene un algoritmo confiable que en principio puede funcionar en el robot con relativamente insignificantes cambios. Varios laboratorios de robótica alrededor del mundo, prefieren el uso del simulador antes de probar extensivamente un algoritmo en el robot real.

Por otra parte, cuando se trabaja con algoritmos genéticos existen muchas críticas que apuntan a que si una conducta es complemtamente desarrollada en simulación no explotara las oportunidades encontradads en el mundo real. Sin embargo, la simulación permite un incremento en la velocidad de la convergencia de una solución que es buscada en un espacio complejo por medio del algoritmo genético. Una alternativa adecuada para la evolución artificial es utilizar un enfoque híbrido [25] [26] donde la simulación se usa para el desarrollo de una conducta y el robot real se emplea como última instancia en el ajuste de la conducta final. Usando este enfoque debe encontrarse una solución comprometida donde el algoritmo genético pueda explotar algunas oportunidades en el robot real. Una mayor desventaja del uso de simulador es que algunos que de los más robustos son ofecidos como software comercial, por lo tanto es necesario invertir en la compra de una plataforma robótica y un software comercial. Sin embargo para el robot Khepera existen diversas opciones ofrecidas como freeware [4] para el control y el desarrollo del robot Khepera. También existen algunos simuladores muchos de los cuales son en dos dimensiones y se ofrecen de manera gratuita. Por otra parte algunos laboratorios que trabajan con robots, se han empeñado en desarrollar sus propias herramientas y simuladores. Este es el caso del Laboratorio de Análisis y de Arquitecturas de Sistemas (LAAS) en Toulouse

[4] Programas informáticos que se distribuyen a través de la red de forma gratuita.

Francia [5]. Este grupo tiene una plataforma bajo la licencia GNU [6], que es un software llamado GDHE para controlar y simular robots. A continuación se presentan dos ejemplos del desarrollo de experimentos empleando las plataformas roboticas Khepera uno y dos.

4.3. Ejemplos de experimentos con el robot Khepera

En esta sección se abordan dos ejemplos de resolución de tareas. Una de forrajeo y otra del tipo de persecución entre un depredador y una presa. Al ser estos dos tipos de problema comúnmente estudiados en la etología se facilita la comparación de los datos obtenidos bajo observación de animales vivos con los de los robots. La modelación de experimentos de este tipo se lleva a cabo con la idea de entender mejor el comportamiento animal y en una última instancia la inteligencia animal.

Resolución de una tarea de forrajeo Se presenta un ejemplo de una tarea de forrajeo resuelta usando CASSF con ajuste de los parámetros de selección a mano y con el uso de una cámara de color para la detección de la comida [14]. Para un ejemplo de optimización evolutiva usando la cámara revisar el siguiente trabajo [27]. Por otra parte para la modelación de la tarea de forrajeo como una sola conducta empleando el robot básico consultar el trabajo de Nolfi [28]. La tarea de forrajeo esta dividida en *buscar-comida, levantar-comida, depositar-comida, buscar-pared, girar-para-lozalizar* donde el robot es colocado en una arena con paredes bajas y cilindros de madera que simulan comida. La comida es de dos tipos azul para comida en buen estado y roja para comida en estado de descomposición. Existe una especie de batería solar, por medio de la cual durante la ejecución de la tarea de forrajeo el robot es capaz de recargarse siempre y cuando se acerque la distancia y el tiempo suficientes (Fig. 9). Para la resolución de esta tarea se integra la información sensora de los infrarojos en el cuerpo del robot, la información de la camara en un espacio RGB-normalizado para evitar cambios iluminación en la imagen bajo sus tres componentes rojo, verde y azul. Adicionalmente a esta información se agrega la proveniente de la pinza del robot, es decir la presencia o la ausencia de un cilindro.

Para estos experimentos se emplea una cámara de color que se monta encima del kit básico del robot. Una vez montada la cámara a color (con una resolución $= 510 * 562$ pixeles) apunta hacia el frente del robot, en la misma dirección donde el robot tiene que mover la pinza. Como se describió anteriormente, cuando el robot Khepera tiene que sujetar un objeto, este tiene que moverse hacia atrás y entonces los sensores infrarojos dejan de detectar el objeto. Una situación similar ocurre con la cámara, el robot es capaz de detectar un cilindro mientras se

[5] http://www.laas.fr/laas/ con acceso válido hasta el momento de la publicación del libro

[6] Proyecto creado en 1984 con el fin de desarrollar un sistema operativo tipo Unix según la filosofía del "software libre".

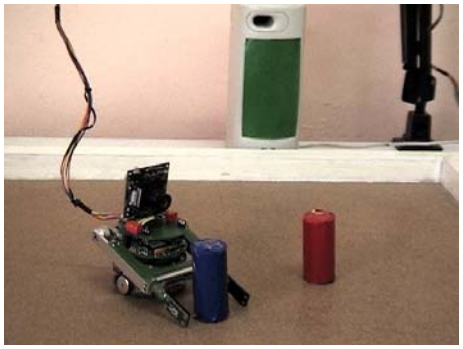


Figura 9. Se muestra un robot Khepera dispuesto en una arena cuadrada donde la comida en buen estado (cilindros azules) y la venenosa (cilindros rojos) estan disponibles. Adicionalmente existe una batería donde el robot puede recargarse (en verde).

está acercando, pero una vez que esta cerca de recogerlo cuando lo esta sosteniendo en la pinza, el robot es incapaz de percibir el cilindro con la cámara. Es por esto que en este caso CASSF integra toda esta información sensora y la convierte en variables perceptuales que son alimentadas al modelo de selección. Con esta informaión el mecanismo de selección es capaz de mantener y terminar la ejecución de las cinco conductas que fueron disieñadas para resolver el problema de selección de acción.

Se debe señalar que aún cuando la cámara se usa para la percepción de la comida, esta no esta activa todo el tiempo solamente se usa para orientarse en línea recta con la comida y la batería. En este sentido la navegación sigue siendo reactiva y se introduce el uso ocasional de la cámara. Se debe recordar que se esta tratando de observar el comportamiento del robot al resolver una tarea como esta usando un esquema reactivo y basado en el comportamiento. De este modo CASSF resuelve la competencia de los módulos conductuales como se explicó anteriormente. Solamente que en este caso todos los módulos y los parámetros de selección fueron ajustados por el diseñador humano. El comportamiento regular para la tarea de forrajeo esta formado por cuatro rondas de la selección de los siguientes comportamientos: buscar-comida, levantar-comida, buscar-pared, depositar-comida con la activación periódica de girar-para-localizar (Fig. 10).

Cada uno de los comportamientos se lleva a cabo de la siguiente manera, *buscar-comida* los sensores frontales del Khepera fueron divididos en las secciones izquierda y derecha, y la diferencia entre ellos se calcula; cuando la diferencia es muy cercana o igual a cero, el robot se detiene porque a localizado un objeto que puede ser una pared o comida; la cámara permite evitar la comida venenosa y acercarse a la batería. Por otra parte *levantar-comida* requiere un movimiento hacia atrás del robot, el brazo se mueve hacia abajo, se cierra la pinza y el brazo se devuelve a su posición inicial. *Depositar-comida* es similar a la acción de recoger la comida, pero en orden inverso. La acción de *buscar-pared* consiste en viajar a

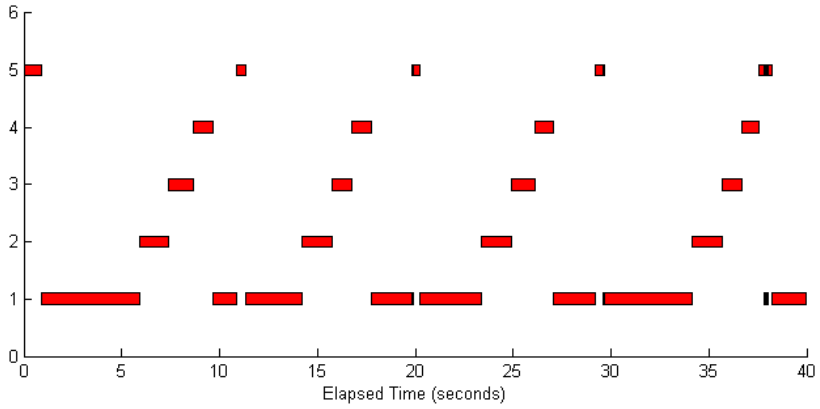


Figura 10. En el etograma se muestra el orden de la activación de los comportamientos. Cada comportamiento es numerado de la siguiente manera: 1. buscar-comida, 2. levantar-comida, 3. buscar-pared, 4. depositar-comida, 5. girar-para-localizar y 6. ninguno. Se debe notar las cuatro rondas que forman un patrón regular. Sin embargo se notan las interrupciones en buscar-comida junto con la activación de girar-para-localizar con el objetivo de encontrar la batería y la comida. Esto debido a que el esquema de navegación se mantiene reactivo aún con el uso de la cámara.

través de la arena hasta que se encuentra una pared evitando cualquier obstáculo. Finalmente, *girar-para-localizar* realiza un giro que puede ser completo o a punto de acompletarse cuando la comida o la batería son detectadas. En esta parte se mostró la resoución de una tarea de forrajeo usando un esquema de navegación reactiva con resolución de tareas basadas en el comportamiento. Las capacidades básicas del robot Khepera son extendidas con el uso de la cámara de color. En el siguiente ejemplo se presenta una combinación de distintos tipos de información visual.

Resolución de tareas sociales competitivas En este experimento se emplearon dos tipos de percepción visual, una periférica y otra frontal junto con robotica reactiva y selección de acción [29]. La percepción frontal es similar a la del experimento anterior solamente que para esta ocasión se colocó encima de un robot Khepera básico un sombrero de color verde para simulador una presa (Fig. 11). El depredador equipado con la camara de CCD es capaz de percibir la presencia de la presa cuando aparece dentro de su campo visual un sombrero verde, por lo tanto convirtiendose en el depredador. La presa no está del todo desprovista de visión otorgandosele una visión periferica limitada de trescientos sesenta grados. Un esquema del tipo el que *gana-toma-todo* se implementa para utilizar la información proveniente de los sensores y conmutar entre las conductas *explorar, perseguir, localizar*. Por otra parte un esquema completamente reactivo permite que la presa explore la arena y cuando la presencia del depredador es sensada se toma una acción evasiva.

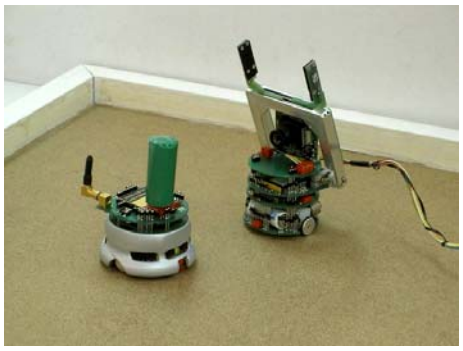


Figura 11. Se muestran un robot Khepera dos que tiene un sombrero verde y que es la presa. El depredador es el robot Khepera uno equipado con la cámara y el brazo (no empleado en este experimento).

Para el depredador la conducta explorar se define como una acción que consiste en atravesar constantemente la arena buscando una presa. Por otra parte la conducta perseguir consiste en dirigirse frontalmente hacia una presa una vez

que esta ha sido detectada, cabe mencionar que la presa puede estar de espaldas o frente a una esquina, por tanto al escapar puede quedar atrapada (Fig. 12). La acción localizar consiste en dar un giro de trescientos sesenta grados sobre su propio eje tratando de encontrar una presa. Por otra parte para la presa se empleó un robot Khepera básico con una cámara CCD colocada encima de la arena. A pesar de que existe una cámara similar a la de visión frontal que esta dispuesta en un ángulo de ciento ochenta grados con respecto al cuerpo del robot y que contiene un espejo convexo para proporcionar una visión cercana a los trescientos sesenta grados con un pequeña área ciega de alrededor 35 milimetros cerca del robot Khepera. Esta cámara es mucho mas costosa que una cámara normal como la que hemos usado, ademas de que requiere algoritmos especializados para reconstruir la imagen a partir de lo que se refleja en el lente convexo.

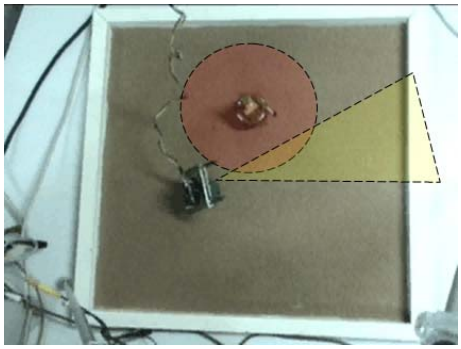


Figura 12. La vista áerea muestra las percepción visual de los dos robots. La presa tiene una visión periférica de trescientos sesenta grados, esta visión es muy limitada para darle oprtunidad al depredador, su campo de visión se representa por un círculo. El depredador muestra un cono que es su campo de visión frontal. En esta imagen la presa podría tomar una acción evasiva ya que la presencia del depredador esta a punto de ser detectada dentro de su campo visual periférico.

En estos dos ejemplos se ha presentado como esquemas reactivos o puramente basados en el comportamiento pueden emplearse para resolver tareas de forrajeo y sociales. Este tipo de tareas son muy soccoridas en la modelación de animales robot debido a que existe una cantidad importante de información realizada bajos estos esquemas con animales vivos. Sin embargo se debe resaltar que estas tareas pueden ser desarrolladas usando evolución artificial como se mostró previamente. A continuación resumimos este capítulo y se proponen algunos ejercicios enfocados a revisar el material propuesto aquí.

5. Conclusiones

En este capítulo se revisó la importancia de la investigación en robótica reactiva para el desarrollo de *animats* que resuelvan tareas mostrando algún tipo de inteligencia y con cierta autonomía. Posteriormente se enfatizó la importancia de la robótica basada en el comportamiento como una etapa intermedia en la resolución de tareas empleando los elementos que han sido asimilados en el repertorio de acciones que un animal robot puede ejecutar. Sin embargo la ejecución del repertorio de conductas del animat requiere que la mayor parte del tiempo se ejecute una acción a la vez, a menos que existan acciones que no presenten incompatiblidades en el uso de los motores del robot. Este problema en el que varios subsistemas tratan de tomar el control de un recurso escaso y compartido como son los motores del robot es un problema de selección de acción. El problema de selección de acción ha sido identificado previamente en la etología, lo cual no se contrapone con el enfoque moderno de la Inteligencia Artificial que permiten un estudio multidisciplinario al emplear conocimiento de otros campos en el desarrollo de mejores aplicaciones. Lo mismo ocurre en la robótica que se ve realimentado de áreas como las neurociencias, la psicología, etc. Sin embargo la resolución de la selección de acción robótica utilizando un enfoque puramente basado en el comportamiento requiere la intervención de un diseñador humano para modelar los comportamientos observados y sintonizar el mecanismo de selección modelado. Por medio del uso de evolución artificial se pretende minimizar el número de decisiones tomadas por el diseñador humano. Por lo tanto como resultado de este trabajo se desprende la idea de continuar el desarrollo de esquemas de selección de acción junto con la robótica evolutiva para la modelación de experimentos conductuales de forrajeo y sociales.

6. Ejercicios propuestos

1. Identifique un problema de selección de acción en la vida diaria y especifique si un mecanismo de selección de acción es útil para su resolución.
2. Ejemplifique tres situaciones donde la ocurrencia de acciones compatibles puede presentarse al mismo tiempo. Las situaciones pueden refererise a sistemas vivos o no vivos.
3. Escriba la definición de la función de calidad para modelar una conducta que consista en realizar un seguimiento de paredes. Es decir se deben promover la evolución de aquellos individuos que viajan en línea recta, evitan obstáculos y viajan paralelamente a la pared cuando ésta es encontrada.
4. Exprese por escrito sus comentarios acerca del fenómeno que se presenta cuando se usa evolución artificial y selección de acción. Específicamente cuando existe una optimización de conductas en espacio y tiempo. Considere cuales serían los resultados de modelar por medio de evolución artificial una tarea de forrajeo como una sola actividad ó descomponerlas en varias actividades usando un enfoque basado en el comportamiento. Exponga las

ventajas y las desventajas y explique sí el modelar una tarea como una sola actividad sufre de una optimización en tiempo y espacio.

5. Implemente un tipo de comportamiento emergente usando una arquitectura del tipo el que *gana-toma-todo* para una conducta social competitiva para al menos tres robots. Puede pensarse en una extensión del ejemplo presentado en este capítulo.
6. Especifique como diseñar una tarea social cooperativa para al menos dos robots cualesquiera. Explique como diseñaria el manejo de los robots. Que tipo de sensores emplearía y sí se permite la comunicación entre robots. Describa sí la robótica evolutiva puede o no ser empleada en esta tarea. Adicionalmente exponga sus argumentos para justificar la viabilidad en la elección de su diseño.
7. Realice una investigación de al menos tres tipos de plataforma robóticas comerciales existentes. Profundice su investigación al identificar el tipo de software utilizado para simular o controlar los robots. Adicionalmente por su tamaño especifique si alguna de estas plataformas pudiera ser utilizada en experimentos para niños. Como un último paso catalogue los distintos tipos de sensores que acompañan a estos robots y explique en sus propias palabras cuales considera que pueden ser los sensores más importantes, se debe notar el tipo de tarea a resolver dependiendo del ambiente, tiempo de respuesta, costo, etc. (*como un ejercicio adicional se puede realizar un experimento pensado de como aplicar lo revisado en este capítulo en un robot construido por el lector*).

Agradecimientos. Este trabajo ha sido patrocinado por CONACyT-MEXICO a través del apoyo SEP-2004-C01-45726.

Referencias

1. Dennett, D. C.: Why not the whole iguana? Behavioral and brain sciences, 1:103-104, 1978.
2. Lorenz, K.: The Foundations of Ethology. New York, Springer-Verlag, 1981.
3. Montes-Gonzalez, F., Prescott, T. J., Gurney, K., Humphries, M., Redgrave, P.: An embodied model of action selection mechanisms in the vertebrate brain. From animals to Animats 6: Proceedings of the 6th International Conference on the Simulation of Adaptive Behavior. J.A. Meyer. Cambridge, MA, MIT Press, 2000.
4. Prescott, T. J., Gurney, K, Montes-Gonzalez, F., Humphries, M., Redgrave, P.: THE ROBOT BASAL GANGLIA: Action selection by and embedded model of the basal ganglia. In Basal Ganglia VII (editor Faulls, R. et al.), New York: Kluwer Academic/Plenum Press, 2002.
5. Prescott, T. J., Montes González F. M., Gurney, K., Humphries, M., Redgrave, P.: A robot model of the basal ganglia: Behavior and intrinsic processing. Neural Networks 19, 31-61, 2006.
6. Mataric, M. J.: Situated Robotics, invited contribution to the Encyclopedia of Cognitive Science, Nature Publishing Group, Macmillan Reference Limited, 2002.

7. Brooks, R. A.: A robust layered control system for a mobile robot. IEEE Journal on Robotics and Automation, RA-2, 14-23, 1986.
8. Maes, P.: How to do the right thing. Connection Science Journal Vol. 1, no. 3, 291-323, 1989.
9. Bäck, T.: Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, 1996.
10. Holland, J.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Ann Arbor, MI: University of Michigan Press, 1975.
11. Mathworks Matlab: Matrix Laboratory Software, http://www.mathworks.com/, 2007.
12. Floreano, D., Mondana, F.: Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot. From Animals to Animats III: Proceedings of the Third International Conference on Simulation of Adaptive Behavior, MIT Press-Bradford Books, Cambridge MA, 1994.
13. Montes González, F M.: Una Metáfora Evolutiva para el desarrollo de Conductas Animales Robóticas, en la Revista Galega de Cooperación Científica Iberoamericana No. 11/2005, 18-22, 2005.
14. Montes-González, F. M., Marín-Hernández A.: Central Action Selection using Sensor Fusion. From Proceedings of the Fifth Mexican International Conference in Computer Science (ENC'04), 289-296, 2004
15. Montes González, F., Marín Hernández A., Ríos Figueroa H.: An effective robotic model of action selection. R. Marin et al. (Eds.): CAEPIA 2005, LNAI 4177, 123-32, 2006.
16. Nolfi, S., Floreano, D.: Evolutionary Robotics. The MIT Press, 2000.
17. Montes-González, F. M., Santos Reyes, J.: Evolving Robot Behavior for Centralized Action Selection, A. Gelbukh, R. Monroy. (Eds.), Advances in Artificial Intelligence Theory Research on Computing Science 16, 213-22, 2005.
18. Mitchell, M.: An Introduction to Genetic Algorithms, MIT Press, 1996.
19. Rosenblatt, F.: Principles of Neurodynamics. New York: Spartan Books, 1962.
20. Montes-González, F., Santos Reyes, J., Ríos Figueroa, H.: Integration of Evolution with a Robot Action Selection Model, A. Gelbukh and C. A. Reyes-García (Eds.): MICAI 2006, LNAI 4293, pp. 1160-1170, 2006
21. Montes-Gonzalez, F. M., Prescott, T. J., Negrete-Martinez: Minimizing human intervention in the development of the basal ganglia inspired robot control, Applied Bionics and Biomechanics, por publicarse.
22. Ebner, M.: Coevolution and the red queen effect shape virtual plants. Genetic Programming and Evolvable Machines 7, no. 1, 103-123, 2006.
23. Montes-Gonzalez, F.: The Coevolution of Robot Behavior and Central Action Selection. J. Mira and J.R. Alvarez (Eds.): IWINAC 2007, Part II, LNCS 4528, 439448, 2007.
24. Mondana, F., Franzi, E., Paolo, I.: Mobile robot miniaturisation: A tool for investigating in control algorithms. Presented at the Proceedings of the 3rd International Symposium on Experimental Robotics, Kyoto Japan, 1993.
25. Nolfi, S., Floreano D., Miglino, O., Mondada, F.: How to evolve autonomous robots: Different approaches in evolutionary robotics, Proceedings of the International Conference Artificial Life IV, Cambridge MA: MIT Press, 1994.
26. Santos, J., Duro, R.: Evolución Artificial y Robótica Autónoma. Ra-Ma Editorial, 2005.

27. Montes González, F., Bautista Cabrera, P., Escobar Ruiz, V.: The use of evolution in a central action selection model, Applied Bionics and Biomechanics, por publicarse.
28. Nolfi, S.: Evolving non-trivial behaviors on real robots: A garbage collection robot. Robotics and automation system, vol. 22, 187-98, 1997.
29. Montes Gonzalez, F., Marin Hernandez, A.: The Use of Frontal and Peripheral Perception in a Prey-Catching System, presented at the 4th International Symposium on Robotics and Automation (ISRA 2004), 482-487, 2004.

Autores de los capítulos

N	Capítulo	Autor
	Introducción	*Grigori Sidorov*
1.	Conceptos básicos de inteligencia artificial y sus relaciones	*Grigori Sidorov*
2.	Búsqueda heurística	*Elena Bolshakova*
3.	Ontologías y Representación del Conocimiento	*Manuel Vilares Ferro,* *Santiago Fernández Lanza,* *Milagros Fernández Gavilanes,* *Francisco José Ribadas Pena*
4.	Programación lógica e inteligencia artificial	*M. Vilares Ferro,* *V. M. Darriba,* *C. Gómez-Rodríguez;* *J. Vilares*
5.	Redes neuronales: consideraciones prácticas	*Sergio Ledesma Orozco*
6.	Computación evolutiva	*Juan J. Flores,* *Félix Calderón*
7.	Paradigmas emergentes en algoritmos bio-inspirados	*Efrén Mezura Montes,* *Mariana Edith Miranda Varela,* *Jorge Isaac Flores Mendoza,* *Omar Cetina Domínguez*

* Traducción de inglés realizada por Liliana Chanona Hernández, Grigori Sidorov, Germán Téllez Castillo.

Afiliaciones de los autores de los capítulos

N	Autor	Institución	Dirección
1.	*Grigori Sidorov*	Centro de Investigación en Computación, Instituto Politécnico Nacional (CIC-IPN)	Av. Juan de Dios Batiz, s/n, Col. Nueva Industrial Vallejo, 07738, Mexico D. F. sidorov@cic.ipn.mx
2.	*Elena Bolshakova*	Faculty of Computational Mathematics and Cybernetics, "Lomonosov" Moscow State University, Moscow, Russia	Faculty of Computational Mathematics and Cybernetics, "Lomonosov" Moscow State University, Vorobieovy gory, Moscow, Russia eibolshakova@gmail.com
3.	*Manuel Vilares Ferro, Santiago Fernández Lanza, Milagros Fernández Gavilanes, Francisco José Ribadas Pena; V.M. Darriba*	Área de Ciencias de la Computación e Inteligencia Artificial; Escuela Superior de Ingeniería Informática; Universidade de Vigo, España	Universidade de Vigo, Campus As Lagoas s/n, 32004 Ourense, Spain vilares@uvigo.es, sflanza@uvigo.es, mfgavilanes@uvigo.es ribadas@uvigo.es darriba@uvigo.es

N	Autor	Institución	Dirección
4.	C. Gómez-Rodríguez; J. Vilares	Department of Computer Science, University of A Coruña	University of A Coruña Campus de Elviña s/n, 15071 A Coruña, Spain cgomezr@udc.es jvilares@udc.es
5.	*Leonid Sheremetov*	Centro de Investigación en Computación, Instituto Politécnico Nacional (CIC-IPN)	Av. Juan de Dios Batiz, s/n, Col. Nueva Industrial Vallejo, 07738, Mexico D. F. sher@cic.ipn.mx
6.	*Leticia Henestrosa Carrasco*	Instituto Politécnico Nacional (IPN)	Unidad Profesional "Adolfo Lopéz Mateos", Col. Lindavista, 07738, México D. F. lhenestrosac @yahoo.com.mx
7.	*Enrique Sucar, Eduardo Morales*	Coordinación de ciencias computacionales, Intituto Nacional de Optica, Astrofísica y Electrónica (INAOE)	Luis Enrique Erro # 1, Sta. María Tonantzintla, 72840, Puebla esucar@inaoep.mx emorales@inaoep.mx
8.	*Sergio Ledesma Orozco*	Departamento de Ingenieria Electrónica, Universidad de Guanajuato	Lascuráin de Retana No. 5 Centro, Guanajuato selo@salamanca.ugto.mx

N	Autor	Institución	Dirección
9.	*Juan J. Flores, Félix Calderón*	División de Estudios de Posgrado Facultad de Ingeniería Eléctrica, Universidad Michoacana de San Nicolás de Hidalgo	Morelia, Michoacan {juanf,calderon}@umich.mx
10.	*Efrén Mezura Montes, Mariana Edith Miranda Varela, Jorge Isaac Flores Mendoza, Omar Cetina Domínguez*	Laboratorio Nacional de Informática Avanzada, (LANIA A.C.)	Rébsamen # 80 Col. Centro, 91000, Xalapa, Veracruz emezura@lania.mx, {mmiranda,jflores, ocetina}@lania.edu.mx
11.	*Jixin Ma*	School of Computing and Mathematical Sciences, The University of Greenwich, London, UK	London, SE10 9LS, U.K. j.ma@gre.ac.uk
12.	*Fernando Martín Montes González*	Departamento de Inteligencia Artificial, Universidad Veracruzana	Sebastian Camacho 5, Centro, Xalapa, Veracruz fmontes@uv.mx