

UNIDAD DIDACTICA

INTELIGENCIA ARTIFICIAL

Ricardo Aler Mur
Daniel Borrajo Millán
Andrés Silva Vázquez

Contents

1	Objetivos docentes	4
2	Introducción	5
3	Introducción a la representación	6
4	Lógica de predicados	6
4.1	El lenguaje de la lógica de predicados	6
4.2	Inferencia en lógica de predicados	11
5	Sistemas de producción	16
5.1	Estructura y funcionamiento de un sistema de producción	16
5.2	Ventajas y desventajas de los sistemas de producción	31
6	Marcos	32
6.1	Definición y propiedades de los marcos	32
6.1.1	Herencia de propiedades	33
6.1.2	Valores por omisión y excepciones	34
6.1.3	Expresión de conjunciones, disyunciones y negaciones	35
6.1.4	Referencias a otros marcos	36
6.1.5	Expresiones funcionales	36
6.1.6	Marcos temáticos	37
6.1.7	Demonios	37
6.2	Equiparación (“pattern-matching”)	38
6.3	Ejemplo de sistema de marcos	40
6.4	Ejemplo de utilización de un sistema de marcos	47
7	Redes semánticas	48
7.1	Herencia en redes semánticas	50
7.2	Expresión de conjunciones, disyunciones y negaciones	50
7.3	El mecanismo de equiparación en redes semánticas	51
8	Otros sistemas de representación	53
8.1	Guiones (“scripts”)	53
8.2	Razonamiento con incertidumbre	55
9	Ejercicios de autocomprobacion	57
10	Ejercicios propuestos no resueltos	62
11	Introducción a la búsqueda	63
12	Espacio de estados	63

13 Búsqueda no informada	65
13.1 Método de generar y comprobar	66
13.2 Búsqueda en amplitud	66
13.3 Búsqueda en profundidad	69
13.4 Búsqueda hacia atrás	70
13.5 Búsqueda bidireccional	71
13.6 Búsqueda con islas	72
13.7 Análisis de complejidad	73
14 Búsqueda heurística	75
14.1 Técnica de escalada	75
14.2 Técnicas de mejor-primero. A*	77
15 Búsqueda en situaciones con contrincantes	83
15.1 Árboles alternados	84
15.2 Método minimax para árboles alternados	86
15.3 Método Alfa-Beta: un procedimiento de poda	91
15.3.1 Introducción	91
15.3.2 El método de poda	93
15.3.3 Mejoras sobre el Alfa-Beta	95
15.4 Algoritmo sss*	99
15.5 Algoritmo B*	102
15.6 Algoritmo Max ⁿ	105
16 Ejercicios de autocomprobacion	106
17 Lecturas recomendadas	111

1 Objetivos docentes

Al finalizar la lectura de la unidad didáctica y realizar los ejercicios propuestos, se espera que el alumno sea capaz de:

- Comprender los elementos básicos de representación de los distintos formalismos de representación.
- Representar un dominio eligiendo el formalismo más adecuado.
- Realizar inferencia básica con cualquiera de los formalismos de representación.
- Comprender el funcionamiento de los diferentes algoritmos de búsqueda.
- Expandir teóricamente árboles de búsqueda.
- Codificar los algoritmos en procedimientos ejecutables.
- Analizar un problema y decidir si se puede o no aplicar búsqueda y, en caso afirmativo, qué tipo de técnica utilizar.

2 Introducción

Dentro del campo de la Informática, una de las áreas que más ha hecho evolucionar los problemas que se pueden resolver por la utilización de las computadoras ha sido la Inteligencia Artificial. El objetivo de la Inteligencia Artificial consiste en la construcción de sistemas, tanto hardware como software, que sean capaces de replicar aspectos de lo que se suele considerar “inteligencia”. Evidentemente, este objetivo está muy ligado a la definición de la propia palabra “inteligencia”, de la que existe, aproximadamente, una definición por cada persona. Esto nos obliga a adoptar un punto de vista práctico y definir la Inteligencia Artificial como el conjunto de técnicas, métodos, herramientas, y metodologías que nos ayudan a construir sistemas que se comportan igual que un humano en la resolución de problemas concretos.

Según esta definición, la Inteligencia Artificial involucra a muchos campos de investigación y desarrollo diferentes, tales como la Robótica, la Visión Artificial, la Resolución de Problemas, los Sistemas Expertos, la Traducción Automática, etc. Esta unidad didáctica se va a centrar en las técnicas básicas de realización de este tipo de sistemas, que son la representación del conocimiento y la búsqueda de soluciones. Dado que el tipo de conocimiento que poseemos las personas sobre un determinado campo no está, normalmente, estructurado según los mecanismos de representación más comúnmente utilizados en programación convencional, como pilas, listas, matrices, variables, etc., sino que están estructurados utilizando mecanismos de representación más complejos, es necesario ampliar el conjunto de técnicas computacionales básicas con un conjunto de formalismos de representación que permitan representar estas estructuras más complejas que empleamos los humanos. Estos formalismos son los que se estudiarán en el capítulo titulado **Representación del Conocimiento**.

Una vez representado el conocimiento del humano, utilizando estos formalismos de representación, es necesario resolver los mismos problemas que resuelve el humano (experto). En el capítulo **Búsqueda Heurística** se estudian una serie de métodos de selección de las mejores alternativas de decisión para poder resolver problemas. Para ello, como se estudiará más adelante, se desarrollarán árboles de búsqueda que describen las posibles alternativas que tiene la resolución de un determinado problema. Asimismo, se estudiará cómo representar y manejar el conocimiento que hace que las personas expertas resuelvan directamente problemas complejos seleccionando, en la mayor parte de las ocasiones, las mejores alternativas. Esto se hará definiendo el concepto de “heurística”, que consiste en el conocimiento que ayuda a los expertos en un campo a resolver problemas de forma más eficiente que los legos en el mismo campo.

Representación del Conocimiento

3 Introducción a la representación

Imaginemos que nunca hemos estado antes en Madrid y queremos encontrar el camino más corto para ir desde una estación de metro a otra. Lo primero que se nos ocurriría es recurrir a un plano de las líneas de metro de Madrid, que no es sino una representación de las líneas de metro reales. Algunos de los motivos por los que recurrimos a una representación en lugar de a las líneas reales de metro son evidentes. Primero porque nos resulta imposible abarcar todas las líneas de metro de un vistazo. Y segundo, porque un plano nos permite concentrarnos en los detalles que realmente nos interesan obviando detalles irrelevantes como si hay o no escaleras mecánicas, o máquinas expendedoras de billetes.

Un plano es algo que los seres humanos usan como representación simplificada del mundo externo, con el objeto de realizar ciertas tareas. De la misma manera, los sistemas de inteligencia artificial recurren a una representación interna para resolver determinados problemas. Por ejemplo, un sistema de inteligencia artificial que encontrara los caminos más cortos entre estaciones de metro debería recurrir igualmente a una representación semejante a un plano de metro. Por supuesto, una representación no nos sirve de nada si no disponemos de unos métodos que nos sirvan para manipularla y para resolver problemas usando esa representación. En nuestro caso, seríamos nosotros quienes aportaríamos el método para encontrar el camino más corto entre dos estaciones de metro, pero en el caso de un sistema inteligente deberíamos hacerle explícito ese método.

Normalmente, es el constructor del sistema el que decide qué tipo de representación es la más adecuada para el problema a resolver. En muchas ocasiones la mera elección de la representación puede hacer que un problema complejo sea más fácil de resolver.

Para que estos sistemas sean útiles, deben ser capaces de dialogar con los seres humanos o con otros sistemas. Esto significa que deben saber aceptar preguntas, resolver el problema que proponen y devolver una respuesta. Para ello, las preguntas deben convertirse a un formato adecuado a la representación interna. Y de la misma manera, la solución obtenida por el sistema debe convertirse del formato interno a una forma comprensible por el ser humano (por ejemplo, a lenguaje escrito). En la figura 1 vemos un resumen de este proceso:

En los siguientes apartados se explicarán las técnicas de representación básicas de inteligencia artificial y se verán ejemplos sobre cómo resolver problemas usando dichas técnicas.

4 Lógica de predicados

4.1 El lenguaje de la lógica de predicados

La lógica de predicados consiste en: un lenguaje para expresar proposiciones; y reglas para inferir nuevas proposiciones a partir de las que ya conocemos. Veamos como podríamos usarla para representar la información contenida en el siguiente párrafo:

D. Vito Corleone es el padrino de la principal mafia neoyorquina y su hijo, Michael Corleone, es su principal lugarteniente (o capo). Entre las aficiones de Michael se cuenta el tiro con colt 45. Aparte, se sabe que odia la pizza. Sonny Corleone es otro de los hijos del padrino. Por su parte, D. Vito tiene cierta alergia a que la policía se meta en sus negocios por lo que viene sobornando al capitán Mc Cluskey desde hace cierto

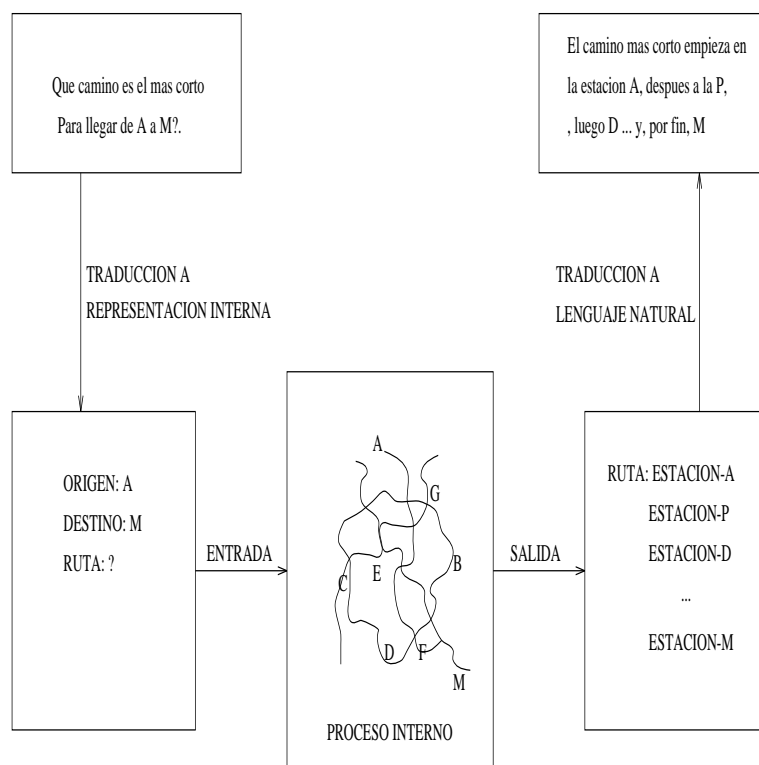


Figure 1: Esquema de representación para el problema del metro

tiempo. Pero, en un momento dado, el capitán Mc Cluskey decide traicionar al padrino. Poco tiempo después Mc Cluskey aparece muerto en una restaurante de Nueva York con dos disparos en la cabeza.

Imaginemos que queremos trabajar en el problema del asesinato. Podríamos empezar por representar la información disponible de una manera más precisa que en el párrafo anterior:

1. Vito Corleone es el Padrino
2. Vito Corleone es el padre de Michael Corleone
3. Vito Corleone es el padre de Sonny Corleone
4. Michael Corleone es capo
5. Michael Corleone usa un Colt-45)
6. Un Colt 45 es una pistola
7. Mc Cluskey es policía
8. Vito Corleone paga a Mc Cluskey)
9. Mc Cluskey traicionó a Vito Corleone

Una traducción de lo anterior a lógica de predicados podría ser la siguiente:

- Es- Padrino (Vito-Corleone)
- Es- Padre (Vito-Corleone, Michael-Corleone)
- Es- Padre (Vito-Corleone, Sonny-Corleone)
- Es- Capo (Michael-Corleone)
- Usa- Arma (Michael-Corleone, Colt-45)
- Es- Pistola (Colt-45)
- Es- Policía (Mc-Cluskey)
- Paga- a (Vito-Corleone, Mc-Cluskey)
- Traiciona (Mc-Cluskey, Vito-Corleone)

Es- Padrino, Es- Padre, Es- Capo, Usa- Arma, Es- Pistola, Es- Policía y Paga- a se llaman **predicados**. Para completar el significado de un predicado, éste necesita de argumentos. Por ejemplo, (*Vito-Corleone, Michael-Corleone*) son los argumentos del predicado Es- Padre. La función de cada uno de los argumentos viene dada por el orden que ocupa. Así, en el caso del predicado Es- Padre, el primer argumento será el padre y el segundo será el hijo. El significado de Es- Padre (Michael-Corleone, Vito-Corleone) sería por tanto completamente distinto.

La representación en calculo de predicados de un conjunto de hechos no es inmediata sino que hay que tomar una serie de decisiones. Por ejemplo, hay que decidir cuales van a ser los predicados que vamos a usar en la representación. Algunos de los hechos anteriores podrían haberse representado también de la siguiente forma:

- Es- Un (Vito-Corleone, padrino)
- Es- Un (Michael-Corleone, capo)
- Es- Un (Colt-45, pistola)
- Es- Un (Mc-Cluskey, Policía)

Queda claro que la representación de un conjunto de hechos en lógica de predicados no es única. Todavía no hemos representado el hecho de que Michael odia la pizza. Tal y como hemos venido haciendo simplemente escribiríamos:

- Odia (Michael-Corleone, pizza)

Pero imaginemos que por alguna razón queremos usar el predicado Gusta en lugar de Odia. En ese caso escribiríamos:

- NOT Gusta (Michael-Corleone, pizza)

cuyo significado es que a Michael-Corleone no le gusta la pizza (NOT corresponde al NO en español) A partir de ahora, a un predicado con argumentos, precedido o no de NOT, le llamaremos *cláusula*.

Hasta el momento, para representar que un conjunto de hechos es cierto, simplemente los hemos escrito uno debajo de otro. Sin embargo, la forma correcta de expresar que todos los hechos son ciertos al mismo tiempo es usando la palabra AND (que corresponde a la conjunción Y en español, y que llamaremos conectiva lógica):

- Es-Padrino (Vito-Corleone) AND
- Es-Padre (Vito-Corleone, Michael-Corleone) AND
- Es-Padre (Vito-Corleone, Sonny-Corleone) AND
- Es-Capo (Michael-Corleone) AND
- Usa-Arma (Michael-Corleone, Colt-45) AND
- Es-Pistola (Colt-45) AND
- Es-Policía (Mc-Cluskey) AND
- Paga-a (Vito-Corleone, Mc-Cluskey) AND
- Traiciona (Mc-Cluskey, Vito-Corleone)

En general, las conectivas lógicas son elementos que sirven para unir cláusulas o fórmulas lógicas (siendo una fórmula \wedge lógica bien una cláusula, bien un conjunto de cláusulas o \vee fórmulas lógicas unidas por conectivas). Imaginemos ahora que se ha muerto D. Vito y que no sabemos quién es el nuevo padrino, aunque sabemos que sólo uno de los dos hijos que aparecen en nuestra historia tienen alguna oportunidad de sucederle. Representaríamos este hecho con la conectiva OR (correspondiente a la conjunción española O) de la siguiente manera:

- Es-Padrino (Michael-Corleone) OR Es-Padrino (Sonny-Corleone)

La fórmula anterior expresa que al menos uno de los dos hijos es el Padrino. Pero la conectiva OR no excluye que ambas cláusulas sean ciertas al mismo tiempo. Así, la fórmula podría servir para expresar que ambos hijos pueden ser padrinos a la vez (si cada uno de ellos encabezara una facción rival, por ejemplo). Para expresar que uno de los dos hijos (pero solo uno) es el padrino podríamos escribir:

- (Es-Padrino (Michael-Corleone) AND NOT Es-Padrino (Sonny-Corleone))
OR
(Es-Padrino (Sonny-Corleone) AND NOT Es-Padrino (Michael-Corleone))

Ahora imaginemos que queremos expresar que si a Michael no le gusta la pizza entonces no es italiano. Para hacerlo introduciremos la conectiva lógica \rightarrow (correspondiente al SI ... ENTONCES ... del español). Podríamos expresarlo de la siguiente manera:

- NOT Gusta (Michael-Corleone, Pizza) \rightarrow NOT Italiano (Michael-Corleone)

A la conectiva $H \rightarrow C$ se la denomina implicación. A la parte H de la implicación se la llama antecedente o hipótesis y a la parte C se la conoce como consecuente.

Todavía nos queda por representar un hecho de nuestra narración: el que alguien ha asesinado a Mc-Cluskey usando una pistola. Podría hacerse de la siguiente manera:

- Asesina (x, Mc-Cluskey, pistola)

Su significado es que alguien (desconocido) ha asesinado a Mc Cluskey con una pistola. Para poder referirnos a un individuo x que desconocemos (en principio podría ser cualquiera) ha sido necesario introducir un nuevo elemento de la lógica de predicados: las variables. Así pues un argumento de un predicado puede ser bien una variable o bien una constante (como Mc-Cluskey o Colt45).

Sin embargo a la cláusula anterior le falta algún elemento para que exprese lo que realmente queremos decir. Para verlo más claramente supongamos que queremos representar la idea de que si a alguien no le gusta la pizza entonces no es italiano. En lógica de predicados podría quedarnos de la siguiente manera:

- NOT Gusta (x, pizza) \rightarrow NOT Italiano (x)

En este último caso estamos intentando expresar que a todo aquel al que no le guste la pizza no es italiano. Sin embargo la idea tras la fórmula sobre el asesinato era que alguien (¡pero no todo el mundo!) había asesinado a Mc Cluskey. Para completar el significado de las dos fórmulas anteriores necesitamos de otro elemento de la lógica de predicados: los cuantificadores. Se suelen usar dos tipos de cuantificadores: el existencial \exists (existe) y el universal \forall (todos). El cuantificador existencial expresa que existe al menos un valor de la variable que hace que la fórmula que le sigue sea cierta, mientras que el cuantificador universal expresa que todos los valores de la variable hacen que la fórmula sea cierta. Veamos como se usarían en las fórmulas anteriores:

- $\exists x$ Asesina (x, Mc-Cluskey, pistola)
- $\forall x$ NOT Gusta (x, pizza) \rightarrow NOT Italiano (x)

La primera fórmula expresa que alguien asesinó a Mc Cluskey con una pistola. Nótese que el cuantificador existencial tan solo afirma que existe al menos un asesino, pero no limita su número a uno (podría haber varios). La segunda fórmula afirma que todos aquellos a quienes no les guste la pizza no son italianos. Si tan solo hubiera un italiano que odiara la pizza esta fórmula sería falsa.

Para recapitular, el lenguaje de representación de la lógica de predicados consta de predicados, variables, constantes, conectivas (NOT, OR, AND, \rightarrow ...) y cuantificadores (\exists y \forall).

A continuación se recuerda la tragedia completa de la familia Corleone representada en el lenguaje de la lógica de predicados:

- Es-Padrino (Vito-Corleone)
- Es-Padre (Vito-Corleone, Michael-Corleone)
- Es-Padre (Vito-Corleone, Sonny-Corleone)
- Es-Capo (Michael-Corleone)
- Usa-Arma (Michael-Corleone, Colt-45)
- Es-Pistola (Colt-45)
- Es-Policía (Mc-Cluskey)
- Paga-a (Vito-Corleone, Mc-Cluskey)
- Traiciona (Mc-Cluskey, Vito-Corleone)
- $\exists x$ Asesina (x, Mc-Cluskey, pistola)

4.2 Inferencia en lógica de predicados

Hasta ahora todo lo que hemos hecho ha sido representar en lógica de predicados un conjunto de hechos sobre la familia Corleone y sus asociados. Pero para que esta representación nos sea útil se deberían poder extraer nuevas afirmaciones a partir de las antiguas y se debería poder responder a preguntas. Por ejemplo, estaríamos bastante interesados en saber quien fue el asesino de Mc Cluskey.

La lógica de predicados es una forma de representación bastante general y sobre ella se pueden efectuar diversos tipos de inferencia. Como ejemplos de inferencia se pueden citar la inducción, la abducción y la deducción. La deducción es el método de inferencia más propio de la lógica de predicados. A continuación se muestra un ejemplo de deducción bastante intuitivo llamado "modus ponens":

Proposiciones de partida:
1. Si a alguien no le gusta la pizza <u>entonces no es italiano</u> .
2. A Schwarzenegger no le gusta la pizza
3. Deducción: <u>Schwarzenegger no es italiano</u>

En general, el modus ponens sigue la regla:

1. Si $P \rightarrow Q$
2. P
3. Q

lo que significa que siempre que tengamos una implicación entre dos proposiciones (la fórmula 1) y un hecho (la fórmula 2) semejante al que aparece en el antecedente de la implicación (P), podemos deducir el consecuente de la implicación (Q)

En lógica de predicados la deducción previa se escribiría:

Cláusulas de partida:
1. $\forall x \text{ NOT Gusta}(x, \text{pizza}) \rightarrow \text{NOT Italiano}(x)$
2. $\text{NOT Gusta}(\text{Schwarzenegger}, \text{pizza})$
3. Cláusula deducida: $\text{NOT Italiano}(\text{Schwarzenegger})$

Más en detalle, el proceso que se ha seguido ha consistido de dos pasos:

A: Unificación y sustitución. Es necesario que el $\text{NOT Gusta}(x, \text{pizza})$ de la fórmula 1 y el $\text{NOT Gusta}(\text{Schwarzenegger}, \text{pizza})$ de la fórmula 2 sean idénticas para poder aplicar el modus ponens. La única diferencia entre ellas es que en la primera aparece una variable cuantificada universalmente (x) como primer argumento y en la segunda aparece la constante *Schwarzenegger*. Para que ambas fórmulas sean idénticas deberemos hacer que x sea igual a *Schwarzenegger*. Al proceso que hemos realizado para obtener que $x = \text{Schwarzenegger}$ se le llama *unificación*. Ahora, para que ambas fórmulas sean idénticas debemos sustituir x por *Schwarzenegger* allá donde aparezca. A este segundo proceso se le denomina *sustitución*. Así pues, tras la unificación y la sustitución, las fórmulas de partida aparecerían como:

1. $\text{NOT Gusta}(\text{Schwarzenegger}, \text{pizza})$ $\rightarrow \text{NOT Italiano}(\text{Schwarzenegger})$
2. $\text{NOT Gusta}(\text{Schwarzenegger}, \text{pizza})$

B: Modus ponens. Y ahora ya podemos aplicar el modus ponens propiamente dicho:

1. $\text{NOT Gusta}(\text{Schwarzenegger}, \text{pizza}) \rightarrow \text{NOT Italiano}(\text{Schwarzenegger})$
2. $\text{NOT Gusta}(\text{Schwarzenegger}, \text{pizza})$
3. $\text{NOT Italiano}(\text{Schwarzenegger})$

En general no basta con aplicar una vez el modus ponens para alcanzar una conclusión, sino que hay que aplicar una cadena de razonamientos. Ahora volvamos al asunto del asesinato de Mc Cluskey.

Por el periódico ya sabemos que Mc Cluskey fue asesinado con una pistola (aunque la identidad del asesino sigue siendo desconocida). Pero lo cierto es que el hábil detective Harry el sucio ya había deducido que el asesinato tendría lugar e incluso conocía la identidad del asesino. Harry había llegado a la conclusión de que D. Vito tiene reservados dos tipos de asesinato para los que le traicionan. Si el traidor no está a sueldo de la familia entonces D. Vito le aplica la venganza conocida como beso

Calabrés. Pero si el traidor está "financiado" por la familia, D. Vito considera que el castigo debe ser ejemplar, una muerte especialmente cruel que el denomina abrazo Siciliano. Obviando los detalles truculentos, el abrazo Siciliano debe ser llevado a cabo por algún hijo de D. Vito que sea capo. En cambio, el beso Calabrés puede ser ejecutado por cualquier matón a sueldo.

Esta y alguna otra información relativa al caso podría representarse de la siguiente manera:

1. Si alguien traiciona a D. Vito y está a sueldo suyo Entonces se le aplicará el Abrazo Siciliano.
2. Si alguien traiciona a D. Vito pero no está a sueldo suyo Entonces se le aplicará un castigo menor, el Beso Calabrés.
3. El Abrazo Siciliano solo puede ser aplicado por un hijo de D. Vito que además sea capo. Si se cumplen esas condiciones y además el susodicho hijo sabe manejar un arma determinada, entonces se producirá un asesinato del traidor con ese arma a manos del capo.
4. El Beso calabrés puede ser aplicado por cualquier sicario a sueldo de D. Vito. Siempre que tengamos un sicario tal que sepa manejar un arma determinada, se producirá un asesinato del traidor con ese arma a manos del sicario.
5. Puesto que una Colt 45 es una pistola, siempre que se produzca un asesinato con una Colt 45, también se podrá decir que se ha producido un asesinato con una pistola.

A continuación se representa la información anterior en el lenguaje de la lógica de predicados:

$$\forall ?traidor$$

$$\text{Traiciona} (?traidor, \text{Vito-Corleone}) \text{ AND}$$

$$\text{Paga-A} (\text{Vito-Corleone}, ?traidor)$$

$$\rightarrow \text{Abrazo-Siciliano} (?traidor)$$

$$\forall ?traidor \text{Traiciona} (?traidor, \text{Vito-Corleone}) \text{ AND}$$

$$\text{NOT } \text{Paga-A} (\text{Vito-Corleone}, ?traidor)$$

$$\rightarrow \text{Beso-Calabres} (?traidor)$$

$$\forall ?traidor, ?hijo, ?arma \text{Abrazo-Siciliano} (?traidor) \text{ AND}$$

$$\text{Padre} (\text{Vito-Corleone}, ?hijo) \text{ AND}$$

$$\text{Capo} (?hijo) \text{ AND}$$

$$\text{Usa-Arma} (?hijo, ?arma)$$

$$\rightarrow \text{Asesina} (?hijo, ?traidor, ?arma)$$

\forall ?traidor, ?sicario, ?arma Beso-Calabres (?traidor) AND Es-Maton (?sicario) AND Paga-A (Vito-Corleone, ?sicario) AND Usa-Arma (?sicario, ?arma) \rightarrow Asesina (?sicario, ?traidor, ?arma)

\forall ?arma, ?gangster, ?victima Pistola (?arma) AND Asesina (?gangster, ?victima, ?arma) \rightarrow Asesina (?gangster, ?victima, Pistola)

Nótese que en la anterior representación, a las palabras que queríamos usar como nombres de variable les hemos antepuesto un interrogante. Los nombres de las variables son tan solo ayudas para que nos resulte sencillo entender las fórmulas. Durante la deducción, una variable como ?sicario sería tan poco significativa para el proceso de deducción como otra llamada ?x.

En los siguientes párrafos se muestra una cascada de deducciones por modus ponens sobre el asesinato de Mc Cluskey:

\forall ?traidor Traiciona (?traidor, Vito-Corleone) AND Paga-A (Vito-Corleone, ?traidor) \rightarrow Abrazo-Siciliano (?traidor) Traiciona (Mc-Cluskey, Vito-Corleone) Paga-A (Vito-Corleone, Mc-Cluskey)
Abrazo-Siciliano (Mc-Cluskey) (Hemos sustituido ?traidor por Mc-Cluskey)

\forall ?hijo, ?arma, ?traidor Abrazo-Siciliano (?traidor) AND Padre (Vito-Corleone, ?hijo) AND Capo (?hijo) AND Usa-Arma (?hijo, ?arma) \rightarrow Asesina (?hijo, ?traidor, ?arma) Abrazo-Siciliano (Mc-Cluskey) ^a Capo (Michael-Corleone) Usa-Arma (Michael-Corleone, Colt-45) ^b
Asesina (Michael-Corleone, Mc-Cluskey, Colt-45) (Hemos sustituido ?arma por Colt-45 e ?hijo por Michael-Corleone)

^aSabemos que esto es cierto por la anterior deducción

^bSabemos que *Capo(Michael-Corleone)* y *Usa-Arma(Michael-Corleone, Colt-45)* son ciertos por la representación que hicimos inicialmente

\forall ?gangster, ?victima, ?arma Pistola (?arma) AND Asesina (?gangster, ?victima, ?arma) \rightarrow Asesina (?gangster, ?victima, Pistola) Pistola (Colt-45) Asesina (Michael-Corleone, Mc-Cluskey, Colt-45)
Asesina (Michael-Corleone, Mc-Cluskey, Pistola) (hemos sustituido ?arma por Colt-45 ?gangster por Michael-Corleone y ?victima por Mc-Cluskey)

Se debe hacer notar que la representación que hemos usado no expresa exactamente lo que realmente queríamos expresar. En efecto, imaginemos que el otro hijo de D. Vito, Sonny Corleone, también es capo. En ese caso, y por el mismo proceso, podríamos haber deducido *Asesina (Sonny-Corleone, Mc-Cluskey, pistola)*¹. Vemos que aunque lo que nosotros queríamos expresar era que solo uno de los capos hijos de D. Vito debían cometer el asesinato, lo que hemos obtenido es que **todos** los hijos capos de D. Vito cometerán el asesinato. Se debe tener mucho cuidado a la hora de representar la realidad en lógica de predicados (y en otros formalismos también) para que realidad y representación coincidan.

Resumiendo, la lógica de predicados consta de:

- Un lenguaje para representar problemas y

¹Siempre que Sonny Corleone supiera usar una pistola, claro está

- Procedimientos para obtener conclusiones sobre los hechos representados.

Ya hemos visto uno de esos procedimientos (la deducción por modus ponens), pero existen otros (más abstractos y automatizados) como la resolución. La resolución sigue un proceso bastante curioso para demostrar que algo es cierto: simplemente comienza por afirmar que lo que se pretende demostrar es falso e intenta alcanzar una contradicción aplicando repetidamente una regla de deducción (la de resolución).

La lógica de predicados es una representación bastante general, pero tiene varias desventajas. En primer lugar, el método más usado para extraer conclusiones y responder preguntas en lógica de predicados es el método de resolución. Desgraciadamente el método de resolución sufre del llamado problema de explosión combinatoria: cuando hay muchas fórmulas, el número de inferencias (llamadas resoluciones) que se deben realizar para alcanzar alguna conclusión se vuelve demasiado grande.

Otra desventaja es que la lógica de predicados no impone una forma única de representar el conocimiento. Ya vimos que el hecho de que algo es una pistola podía representarse tanto como Pistola (Colt45) como Es-Un (pistola, Colt45). Mientras unas formas de representación harán que el problema se resuelva casi inmediatamente, otras lo harán innecesariamente complicado. Por último, la lógica de predicados no permite estructurar el conocimiento. Por ejemplo, si queremos afirmar varias propiedades acerca de una pistola, cada una de ellas se representará en una fórmula distinta (por ejemplo: Pistola (Colt45), Negra (Colt45), Pesada (Colt45), Posee (Michael-Corleone, Colt45), etc).

5 Sistemas de producción

5.1 Estructura y funcionamiento de un sistema de producción

Aunque no se hizo explícito, cuando veíamos la inferencia por modus ponens se podía observar que existían dos tipos de fórmulas: afirmaciones o hechos (como Usa-Arma (Michael-Corleone, Colt-45)) e implicaciones (como NOT Gusta(x, Pizza) \rightarrow NOT Italiano (x)). El proceso de deducción por modus ponens consistía simplemente en aparear hechos con implicaciones para obtener nuevas conclusiones. Los sistemas de producción intentan generalizar la aproximación anterior. Así, un sistema de producción consta de:

- Una base de hechos
- Una base de implicaciones (llamadas producciones o reglas)
- Un mecanismo de control (o motor de inferencia)

Al contrario que en lógica de predicados, en sistemas de producción se realiza la suposición de que si un hecho no está presente en la base de hechos, ese hecho es falso.

En la figura 2 se puede ver el asesinato de Mc Cluskey estructurado como un sistema de producción:

La base de reglas y la base de hechos son bastante intuitivas, por lo que no se entrará en más detalle. El tercer elemento de un sistema de producción, el motor de inferencia sigue un ciclo que, de manera simplificada, consta de los siguientes pasos:

1. Obtener el conjunto de posibles reglas que se puede combinar con algún hecho de la base de hechos. Esta fase se denomina *equiparación* (o emparejamiento) de reglas con hechos. A las reglas emparejadas se las denomina reglas activas.

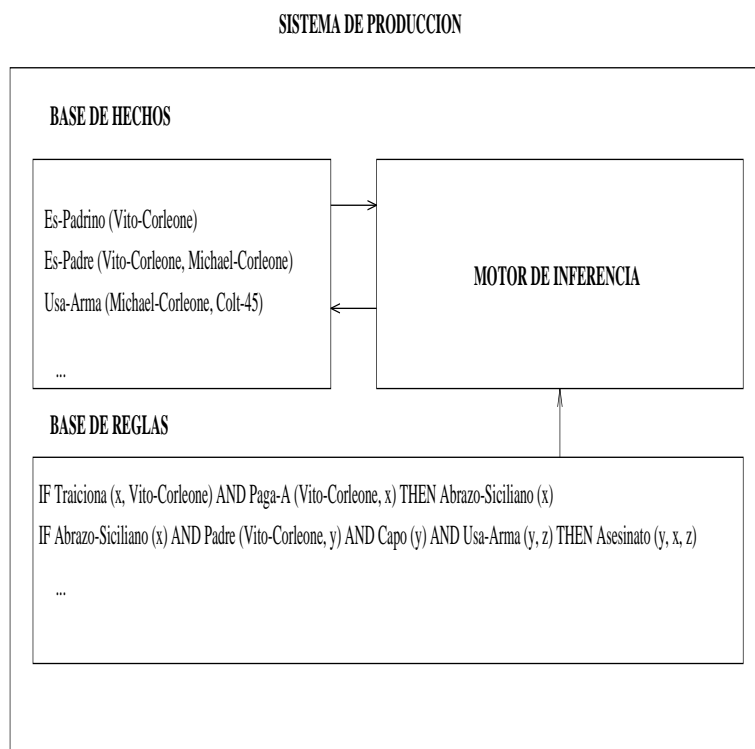


Figure 2: Esquema de sistema de producción

2. Seleccionar alguno (o algunos) de los emparejamientos obtenidos en el paso anterior. Este paso se denomina *resolución de conflictos*.
3. Disparar los emparejamientos seleccionados en la fase 2 y modificar la base de hechos según dictan los consecuentes de las reglas activas disparadas.

El ciclo termina cuando en la base de hechos aparece el hecho que resuelve el problema (en el caso del asesinato de Mc Cluskey, cuando apareciera en la base de hechos *Asesina (Michael Corleone, Mc Cluskey, Pistola)*).

Pero como ya se ha dicho, los sistemas de producción son una generalización del proceso de deducción lógica. En deducción, el paso 3 consistiría exclusivamente en añadir nuevos hechos a la base de hechos. Cuando trabajamos en lógica se supone que las fórmulas de las que partimos (también llamadas axiomas) son consistentes, es decir, que no van a generar hechos que estén en contradicción con los ya existentes. Pero en sistemas de producción los consecuentes de las reglas son acciones a ejecutar sobre la base de hechos. En particular esas acciones pueden ser añadir nuevos hechos (como en deducción) o borrar hechos de la base. Podría ocurrir que se dispararan dos reglas, una de las cuales intentara añadir un hecho a la base y otra intentara borrarlo. Es debido a casos como éste que es necesario el paso 2 del ciclo - seleccionar una o varias reglas de entre todas las activas - en lugar de disparar todas las reglas activas². En otras ocasiones conviene que la base de reglas pueda contener producciones inconsistentes y dejar al sistema de resolución de conflictos que decida que regla se va a disparar. A veces, por cuestiones de eficiencia o para

²Una regla activa es aquella cuyos antecedentes se pueden emparejar a elementos presentes en la base de hechos

comprender mejor el comportamiento del sistema, se desea que tan solo se dispare una regla en cada ciclo de entre todas las posibles. Veamos un ejemplo demostrativo de la utilidad de la resolución de conflictos.

Supongamos que Don Vito ha decidido construir un robot mafioso para que le solucione algunos asuntos rutinarios. Don Vito ha introducido la siguiente base de reglas en el robot³

Regla 1	
IF	Traiciona (?traidor, Vito-Corleone) AND Paga-A (Vito-Corleone, ?traidor)
THEN	Añade (Abrazo-Siciliano (?traidor))

Regla 2	
IF	Traiciona (?traidor, Vito-Corleone) AND NOT Paga-A (Vito-Corleone, ?traidor)
THEN	Añade (Beso-Calabres (?traidor))

Regla 3	
IF	Traiciona (?traidor, Vito-Corleone) AND Paga-A (Vito-Corleone, ?traidor) AND NOT Amigo-Personal (?traidor, Vito-Corleone)
THEN	Añade (Estrujon-Siciliano (?traidor))

Partamos de una base inicial de hechos como la siguiente:

A Traiciona (Mc-Cluskey, Vito-Corleone)
B Paga-A (Vito-Corleone, Mc-Cluskey)

Siguiendo el ciclo de los sistemas de producción, primero emparejaríamos las condiciones de las reglas con los hechos. Podríamos encontrar las siguientes parejas⁴:

Regla 1. con hechos A y B	
IF	Traiciona (Mc-Cluskey, Vito-Corleone) AND Paga-A (Vito-Corleone, Mc-Cluskey)
THEN	Añade (Abrazo-Siciliano (McCluskey))

³La implicación lógica se representaba con el símbolo \rightarrow , mientras que en sistemas de producción las reglas se representan por medio de las palabras **IF** ... **THEN**, correspondientes al **SI** ... **ENTONCES** en español. Además, en sistemas de producción, todas las variables se suponen cuantificadas por \forall , por lo que dicho cuantificador no aparece en la regla

⁴Nótese que ya se han hecho las unificaciones y sustituciones de variables necesarias

Regla 3 con hechos A y B	
IF	Traiciona (Mc-Cluskey, Vito-Corleone) AND Paga-A (VitoCorleone, Mc-Cluskey) AND NOT Amigo-Personal (Vito-Corleone, Mc-Cluskey)
THEN	Añade (Estrujon-Siciliano (Mc-Cluskey))

En este momento se pueden hacer dos observaciones. Primero, al contrario que en lógica de predicados, en sistemas de producción se hace la suposición de que si un hecho no está presente en la base de hechos, ese hecho es falso. Como el hecho Amigo-Personal (Vito-Corleone, Mc-Cluskey) no aparece en la base de hechos, se supone que Mc-Cluskey no es amigo personal de Don Vito, por lo que se puede activar el segundo emparejamiento (la regla 3 con los hechos A y B).

Ahora observemos qué sucedería si aplicáramos las dos reglas activas (la 1 y la 3) al tiempo. Simplemente aparecerían en la base de hechos dos nuevos hechos: Abrazo-Siciliano (Mc-Cluskey) y Estrujon-Siciliano (Mc-Cluskey). Si el robot usara los hechos Abrazo-Siciliano o Estrujon-Siciliano para realizar sus acciones subsiguientes, es evidente que en este momento se encontraría hecho un lío. Es en este punto donde el paso 3 del ciclo, la resolución de conflictos resolvería el problema. Lo más inmediato sería decidirse por la regla más específica, es decir, aquella que requiere más condiciones. En nuestro caso, la regla 3 requiere las mismas condiciones que la 2 y además requiere la condición NOT Amigo-Personal (Vito-Corleone, Mc-Cluskey). Por ello decimos que es más específica. Intuitivamente parece lógico destinar el castigo más desagradable (el estrujón siciliano) a alguien que no es amigo personal, siendo iguales el resto de las condiciones.

A continuación se discuten otras estrategias de resolución de conflictos:

1. Seleccionar la primera regla que se equipare con la base de hechos. Esto presupone que las reglas de la base de reglas han sido ordenadas de acuerdo a algún criterio.
2. Seleccionar la regla de prioridad más alta. Es un criterio similar al anterior, salvo que a cada regla se le asigna un número que representa la prioridad.
3. La regla más específica, es decir, aquella que se pueda equiparar con un mayor número de hechos en la base de hechos.
4. La regla que concierne al hecho añadido más recientemente a la base de hechos.
5. Seleccionar aquella regla que nunca se ha disparado (o lo contrario, seleccionar aquella que se disparó más recientemente).
6. Seleccionar una regla al azar.
7. Explorar en paralelo los disparos de todas las reglas que equiparan con hechos de la base de hechos.

En general estas estrategias se programan directamente en el motor de inferencia. Otra posibilidad más flexible sería tener otro conjunto de reglas (llamadas meta-reglas o conocimiento de control) que determinarían qué reglas de la base de reglas deberían dispararse en cada momento.

Supongamos que Don Vito le quiere encargar un trabajo fácil a su robot mafioso. Le ha dispuesto un escenario de tres habitaciones. En la habitación C ha colocado una pistola y en la habitación A ha colocado un muñeco con gran parecido a Mc Cluskey. El robot mismo comienza en la habitación B (véase la figura 3. En este caso, la base de hechos contendrá el estado del mundo,

y las reglas contendrán las formas de transformar el mundo (o lo que es lo mismo, de transformar la base de hechos). Así pues, la base de hechos inicial podría contener lo siguiente:

H1: Esta (pistola, C)
H2: Esta (robot, B)
H3: Esta (figura-Mc-Cluskey, A)
H4: Hay-Puerta (B, A)
H5: Hay-Puerta (B, C)

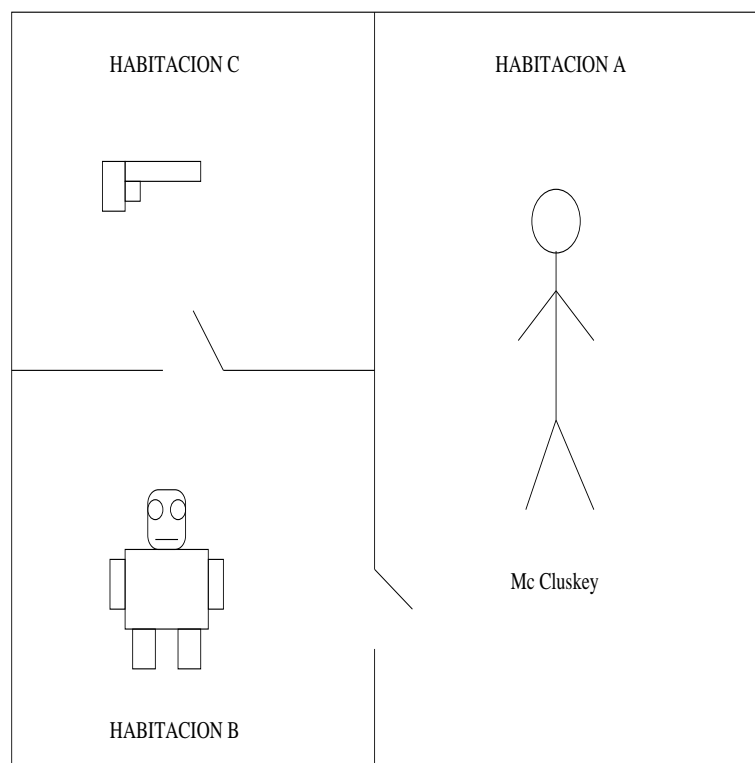


Figure 3: Escena en la que se debe desenvolver el robot de Don Vito

La base de reglas podría ser:

Regla 1.	
IF	Esta (robot, ?sitio) AND Esta (?objeto, ?sitio)
THEN	Añadir (Tiene (robot, ?objeto)), Quitar (Esta (?objeto, ?sitio))

Es decir, si el robot está en el mismo lugar que un objeto, entonces puede cogerlo

Regla 2	
IF	Esta (robot, ?sitio1) AND Hay-Puerta (?sitio1, ?sitio2)
THEN	Quitar (Esta (robot, ?sitio1)), Añadir (Esta(robot, ?sitio2))

O lo que es lo mismo, si el robot está en una habitación que tiene una puerta que comunica con otra habitación, entonces el robot puede pasar a esa nueva habitación.

Regla 3.	
IF	Hay-Puerta (?sitio1, ?sitio2)
THEN	Añadir (HayPuerta (?sitio2, ?sitio1))

Es decir, si se puede pasar de una habitación a otra, entonces también se puede pasar desde esa otra habitación a la primera.

Regla 4	
IF	Esta (figura-Mc-Cluskey, ?sitio) AND Esta (robot, ?sitio) AND NOT Tiene (robot, pistola)
THEN	Quitar (Esta (robot, ?sitio))

La regla número 4 especifica que en caso de que el robot se presentara ante Mc-Cluskey sin la pistola, Mc-Cluskey mataría al robot (lo quitaría del mundo) y el problema ya no tendría solución: el robot habría fracasado.

Regla 5	
IF	Esta (robot, ?sitio) AND Esta (figura-Mc-Cluskey, ?sitio) AND Tiene (robot, pistola)
THEN	Quitar (Esta (McCluskey, ?sitio))

La regla número 5 especifica que caso de que Mc Cluskey y el robot se encuentren en el mismo sitio y el robot tenga la pistola, el robot mata a Mc Cluskey (desaparece del mundo)

El sistema de producción ejecutará sus ciclos repetidamente hasta que desaparezca de la base de hechos el hecho Esta (Mc-Cluskey, ?sitio) o Esta (robot, ?sitio) (donde sitio representa cualquier habitación). En el primer caso el robot habría triunfado y en el segundo habría fracasado.

Y ahora veamos el comportamiento del sistema de producción ciclo a ciclo:

PASO 1: Las reglas que se pueden emparejar con la base de hechos inicial son R2 y R3:

Regla 2. con H2-H5	
IF	Esta (robot, B) AND Hay-Puerta (B, C)
THEN	Quitar (Esta (robot, B)), Añadir (Esta (robot, C))

Regla 2. con H2-H4	
IF	Esta (robot, B) AND Hay-Puerta (B, A)
THEN	Quitar (Esta (robot, B)), Añadir (Esta (robot, A))

Regla 1. con H2-H2	
IF	Esta (robot, C) AND Esta (robot, C)
THEN	Añadir (Tiene (robot, robot)), Quitar (Esta (robot, robot))

Regla 3. con H5	
IF	Hay-Puerta (B, A)
THEN	Añadir (Hay-Puerta (A, B))

Regla 3. con H4	
IF	Hay-Puerta (B, C)
THEN	Añadir (Hay-Puerta (C, B))

Nótese que una misma regla se puede emparejar con distintos conjuntos de hechos (R2 se empareja dos veces y lo mismo le ocurre a R3)

PASO 2: Nótese que en este sistema de producción si que se producirían conflictos si se disparasen todas las reglas al mismo tiempo. Si se disparasen los dos primeros emparejamientos (H2H4:R2 y H2-H5:R2) en la base de hechos aparecerían los hechos Esta (robot, A) y Esta (robot, C) lo cual es claramente inconsistente. Por tanto deberemos establecer una estrategia de resolución de conflictos. En principio nos limitaremos a escoger una sola regla al azar. Imaginemos que por casualidad seleccionamos el emparejamiento H2-H4:R2.

PASO 3: Ahora debemos disparar el emparejamiento seleccionado anteriormente (H2-H4:R2) y actualizar la base de hechos según indique su consecuente. Nuestra base de hechos habrá cambiado a:

H1: Esta (pistola, C)
H2: Esta (robot, A)
H3: Esta (figura-Mc-Cluskey, A)
H4: Hay-Puerta (B, A)
H5: Hay-Puerta (B, C)

Como tanto el robot como Mc Cluskey siguen existiendo (hechos H2 y H3), todavía no hemos cumplido con la condición de finalización y el ciclo debería volver a comenzar:

PASO 1:

Regla 4. con H2-H3	
IF	Esta (figura-Mc-Cluskey, A) AND Esta (robot, A) AND NOT Tiene (robot, pistola)
THEN	Quitar (Esta (robot, ?sitio))

Regla 3. con H5	
IF	Hay-Puerta (B, A)
THEN	Añadir (Hay-Puerta (A, B))

Regla 3. con H4	
IF	Hay-Puerta (B, C)
THEN	Añadir (Hay-Puerta (C, B))

PASO 2: Supongamos que se selecciona H2-H3:R4

PASO 3: Al disparar H2-H3 desaparecería de la base de hechos el hecho Esta (robot, A). Como ésta era una de nuestras condiciones de terminación con fracaso, el motor de inferencia simplemente indicaría que había sido incapaz de resolver el problema y terminaría. Pero evidentemente el conjunto de reglas suministrado al robot era suficiente para que éste llevara a cabo su tarea. La razón del fracaso está en que la estrategia de resolución de conflictos ha aplicado las reglas en un orden inapropiado y ha llegado a un callejón sin salida. En un sistema con pocas reglas como éste, la intuición nos dice en que orden deberíamos aplicar las reglas. Pero imaginemos que tuvieramos cientos de reglas. En ese caso solo tendríamos dos posibilidades: o bien nuestras estrategias de resolución de conflictos son muy inteligentes o bien al sistema de producción se le permite volver a un punto donde tomó una decisión fallida (una selección de regla) y tomar otra decisión. De esta manera podríamos ver la evolución de un sistema de producción como la exploración de un árbol en la que las bases de hechos en cada instante de tiempo serían los nodos y los arcos indicarían las reglas que se habían disparado para pasar de una base de hechos a otra. En la figura 4 se puede ver el árbol explorado hasta el momento por nuestro sistema de producción.

En la figura 4, el nodo hoja indica nuestra situación actual, que como habíamos visto es un callejón sin salida, por lo que habría que retroceder un nodo hacia atrás. Una vez restaurada la base de hechos, volveríamos a ejecutar el ciclo (pero esta vez ya no seleccionaríamos la regla H2-H4:R2 que nos había conducido al fallo).

PASO 1:

Regla 2. con H2-H5	
IF	Esta (robot, B) AND Hay-Puerta (B, C)
THEN	Quitar (Esta (robot, B)), Añadir (Esta (robot, C))

Regla 2. con H2-H4 (Falló)	
IF	Esta (robot, B) AND Hay-Puerta (B, A)
THEN	Quitar (Esta (robot, B)), Añadir (Esta (robot, A))

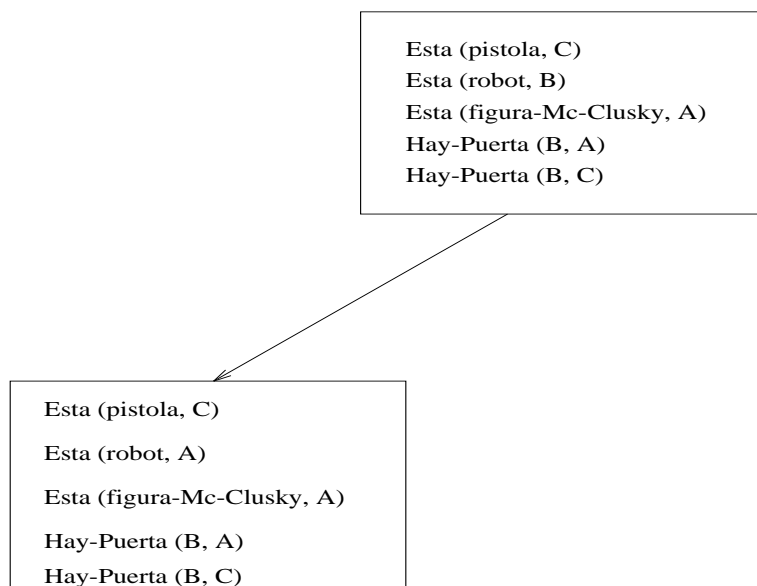


Figure 4: Arbol de búsqueda parcial

Regla 3. con H5	
IF	Hay-Puerta (B, A)
THEN	Añadir (Hay-Puerta (A, B))

Regla 3. con H4	
IF	Hay-Puerta (B, C)
THEN	Añadir (Hay-Puerta (C, B))

PASO 2: A partir de ahora (para ahorrar espacio) supondremos que nuestra estrategia de resolución de conflictos siempre va a seleccionar la regla correcta en cada situación. En la realidad sería prácticamente imposible alcanzar esta perfección, por lo que serían necesarias muchas vueltas atrás en cuanto el motor de inferencia se encontrara en un callejón sin salida. En este punto supondremos que nuestro motor de inferencia seleccionará H2H5:R2).

PASO 3: Nuestra base de hechos pasaría a ser:

H1: Esta (pistola, C)
H2: Esta (robot, C)
H3: Esta (figura-Mc-Cluskey, A)
H4: Hay-Puerta (B, A)
H5: Hay-Puerta (B, C)

Y el ciclo volvería a empezar:

PASO 1: Reglas que se podrían disparar:

Regla 1	
IF	Esta (robot, C) AND Esta (pistola, C)
THEN	Añadir (Tiene (robot, pistola)), Quitar (Esta (pistola, C))

Regla 2	
IF	Hay-Puerta (B, A)
THEN	Hay-Puerta (A, B)

Regla 3	
IF	Hay-Puerta (B, C)
THEN	Hay-Puerta (C, B)

PASO 2: Supongamos que disparamos la regla 1

PASO 3: La nueva base de hechos será:

H1: Tiene (robot, pistola)
H2: Esta (robot, C)
H3: Esta (figura-Mc-Cluskey, A)
H4: Hay-Puerta (B, A)
H5: Hay-Puerta (B, C)

PASO 1:

Regla 1	
IF	Hay-Puerta (B, A)
THEN	Hay-Puerta (A, B)

Regla 2	
IF	Hay-Puerta (B, C)
THEN	Hay-Puerta (C, B)

PASO 2: Supongamos que se selecciona la regla 2.

PASO 3:

H1: Tiene (robot, pistola)
H2: Esta (robot, C)
H3: Esta (figura-Mc-Cluskey, A)
H4: Hay-Puerta (B, A)
H5: Hay-Puerta (B, C)
H6: Hay-Puerta (C, B)

PASO 1:

Regla 1
IF Esta (robot, C) AND Hay-Puerta (C, B)
THEN Quitar (Esta (robot, C)), Añadir (Esta (robot, B))

Regla 2
IF Hay-Puerta (B, A)
THEN Hay-Puerta (A, B)

Regla 3
IF Hay-Puerta (B, C)
THEN Hay-Puerta (C, B)

Regla 4
IF Hay-Puerta (C, B)
THEN Hay-Puerta (B,

PASO 2: Supongamos que se selecciona la regla 1

PASO 3:

H1: Tiene (robot, pistola)
H2: Esta (robot, B)
H3: Esta (figura-Mc-Cluskey, A)
H4: Hay-Puerta (B, A)
H5: Hay-Puerta (B, C)
H6: Hay-Puerta (C, B)

PASO 1:

Regla 1
IF Esta (robot, B) AND Hay-Puerta (B, A)
THEN Quitar (Esta (robot, A)), Añadir (Esta (robot, A))

Regla 2	
IF	Esta (robot, B) AND Hay-Puerta (B, C)
THEN	Quitar (Esta (robot, B)), Añadir (Esta (robot, C))

Regla 3	
IF	Hay-Puerta (B, A)
THEN	Hay-Puerta (A, B)

Regla 4	
IF	Hay-Puerta (B, C)
THEN	Hay-Puerta (C, B)

Regla 5	
IF	Hay-Puerta (C, B)
THEN	Hay-Puerta (B, C)

PASO 2: Supongamos que se dispara la regla 1

PASO 3:

H1: Tiene (robot, pistola)
H2: Esta (robot, A)
H3: Esta (figura-Mc-Cluskey, A)
H4: Hay-Puerta (B, A)
H5: Hay-Puerta (B, C)
H6: Hay-Puerta (C, B)

PASO 1:

Regla 1	
IF	Esta (robot, A) AND Esta (figura-Mc-Cluskey, A) AND Tiene (robot, pistola)
THEN	Quitar (Esta (Mc-Cluskey, A))

Regla 2	
IF	Hay-Puerta (B, A)
THEN	Hay-Puerta (A, B)

Regla 3	
IF	Hay-Puerta (B, C)
THEN	Hay-Puerta (C, B)

Regla 4	
IF	Hay-Puerta (C, B)
THEN	Hay-Puerta (B, C)

PASO 2: Supongamos que se selecciona una vez más la regla que nos conviene (la 1)

PASO 3:

H1: Tiene (robot, pistola)
H2: Esta (robot, A)
H4: Hay-Puerta (B, A)
H5: Hay-Puerta (B, C)
H6: Hay-Puerta (C, B)

Y en este momento se cumple una de las condiciones de terminación con éxito: Mc Cluskey ha desaparecido de la base de hechos. El árbol de búsqueda de este sistema de producción aparece en la figura 5. Nótese que no hemos necesitado usar la vuelta atrás en ningún otro momento puesto que hemos ido seleccionando las reglas que nos convenían. Pero podría haber ocurrido que estando el robot en C con la pistola a su alcance, se hubiera vuelto por donde había venido antes de coger el arma, y hubiera pasado de C a B y de ahí a A encontrándose de nuevo con Mc Cluskey. Como el robot no llevará el arma, Mc Cluskey habría matado al robot, con lo que el motor de inferencia habría dado por terminado el problema con un fracaso. En ese caso una vuelta atrás hubiera sido necesaria.

También podría haber ocurrido que el robot hubiera estado yendo y viniendo de la habitación C a la B, sin cumplir nunca su objetivo de matar a Mc Cluskey. Para evitar que el sistema de producción entre en ciclos infinitos como el anterior, nuestra estrategia de resolución de conflictos podría intentar disparar una regla que no se hubiera disparado antes, o disparar la regla que se disparó hace más tiempo. Observese que una cosa es la representación del problema y otra muy diferente el conocimiento o el método necesario para resolver el problema.

Del ejemplo anterior se dice que sigue un modo de inferencia con encadenamiento hacia adelante porque intenta pasar desde una situación inicial a una situación final deseada (u objetivo) por aplicación de reglas. La base de hechos contiene el estado del mundo en cada momento y las reglas indican cómo hacer transiciones entre estados del mundo. Pero ésta no es la única posibilidad. Imaginemos que partimos del objetivo y que intentamos llegar a la situación inicial aplicando las reglas al revés. Este modo de inferencia se denomina de encadenamiento hacia atrás. Veamos un ejemplo:

Para simplificar, imaginemos que tan solo le ordenamos a nuestro robot que coja la pistola que está en la habitación C . ¿Qué regla nos permite hacer esto?, o lo que es lo mismo, ¿Que regla tiene Añadir (Tiene (robot, pistola)) en el consecuente?. La regla R1 tiene Añadir (Tiene (robot, ?objeto)) que se puede emparejar con Tiene (robot, pistola) haciendo la sustitución ?objeto=pistola:

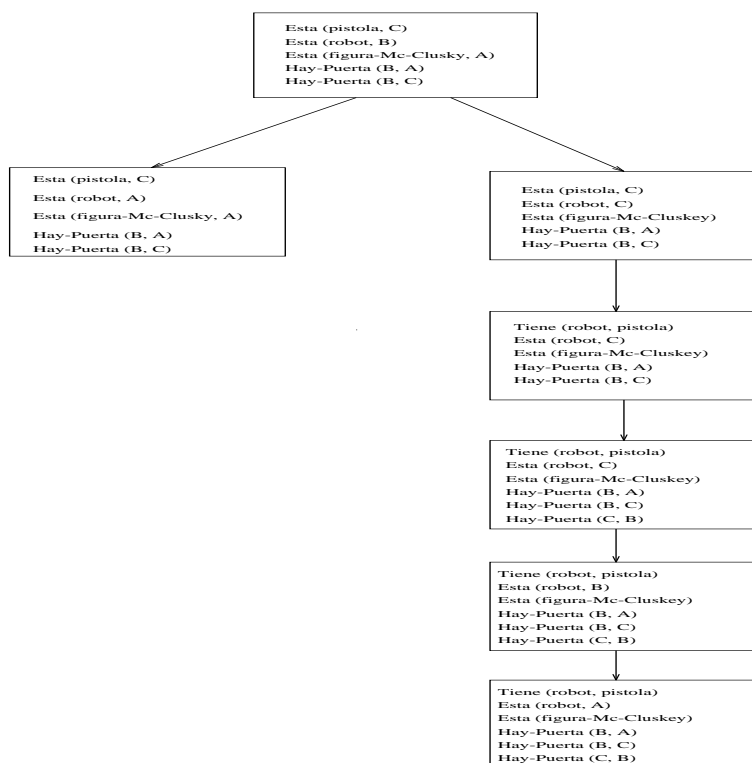


Figure 5: Arbol de búsqueda completo

Regla 1/?objeto=pistola:	
IF	Esta (robot, ?sitio) AND Esta (pistola, ?sitio)
THEN	Añadir (Tiene (robot, pistola)), Quitar (Esta (pistola,?sitio))

Pero para poder añadir ese hecho se deben cumplir las dos condiciones que aparecen en el antecedente de R1: Esta (pistola, ?sitio) y Esta (robot, ?sitio). Si hacemos la sustitución ?sitio=C, la primera condición se cumple en nuestra base de hechos inicial puesto que tenemos Esta (pistola, C), pero no se cumple que Esta (robot, C). Así pues, para poder añadir el hecho Tiene (robot, pistola) hemos llegado a la conclusión de que se debe añadir el hecho Esta (robot, C) a la base de hechos. ¿Cómo haremos para añadirlo?. Pues igual que antes, buscaremos una regla que tenga en su consecuente Añadir (Esta (robot, C)). La regla R2 podría servirnos:

Regla 2	
IF	Esta (robot, ?sitio1) AND Hay-Puerta (?sitio1, ?sitio2)
THEN	Quitar (Esta (robot, ?sitio1)), Añadir (Esta (robot, ?sitio2))

Para ello debemos unificar la variable ?sitio2 con C. Así, haciendo la sustitución ?sitio2=C tendríamos:

Regla 2/?sitio2=C	
IF	Esta (robot, ?sitio1) AND Hay-Puerta (?sitio1, C)
THEN	Quitar (Esta (robot, ?sitio1)), Añadir (Esta (robot, C))

Ahora tendremos que satisfacer las condiciones de la regla anterior: Esta (robot, ?sitio1) y Hay-Puerta (?sitio1, C). Ambas condiciones se satisfacen en la base de hechos si hacemos la sustitución ?sitio1=B. Así pues, por medio del encadenamiento hacia atrás hemos llegado a la conclusión de que el robot puede coger la pistola. Todo el proceso de encadenamiento se puede observar con más claridad en la figura 6.

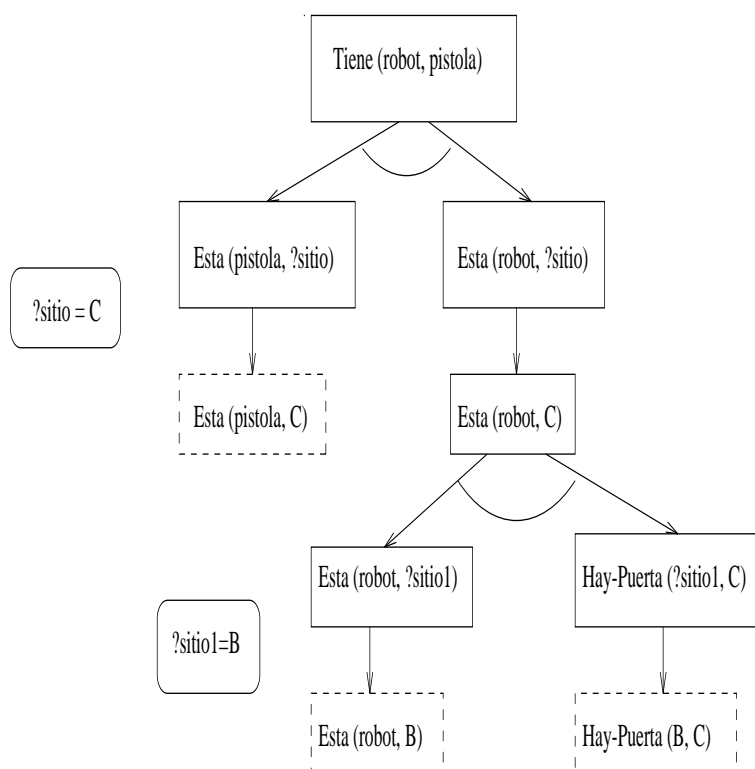


Figure 6: Arbol de búsqueda para el encadenamiento hacia atrás

Hay que tener en cuenta que en el caso anterior siempre hemos elegido las sustituciones de variables adecuadas y hemos ordenado las metas a resolver de la manera más apropiada posible. Esa manera de ordenar las metas no es conocida por el motor de inferencia a priori, por lo que habría que dotar al motor de inferencia de estrategias de control inteligentes o con mecanismos de vuelta atrás (al igual que hacíamos con el encadenamiento hacia adelante).

Resumiendo, el encadenamiento hacia atrás consiste en partir de un objetivo principal y obtener que objetivos parciales previos se deben satisfacer para que se pueda cumplir nuestro objetivo principal. Por supuesto, si un objetivo parcial aparece en nuestra base de hechos, ese objetivo queda satisfecho automáticamente. En ocasiones se pueden combinar el encadenamiento hacia atrás con el encadenamiento hacia adelante, con lo que en la base de hechos tendríamos tanto hechos como metas. A esos sistemas de producción se les denomina bidireccionales.

Cada uno de los dos modos de encadenamiento tiene sus ventajas e inconvenientes. Por ejemplo, si tenemos muchas posibles metas (u objetivos) será mejor utilizar un sistema con encadenamiento hacia adelante. Por otro lado, es mucho más sencillo seguir un sistema de producción con encadenamiento hacia atrás que uno hacia adelante. Si queremos comprender cómo el sistema de producción ha alcanzado una conclusión determinada, deberemos usar encadenamiento hacia atrás.

En otras ocasiones sería ventajoso usar encadenamiento bidireccional. La idea básica es acercar la meta al encadenamiento hacia adelante (por medio del encadenamiento hacia atrás) y acercar los hechos al encadenamiento hacia atrás (por medio del encadenamiento hacia adelante) con el objetivo de que ambos se crucen en algún punto. Pero podría ocurrir que ambos encadenamientos jamás llegaran a cruzarse con lo que el sistema sería doblemente ineficiente.

5.2 Ventajas y desventajas de los sistemas de producción

Entre las ventajas de un sistema de producción se pueden citar:

- Idealmente cada regla es independiente de las demás, por lo que resulta sencillo modificarlas, borrarlas o introducirlas sin afectar al resto del sistema. Desafortunadamente no siempre es posible mantener esta independencia.
- Parecen un buen modelo de como los seres humanos representan y resuelven problemas.
- Es bastante natural expresar el conocimiento humano en forma de reglas.
- Es bastante sencillo incrementar el conocimiento de un sistema de producción añadiéndole nuevas reglas.
- Obligan a representar el conocimiento de una forma uniforme.
- Son mucho más flexibles que un programa de ordenador normal. En los programas de ordenador, la secuencia que han de seguir las instrucciones está decidida desde el principio. Sin embargo, en un sistema de producción las reglas se disparan según el contenido de la base de hechos, por lo que la reacción del sistema se adapta al momento.
- Funcionan bien en entornos reales, como se ha comprobado repetidamente en los sistemas expertos (que son básicamente sistemas de producción).

Pero también tienen desventajas:

- Debido a que los sistemas de producción reaccionan en todo momento al contenido de la base de hechos, es muy difícil imponer que las reglas se disparen en determinada secuencia (cosa que en ciertas ocasiones sería deseable).
- Resulta difícil seguir el comportamiento de un sistema de producción. Aunque es muy sencillo saber qué es lo que cada regla está haciendo en cada momento, es muy difícil determinar que es lo que el conjunto de reglas está haciendo globalmente.

6 Marcos

Un modo de representación que ayuda a resolver los problemas encontrados en la representación mediante lógica de predicados es la representación basada en **marcos** (**frames**, en inglés).

Ideados por Marvin Minsky en la década de los setenta, en relación con sistemas de visión por computador, ofrecen una estructura mucho más compacta que la lógica de predicados para representar conceptos y relaciones entre conceptos. En la mayoría de los casos, lo expresable en lógica de predicados es también expresable en forma de marcos, y viceversa.

Lo que hace que la preferencia del representador se incline hacia una u otra opción son criterios como modularidad, encapsulamiento de la información, concisión, facilidad de uso, representación intuitiva, eficacia y facilidad de recuperación de información, posibilidades de implementación en computador, rendimiento, etc.

Por otra parte, nunca está de más recordar que los marcos, nacidos en el mundo de la Inteligencia Artificial, son el origen de la orientación a objetos. De hecho, marco y objeto son prácticamente sinónimos.

6.1 Definición y propiedades de los marcos

Un marco es una colección de atributos (o **slots**, en inglés) que poseen determinados valores, y que describen una entidad del mundo. Los marcos se agrupan en sistemas de marcos, relacionados unos con otros, modelizando así las relaciones que existen entre las entidades del mundo representado.

Por ejemplo, un marco que represente la idea de POLICIA podría ser el siguiente, que además contiene información sobre que tipo de arma utilizan los policías:

POLICIA
ES-UN: PERSONA
Sexo:
Edad:
Ciudad:
Distrito:
Pistola: Colt-45

Compárese con una fórmula equivalente del cálculo de predicados, que expresa la misma idea:

$$\boxed{\forall x, \text{IF Policia}(x) \text{ THEN usa-arma}(x, \text{Colt-45})}$$

Al igual que con el cálculo de predicados, una representación del mismo dominio puede realizarse de varias formas diferentes. Depende de cada caso concreto el elegir entre una forma u otra de representar los mismos contenidos.

Obsérvese que, según la definición dada, los marcos reflejan un conocimiento estático del dominio, es decir, modelizan la situación del mundo en un momento dado. Son como una fotografía de un instante de la realidad. Pero esto no hay que tomarlo en sentido estricto: se pueden definir una serie de reglas y procesos que modifiquen los valores del sistema de marcos en el estado actual, pasando así a reflejar un nuevo estado del mundo.

Como ejemplo, si un cierto día se decide que los policías deben cambiar su arma de un Colt-45 a un Colt-90, entonces se cambiaría el valor del slot 'Pistola' en el marco POLICIA para reflejar

esta nueva situación o estado. Incluso a veces podría ser necesario crear nuevos marcos o nuevos slots, así como borrar algunos ya existentes.

De todas las relaciones existentes entre entidades del mundo, son especialmente relevantes para los sistemas de marcos las relaciones que expresan la pertenencia de un elemento a un conjunto (relación INSTANCIA-DE) y las que expresan la relación de subconjunto (relación ES-UN). Esto conduce a una **organización jerárquica** de los marcos, según clases, subclasses y elementos.

Así, por ejemplo, se tiene que CAP-MC-CLUSKEY es un elemento del conjunto POLICIA, y POLICIA es un subconjunto del conjunto más general PERSONA. Esto se representaría mediante el siguiente sistema de marcos:

PERSONA
ES-UN: MAMIFERO
Sexo:
Edad:
Ciudad:

POLICIA
ES-UN: PERSONA
Sexo:*
Edad:*
Ciudad:*
Distrito:
Pistola: Colt-45

CAP-MC-CLUSKEY
INSTANCIA-DE: POLICIA
Sexo:* V
Edad:* 55
Ciudad:* New-York
Distrito:* 23
Pistola:* Colt-45

En forma gráfica, la jerarquía de marcos es la de la figura 7.

Basándonos en este ejemplo y extendiéndolo, pasaremos a explicar las ideas fundamentales que subyacen a la representación del conocimiento mediante marcos.

6.1.1 Herencia de propiedades

En una jerarquía de marcos, como en el ejemplo expuesto, queda claro que las propiedades o slots de los marcos situados más arriba, y que corresponden por tanto a los conceptos más generales, se transmiten a los marcos que representan clases de objetos situadas más abajo en la jerarquía, así como a las instancias de dichas clases. Dicha transmisión de propiedades de un conjunto a sus subconjuntos, se denomina **herencia**. En el ejemplo visto, los slots heredados se han marcado con *.



Figure 7: Ejemplo de jerarquía de marcos

Así, por ejemplo, al ser POLICIA un subconjunto de la clase PERSONA, heredará las propiedades de Sexo y Edad. Igualmente, la instancia CAP-MC-CLUSKEY, hereda los slots correspondientes a POLICIA, y por tanto, también los de PERSONA. Esto permite realizar inferencias sobre los conceptos representados. Por ejemplo, sabiendo que CAP-MC-CLUSKEY es policía, subiendo por la jerarquía de marcos, se deduce que su arma es un Colt-45. Pero no se podría asegurar nada sobre su ciudad o distrito, a menos que se disponga de información adicional.

En este momento, debería quedar claro que esto es una forma de representar el hecho "Todos los policías tienen un Colt-45", pero de forma más concisa y más clara que la representación equivalente en lógica de predicados. Además, la expresividad de los marcos permite tratar casos excepcionales, como se verá en el siguiente apartado.

6.1.2 Valores por omisión y excepciones

Apoyándose en el mecanismo de herencia, se pueden proporcionar valores por omisión de los atributos que se desee. Un ejemplo es el que se ha comentado, respecto al valor del slot 'Arma' en el marco POLICIA. Pero, habitualmente, en muchos dominios de la vida real aparecen excepciones. Sobradamente conocido es el caso de que todas las aves pueden volar, pero no los pingüinos, o que ningún mamífero puede volar, salvo el murciélago.

En el ejemplo de los policías, se tiene que todos ellos utilizan como arma un Colt-45, pero puede haber excepciones como la siguiente:

HARRY-EL-SUCIO
INSTANCIA-DE: POLICIA
Sexo: V
Edad: 45
Ciudad: San-Francisco
Distrito: 42
Pistola: Magnum-70

Los sistemas de marcos no tienen ningun problema en tratar correctamente estos casos. Ante una consulta del tipo ¿Que arma utiliza HARRY-EL-SUCIO? primero se extrae el valor almacenado en el slot 'Arma' de HARRY-EL-SUCIO, devolviendo 'Magnum-70' . En el supuesto de que no hubiese ningun valor, al ascender por la jerarquía de marcos se encuentra que en el marco correspondiente a POLICIA el valor por omisión de 'Arma' es 'Colt-45', y sería éste el valor devuelto. Siempre se recuperan los valores de abajo-arriba a traves de la jerarquía.

6.1.3 Expresión de conjunciones, disyunciones y negaciones

Un paso más en la expresividad de los marcos se da cuando se trata de representar disyunciones y conjunciones de hechos. Por ejemplo, la proposición “Los policias tienen un Colt-45 o un Magnum-70” se expresaría de la siguiente forma:

POLICIA
ES-UN: PERSONA
Sexo:
Edad:
Ciudad:
Distrito:
Pistola: ({Colt-45} OR {Magnum-70})

Análogamente, podría expresarse "Harry el sucio trabaja en los distritos 23, 35 y 42" así:

HARRY-EL-SUCIO
INSTANCIA-DE: POLICIA
Sexo: V
Edad: 45
Ciudad: San-Francisco
Distrito: ({23} AND {35} AND {42})
Pistola: Magnum-70

Si además se dispone de la negación, podrían expresarse en un solo marco proposiciones complejas como "Harry el sucio no tiene un Colt-45, pero tiene una Magnum-70 o una Magnum-50":

HARRY-EL-SUCIO
INSTANCIA-DE: POLICIA
Sexo: V
Edad: 45
Ciudad: San-Francisco
Distrito: 42
Pistola: (NOT ({Colt-45}) AND ({Magnum-70} OR {Magnum-50}))

6.1.4 Referencias a otros marcos

No sólo se pueden relacionar unos marcos con otros a través de las relaciones ES-UN e INSTANCIA-DE. En muchos casos, los valores de slot de un marco pueden apuntar a otro marco, para expresar un hecho que implica a ambos.

Según esto, si en el mundo que se pretende representar son relevantes las ciudades y sus características, es posible que se disponga del marco CIUDAD, una de cuyas instancias podría ser el marco SAN-FRANCISCO. Entonces podría expresarse así el hecho de que Harry el Sucio viva en la ciudad San Francisco:

SAN-FRANCISCO
INSTANCIA-DE: CIUDAD
Alcalde: ...
Habs: ...
...

HARRY-EL-SUCIO
INSTANCIA-DE: POLICIA
Sexo: V
Edad: 45
Ciudad: SAN-FRANCISCO
Distrito: 42
Pistola: Magnum-70

Obsérvese que esto sería útil para el caso de tener que responder a preguntas del tipo ¿Cuántos habitantes tiene la ciudad en la que habita Harry el Sucio?.

6.1.5 Expresiones funcionales

Aparte de las importantes relaciones de INSTANCIA-DE y ES-UN, que referencian a un marco desde otro, puede haber otro tipo de expresiones que relacionen distintos marcos entre sí, a través de un valor de slot. Como ejemplo, esta es la representación de la proposición "Mc Cluskey trabaja en la misma ciudad y distrito que Harry el sucio", que relaciona ambos marcos:

CAP-MC-CLUSKEY
INSTANCIA-DE: POLICIA
Sexo: V
Edad: 55
Ciudad: Ciudad(HARRY-EL-SUCIO)
Distrito: Distrito(HARRY-EL-SUCIO)
Pistola: Colt-45

La riqueza y concisión de la representación mediante marcos es evidente. Posteriormente se expondrá un ejemplo de mayor complejidad que éste y en el que intervendrán un mayor número de marcos.

6.1.6 Marcos temáticos

No sólo se pueden expresar propiedades de individuos o de objetos. También es importante la representación del conocimiento implicado en las acciones, así como los objetos implicados en ella. Tales marcos, que reflejan sucesos del mundo, se denominan marcos temáticos, e incluyen slots que representan al agente de la acción, al receptor de la acción, los instrumentos, etc. Véase, como ejemplo, el siguiente marco temático:

ASESINATO
ES-UN: CRIMEN
Condena: 15
Lugar:
Hora:
Autor:
Victima:
Arma:

Este marco refleja un asesinato genérico. Un caso concreto de asesinato, como el que ocurre en "El Padrino", debería ser una instancia de la clase ASESINATO, como la siguiente:

ASESINATO-1
INSTANCIA-DE: ASESINATO
Condena: 15
Lugar: Restaurante-Fredo's
Hora: 22:30
Autor: Michael-Corleone
Victima: CAP-MC-CLUSKEY
Arma: Colt-45

6.1.7 Demonios

Es preciso, a veces, indicar un modo de obtención de los valores de los slots por medio de un procedimiento. Tales procedimientos se llaman demonios (**daemons**, en inglés). Un proceso

demonio es aquel que se encuentra a la espera de que ocurra un determinado evento, y entonces se ejecuta.

Hay dos tipos principales de procedimientos demonio: aquellos que se ejecutan cuando es necesario llenar un slot, y aquellos que se ejecutan cuando ya se ha llenado un slot. Los primeros se especifican en el marco con la etiqueta PARA-LLENAR y los segundos, con la etiqueta CUANDO-LLENO.

Por ejemplo, si se quiere expresar en un slot del marco ASESINATO la idea de que los años de condena para el culpable son 15 multiplicado por el número de veces que el autor cometió un crimen parecido en el pasado (antecedentes penales), se emplearía lo siguiente:

ASESINATO
ES-UN: CRIMEN
Condena: PARA-LLENAR: Daemon-Calculo-de-pena
Lugar:
Hora:
Autor:
Victima:
Arma:
Daemon-Calculo-de-pena = 15 + Antecedentes(Autor(CRIMEN))

Como ejemplo de demonio del segundo tipo, se presenta una ampliación del anterior marco, que refleja el hecho de que cuando se comete un asesinato, al autor se le debe incrementar en uno el número de antecedentes:

ASESINATO
ES-UN: CRIMEN
Condena: PARA-LLENAR: Daemon-Calculo-de-pena
Lugar:
Hora:
Autor: CUANDO-LLENO: Daemon-Incr-antecedentes
Victima:
Arma:
Daemon-Incr-antecedentes = Registrar(Antecedentes(Autor(ASESINATO)) + 1)

6.2 Equiparación (“pattern-matching”)

Una vez representado un dominio del mundo real por medio de un sistema de marcos, la utilidad de tal sistema deriva de la posibilidad de que se pueda emplear para contestar preguntas y derivar nuevos hechos, y así prestar ayuda en la solución de problemas relacionados con dicho dominio. Para ello se utiliza la confrontación de patrones o **equiparación**, que es un proceso análogo a la unificación en cálculo de predicados.

El proceso es el siguiente: El sistema de marcos que refleja el conocimiento acerca de un dominio se denomina conjunto de MARCOS-HECHO. Si se quiere extraer una respuesta a una pregunta, se construirá un MARCO-OBJETIVO, que represente la pregunta formulada. Este marco-objetivo se compara con los marcos-hechos, hasta encontrar uno que coincida con él slot por slot, teniendo

en cuenta además el conocimiento heredado. Una vez encontrado, este marco-hecho proporciona la información deseada.

Por ejemplo: Con la jerarquía de marcos vista anteriormente, se quiere conocer cual es la edad del policía de San Francisco. Esta pregunta se puede expresar mediante el marco-objetivo:

OBJETIVO
INSTANCIA-DE: POLICIA
Ciudad: San-Francisco
Edad: ?

No importa en que orden se comparen los slots. Lo relevante es que coincidan todos los del marco-objetivo con un subconjunto de los slots de algún marco-hecho. De acuerdo con esto, el marco-objetivo confronta con el marco-hecho siguiente:

HARRY-EL-SUCIO
INSTANCIA-DE: POLICIA
Sexo: V
Edad: 45
Ciudad: San-Francisco
Distrito: 42
Pistola: Magnum-70

Por tanto, se obtiene la respuesta de que la edad buscada es 45 años. Obsérvese que se podría haber preguntado ¿Cuál es la edad del policía de San Francisco que trabaja en el distrito 21? Esta pregunta corresponde al marco-objetivo:

OBJETIVO
INSTANCIA-DE: POLICIA
Ciudad: San-Francisco
Distrito: 21
Edad: ?

En este caso, no se produce ninguna confrontación válida dentro del conjunto de marcos-hechos de los que se dispone. La respuesta sería un mensaje de error, informando que no se conoce ningún policía que cumpla con las características dadas.

En aquellos casos en los que un slot de un marco referencia a otro marco, bien directamente o a través de expresiones funcionales, el proceso de confrontación se encuentra con complejidades adicionales, debido a la "navegación" necesaria a través del sistema de marcos.

Por ejemplo, el sistema dispone, además de los marcos-hechos anteriores, del siguiente marco-hecho:

CAP-MC-CLUSKEY
INSTANCIA-DE: POLICIA
Sexo: V
Edad: 55
Ciudad: Ciudad(HARRY-EL-SUCIO)
Distrito: 23
Pistola: Colt-45

Ahora se trata de contestar a la pregunta ¿Cual es la ciudad en la que habita el policia de 55 años?. Esta pregunta se traslada a su representación en la forma de marco-objetivo siguiente:

OBJETIVO
INSTANCIA-DE: POLICIA
Edad: 55
Ciudad: ?

El proceso de equiparación encuentra una coincidencia cuando equipara el anterior marco-objetivo con el marco-hecho correspondiente a CAP-MC-CLUSKEY. La respuesta a la pregunta, en una primera aproximación, sería:

Ciudad: Ciudad(HARRY-EL-SUCIO)

Pero esta respuesta no puede satisfacer a nadie. Por tanto, el proceso de equiparación debe acceder al marco-hecho correspondiente a HARRY-EL-SUCIO y extraer el valor del slot correspondiente a 'Ciudad'. Se obtiene así la respuesta correcta:

Ciudad: San-Francisco

Se pueden imaginar niveles adicionales de complejidad. Por ejemplo, antes se propuso un problema en el que SAN-FRANCISCO era un marco instancia del marco CIUDAD. En una situación así podría preguntarse ¿Cual es el numero de habitantes de la ciudad en la que habita el policia de 55 años?. Esto requeriría un acceso adicional al slot 'Habs' del marco SAN-FRANCISCO, con el consiguiente coste.

Otras cuestiones que plantean complicaciones son las del tipo ¿Qué policías trabajan en San-Francisco? o incluso ¿Qué personas habitan en San Francisco? Habría que acceder a todos y cada uno de los marcos-hecho representando a personas, incluyendo a todas las subclases de persona, y ver que valor contiene el slot 'ciudad'.

En sistemas reales, el número de marcos y relaciones entre ellos es infinitamente mayor que en este ejemplo, y las preguntas que un usuario formula a tales sistemas son también mucho más complejas. Puede ser necesario organizar el conocimiento de alguna otra forma, para facilitar estos procesos, que de otra forma serían inabordables. Esta idea se desarrollará posteriormente, cuando se introduzcan las redes semánticas.

6.3 Ejemplo de sistema de marcos

Se trata de representar en este sistema el dominio del ejemplo, basado en los personajes y circunstancias de la película "El Padrino". Evidentemente, para realizar aplicaciones que resuelvan problemas del mundo real, tales como el diagnóstico médico, o la detección de posibles causas de problemas en una planta industrial, los conceptos e ideas que aquí se tratan son perfectamente extrapolables.

Se manejará la jerarquía de marcos representada en la figura 8.

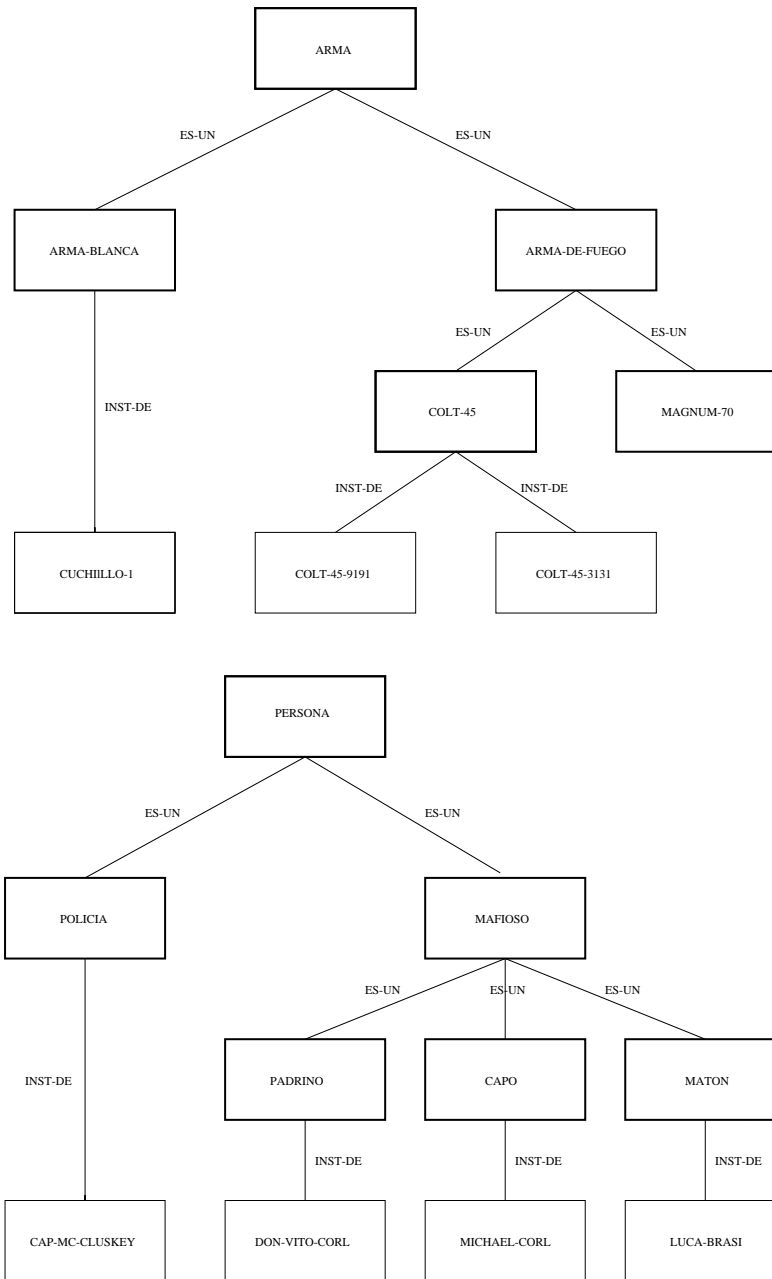


Figure 8: Jerarquía de marcos

Supóngase que, antes de nada, se necesita representar conocimiento acerca de las armas que utilizan los personajes de la película. Para ello se dispondrá del marco ARMA, del cual descenderán los marcos ARMA-BLANCA y ARMA-DE-FUEGO:

ARMA
ES-UN: OBJETO
Precio:
Alcance:

ARMA-BLANCA
ES-UN: ARMA
Precio:
Alcance: Corto
Longitud:

ARMA-DE-FUEGO
ES-UN: ARMA
Precio:
Alcance: Largo
Calibre:
Disparos:
Numero-serie:

COLT-45
ES-UN: ARMA-DE-FUEGO
Precio: 1500
Alcance: Largo
Calibre: 45
Disparos: 6
Numero-serie:

MAGNUM-70
ES-UN: ARMA-DE-FUEGO
Precio: 9000
Alcance: Muy-Largo
Calibre: 70
Disparos: 10
Numero-serie:

Se necesitarán armas concretas, que seran la que utiliza Michael Corleone y la que utiliza el capita McCluskey, así como el cuchillo de Luca Brasi. Por tanto, se instancian de la siguiente forma:

COLT-45-9191
INSTANCIA-DE: COLT-45
Precio: 1500
Alcance: Largo
Calibre: 45
Disparos: 6
Numero-serie: 9191

COLT-45-3131
INSTANCIA-DE: COLT-45
Precio: 1500
Alcance: Largo
Calibre: 45
Disparos: 6
Numero-serie: 3131

CUCHILLO-1
INSTANCIA-DE: ARMA-BLANCA
Precio: 2000
Alcance: Corto
Longitud: 30

A continuación, y siguiendo un orden de arriba a abajo a traves de la jerarquía de clases, se dispone del marco PERSONA, visto anteriormente:

PERSONA
ES-UN: MAMIFERO
Sexo:
Edad:
Ciudad:

Evidentemente, podría mostrarse tambien el marco MAMIFERO, y por encima de éste, el marco ANIMAL-VERTEBRADO, etc. Pero se considera que no es necesario llegar a ese nivel para modelizar el ejemplo tratado. En otras palabras, las propiedades de los mamíferos o de los animales vertebrados no se consideran relevantes para tratarlas en este dominio concreto.

Posteriormente, es necesario representar qué clases de personas del dominio son las que interesa reflejar en el sistema. De entrada, parece claro que hay, por un lado, policías, y por el otro, mafiosos:

POLICIA
ES-UN: PERSONA
Sexo:
Edad:
Ciudad:
Distrito:
Pistola: COLT-45

MAFIOSO
ES-UN: PERSONA
Sexo:
Edad:
Ciudad:
Familia:
Rango:

En principio, dentro de los mafiosos se distinguirán tres grandes grupos, los matones, los capos y los padrinos, que tendrán distintos rangos y distintas propiedades, según se ve a continuación:

PADRINO
ES-UN: MAFIOSO
Sexo: V
Edad:
Ciudad:
Familia:
Rango: 1
Padre-de:
Paga-a:

CAPO
ES-UN: MAFIOSO
Sexo:
Edad:
Ciudad:
Familia:
Rango: 2
Territorio:
Arma: COLT-45

MATON
ES-UN: MAFIOSO
Sexo:
Edad:
Ciudad:
Familia:
Rango: 3
Arma:

En este momento, se podrían representar algunos de los personajes de la película, por medio de instancias de la clase correspondiente. Así, se tiene:

CAP-MC-CLUSKEY
INSTANCIA-DE: POLICIA
Sexo: V
Edad: 55
Ciudad: New-York
Distrito: 23
Pistola: COLT-45-3131

LUCA-BRASI
INSTANCIA-DE: MATON
Sexo: V
Edad: 40
Ciudad: Chicago
Familia: Corleone
Rango: 3
Arma: CUCHILLO-1

MICHAEL-CORLEONE
INSTANCIA-DE: CAPO
Sexo: V
Edad: 33
Ciudad: New-York
Familia: Corleone
Rango: 2
Territorio: Brooklyn
Arma: COLT-45-9191

DON-VITO-CORLEONE
INSTANCIA-DE: PADRINO
Sexo: V
Edad: 70
Ciudad: New-York
Familia: Corleone
Rango: 1
Padre-de: MICHAEL-CORLEONE
Paga-a: {MC-CLUSKEY} AND {LUCA-BRASI}

En un momento de la película, Michael Corleone mata al Capitán McCluskey con un Colt-45. Este tipo de hechos también pueden expresarse perfectamente con un modelo de representación basado en marcos. Se tiene un asesinato, que es una subclase de crimen, y que implica a dos personas, así como a un arma homicida. Por tanto:

CRIMEN
ES-UN: ACTO
Condena:
Lugar:
Fecha:
Hora:
Autor:

ASESINATO
ES-UN: CRIMEN
Condena: 15
Lugar:
Fecha:
Hora:
Autor:
Victima:
Arma:

ASESINATO-1
INSTANCIA-DE: ASESINATO
Condena: 15
Lugar: Restaurante-Fredo's
Fecha: 15/6/54
Hora: 22:30
Autor: MICHAEL-CORLEONE
Victima: CAP-MC-CLUSKEY
Arma: COLT-45-9191

6.4 Ejemplo de utilización de un sistema de marcos

La obtención de respuestas a preguntas por medio de la equiparación es la principal de las tareas que se pueden realizar sobre los marcos. Otra tarea importante sería la modificación de los valores expresados en el sistema de marcos con el objeto de reflejar cambios de estado en el mundo.

Como ejemplo de equiparación, se mostrará el proceso de obtención de respuesta a la pregunta ¿Cual era el calibre del arma de fuego utilizada en el asesinato cometido en el Restaurante Fredo's?

En primer lugar, esta pregunta debe representarse en un modo que permita realizar la equiparación. Expresado como marcos-objetivo sería:

OBJETIVO-1
INSTANCIA-DE: ASESINATO
Lugar: Restaurante-Fredo's
Arma: OBJETIVO-2

OBJETIVO-2
INSTANCIA-DE: ARMA-DE-FUEGO
Calibre: ?

Primero se resuelve el OBJETIVO-1. Se busca una posible equiparación con algún marco-hecho, resultando que ASESINATO-1 es el marco buscado. En dicho marco el slot 'Arma' tiene el valor COLT-45-9191.

El siguiente paso es verificar que el marco-objetivo OBJETIVO-2 es equiparable con el marco-hecho correspondiente a COLT-45-9191. El primero es una instancia de un ARMA-DE-FUEGO, y el segundo es una instancia de COLT-45. Por tanto, el mecanismo de equiparación debe resolver esta aparente discrepancia recurriendo a la herencia, encontrándose que, efectivamente, un COLT-45 es un subconjunto de ARMA-DE-FUEGO. Por tanto, la equiparación tiene éxito y se obtiene el valor de 'Calibre' de COLT-45-9191, que es la respuesta a la pregunta inicialmente formulada:

$$\text{Calibre}(\text{COLT-45-9191}) = 45$$

Un ejemplo sencillo de modificación de los valores del sistema de marcos para reflejar cambios en el estado del mundo, podría ser el siguiente: Supóngase que, una vez muerto DON-VITO-CORLEONE, para elegir un sucesor se decide que MICHAEL-CORLEONE será el proximo PADRINO de la familia Corleone. Para relejar esto en el sistema, se producirían los siguientes cambios:

1. Borrado del marco DON-VITO-CORLEONE.
2. Cambio en el slot 'INSTANCIA-DE' de MICHAEL-CORLEONE. Pasa de contener el valor CAPO a tener el valor PADRINO. Como consecuencia de esto, automáticamente el slot 'Rango' deja de tener valor 2 y pasa a tener valor 1.

Gráficamente, este cambio de situación situación podría expresarse como en la figura 9.

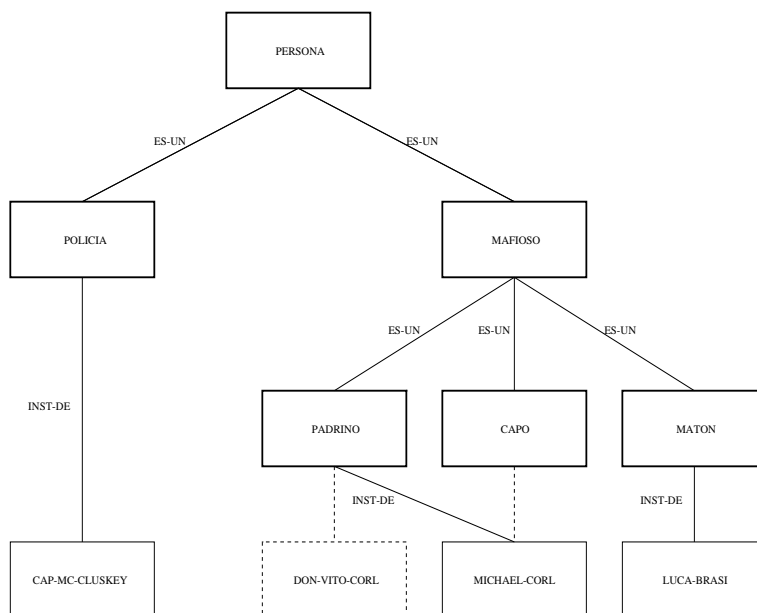


Figure 9: Ejemplo de cambio de situación

7 Redes semánticas

En la representación del conocimiento por medio de **redes semánticas**, los conceptos se representan como un conjunto de **nodos** conectados unos a otros por medio de un conjunto de **arcos** etiquetados, que expresan las relaciones existentes entre los nodos.

La representación del conocimiento por medio de sistemas de marcos puede conducir directamente al formalismo gráfico de redes semánticas: Se debe convertir cada nombre de marco en un nodo, cada valor de slot en otro nodo y el arco que los une irá etiquetado con el nombre del slot.

Por ejemplo, sea el siguiente sistema de marcos:

ARMA-DE-FUEGO
ES-UN: ARMA
Precio:
Alcance:

COLT-45
ES-UN: ARMA-DE-FUEGO
Precio: 1500
Alcance: Largo

MAGNUM-70
ES-UN: ARMA-DE-FUEGO
Precio: 9000
Alcance: Muy-Largo

COLT-45-9191
INSTANCIA-DE: COLT-45
Precio: 1500
Alcance: Largo

Su representación como red semántica puede verse en la figura 10.

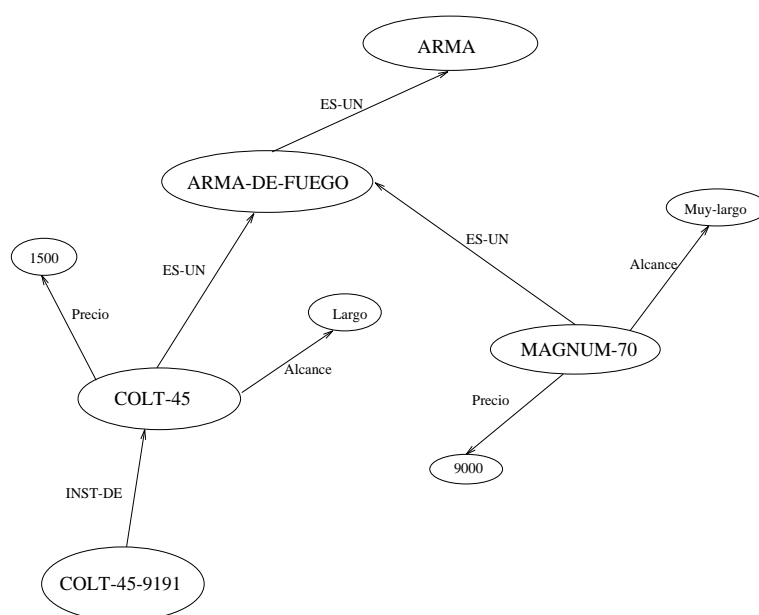


Figure 10: Ejemplo de Red Semántica

Esta pequeña red semántica es útil para realizar inferencias con el objeto de obtener respuestas a preguntas como ¿Cuál es el precio de una Magnum-70? ¿Cuál es su alcance?.

Pero es importante destacar que los apuntadores pueden recorrerse también en sentido inverso, aunque en el gráfico aparezcan ser unidireccionales. Esto permite responder fácil y rápidamente a preguntas como ¿Cuál es el arma de fuego que vale 9000\$? ¿Qué tipos de armas de fuego se conocen?. En este sentido pueden ser más eficientes que los sistemas de marcos, aunque estos últimos presenten una mayor modularidad y encapsulamiento de la información.

Puede constatarse que en la representación por medio de redes semánticas, aparte de las ventajas que derivan de su carácter gráfico, es evidente que la representación de las referencias entre marcos, expuestas en el tema anterior, aquí cobran un carácter más expresivo e intuitivo, tanto si dichas relaciones son de carácter hereditario como si son de otro tipo.

Las redes semánticas, así definidas, ofrecen un nuevo modo de representación del conocimiento. Y de nuevo se observa el hecho de que lo representable mediante una red semántica es representable mediante marcos y mediante la lógica de predicados. Todos estos métodos basan la representación del conocimiento en las categorías de Objetos, Atributos y Valores, fundamentalmente.

La elección de una representación frente a otra depende, por tanto, de cada caso concreto, intentando siempre elegir la que pueda ser más eficaz y sencilla para cada aplicación particular.

7.1 Herencia en redes semánticas

En la representación basada en redes semánticas existe también el mecanismo de herencia, a través de los apuntadores etiquetados con ES-UN e INST-DE. Como se ve en el ejemplo, la instancia COLT-45-9191 hereda las propiedades de COLT-45 y por eso no es necesario que figuren explícitamente en el dibujo.

7.2 Expresión de conjunciones, disyunciones y negaciones

La representación en redes semánticas no se encuentra en desventaja, respecto a los sistemas ya vistos, en cuanto a la representación de las conectivas proposicionales OR y NOT.

Para ello se utilizan líneas cerradas discontinuas, como en el ejemplo de la figura 11, que corresponde al hecho "Una Magnum-70 tiene un alcance largo o muy largo y su precio no es 1500 \$":

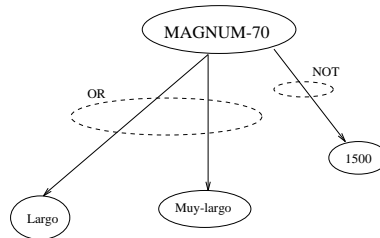


Figure 11: Conjunciones y Disyunciones

7.3 El mecanismo de equiparación en redes semánticas

Este mecanismo, que es el que permite realizar inferencias sobre la red, es análogo al mecanismo de la unificación en lógica de predicados y a la equiparación en sistemas de marcos.

En general, se tendrá un objeto meta que se quiere equiparar a uno de los objetos-hecho que constituyen la red semántica. La forma de actuar es la siguiente:

Se construye un fragmento de red, que representa un objeto o pregunta a realizar, y se coteja con la red de hechos, para ver si tal objeto existe. Los nodos que en la pregunta tengan valor desconocido se ligan en el proceso de equiparación a los valores que debían tener para que la equiparación fuera perfecta. Estos valores proporcionan así una respuesta a la pregunta formulada. Es importante que el mecanismo de equiparación trate correctamente los valores heredados.

Como ejemplo, sea la red de hechos de la figura 12, basada en el sistema de marcos anteriormente expuesto :

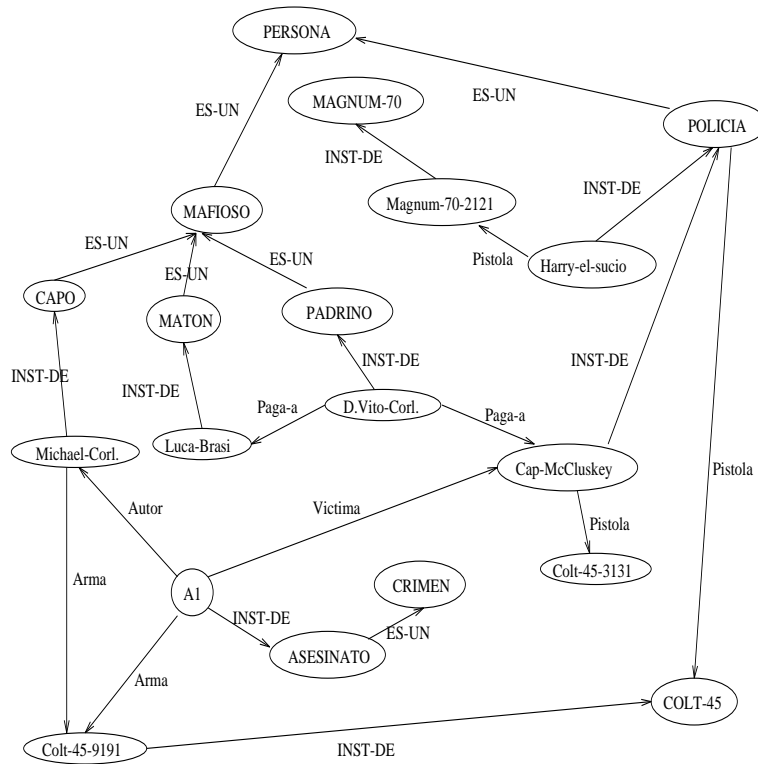


Figure 12: Red de Hechos

Se desea saber si hay algún policía corrupto en la ciudad, entendiendo por tal a aquel policía que es pagado por un mafioso. Es decir, se necesita encontrar valores que se equiparen con X? en la red-pregunta representada en la figura 13.

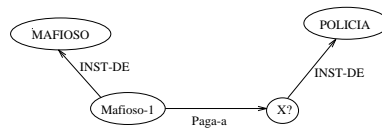


Figure 13: Red-Pregunta

En la figura 14, la subred de hechos equiparada con la red-pregunta se muestra mediante una línea de puntos.

El proceso de equiparación puede empezar por unificar el nodo ?X con una instancia de policía concreto, y ver si algún arco 'PAGA-A' incide en él. El nodo correspondiente al otro extremo de dicho arco es necesario que corresponda a una instancia de MAFIOSO. Se precisa del mecanismo de herencia para resolver esto, pues a través de PADRINO llega a MAFIOSO mediante el arco ES-UN. Se obtiene así la única respuesta posible, que unifica a X? con Cap-Mc-Cluskey.

Otra pregunta posible sería: ¿A quien paga Don Vito Corleone?. Se trata de obtener valores que se unifiquen con el nodo etiquetado con X? en la red semántica que traduce dicha pregunta, que aparece representada en la figura 15. En este caso, hay dos posibilidades válidas para X?. Las personas pagadas por Don Vito Corleone son Luca-Brasi y Cap-Mc-Cluskey, como se puede ver en la figura 16.

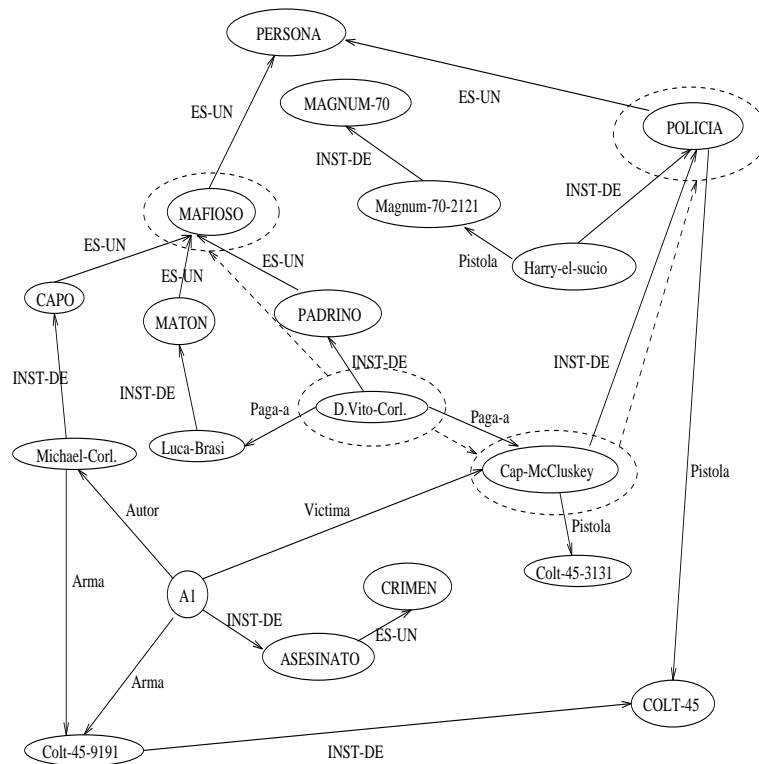


Figure 14: Equiparación de Red-Hecho y Red-Pregunta

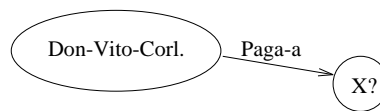


Figure 15: Red-Pregunta

8 Otros sistemas de representación

En esta sección se introducirán brevemente métodos de representación alternativos y, a veces, complementarios, de los sistemas vistos. Cada uno de ellos enfatiza en mayor o menor grado ciertos aspectos del conocimiento que se consideran relevantes para el problema considerado. Así, por ejemplo, los Scripts muestran la secuencia de los eventos a lo largo del tiempo, y los sistemas de razonamiento bajo incertidumbre tienen en cuenta valores numéricos que indican el grado de evidencia de una proposición.

8.1 Guiones (“scripts”)

Un **script** o guión es una estructura que describe una secuencia fija de los eventos que ocurren en un determinado contexto. Tiene una estructura muy similar a la de los marcos, pero ampliada para reflejar la sucesión de hechos en el tiempo, así como los cambios que tales hechos producen. La mejor forma de verlo es a través de un ejemplo. Se construirá el guión RESTAURANTE, referente al hecho de ir a cenar a un restaurante.

Todo guión consta, principalmente, de las siguientes partes:

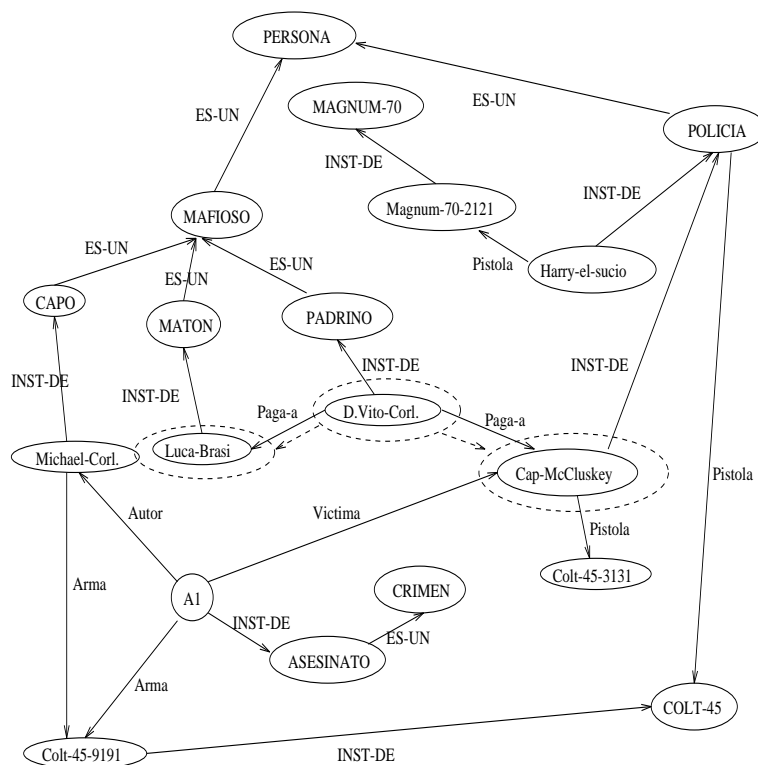


Figure 16: Ejemplo de Equiparación

- **CONDICIONES-DE-ENTRADA:** Son una serie de condiciones que deben ser verdaderas para poder ejecutarse el guión.
- **RESULTADOS:** Condiciones que obligatoriamente serán ciertas una vez que se realicen los eventos del guión.
- **OBJS:** Es un conjunto de slots que representan los objetos involucrados en los eventos del guión.
- **ROLES:** Conjunto de slots que representan a los agentes (personas, robots, etc) que intervienen en el guión.
- **ESCENAS:** Son los eventos que realmente ocurren, dispuestos secuencialmente. Pueden representarse mediante algún formalismo (como el de dependencias conceptuales), pero se utilizará el lenguaje natural.

En definitiva, este es el script del restaurante:

GUIÓN: RESTAURANTE
CONDICIONES-DE-ENTRADA: tiene-hambre(C) AND tiene-dinero(C)
RESULTADOS: NO tiene-hambre(C) AND tiene-menos-dinero(C) AND tiene-mas-dinero(P)
OBJS: Mesas, Menu, Comida, Cuenta, Dinero, Caja.
ROLES: C=Cliente, CA=Camarero, P=Propietario
ESCENA-1: ENTRAR C entra en el restaurante, busca una mesa libre. Si la encuentra, se sienta. Si no, se va.
ESCENA-2: PEDIR CA trae el menu a C. Este elige los platos y se los dice a CA, que toma nota de ello.
ESCENA-3: COMER C come los platos pedidos. Si sigue con hambre, llama al camarero y vuelve a la ESCENA-2.
ESCENA-4: SALIR CA le da la cuenta a C, que la paga. CA ingresa el dinero en la caja de P. C se levanta y sale del restaurante. FIN.

Los eventos que forman parte de un guión se encuentran encadenados unos a otros en relacion causa-efecto. A su vez, los guiones pueden encadenarse también unos a otros, cuando las condiciones de salida de un guión son las condiciones de entrada de otro. Por ejemplo, es posible que para ejecutar el guión RESTAURANTE sea necesario ejecutar antes un guión que haga cierta la condicion de entrada Tiene-dinero(C). Este guión podría ser, por ejemplo, el guión IR-AL-BANCO.

Por tanto, al igual que se hablaba de sistemas de marcos y no de un marco individualizado, parece más correcto hablar de sistemas de guiones. Se tiene así una representación adecuada para grandes cadenas causales de hechos y tareas, que reflejan los patrones de ocurrencia de los eventos en el mundo real. Esto permite también responder a preguntas relacionadas con las causas o los efectos, dada una situación. Por ejemplo: ¿Qué se debe hacer si se tiene hambre?, ¿Qué sucede con el dinero que se paga al camarero?, etc.

8.2 Razonamiento con incertidumbre

En muchos casos, es necesario que los modelos de representación que se han visto, se adapten, en el sentido de que permitan tratar un cierto grado de incertidumbre, ya que, a veces, es obligado representar un mundo en el que no se dispone de conocimiento completo, consistente y constante. Un caso típico lo constituye el diagnóstico médico, que realiza inferencias basándose en la abducción, que no es una regla de inferencia legal, como pueda serlo el Modus Ponens, pero, a pesar de ello, puede resultar útil.

Básicamente, la abducción se basa en que, partiendo de la proposición (IF A THEN B) y de la proposición B, se deduce A. Evidentemente, esto puede conducir a conclusiones incorrectas. Véase un ejemplo:

$\forall x, \text{IF es-mafioso}(x) \text{ THEN usa-pistola}(x)$ $\text{usa-pistola}(\text{Pepe})$
$\text{es-mafioso}(\text{Pepe}) \text{ ???}$

Es por esto por lo que la abducción no se considera una regla de inferencia de pleno derecho. Pero esto no implica que no sirva, pues bien utilizada puede ser de gran ayuda en muchos casos. De hecho, es ampliamente utilizada en la construcción de sistemas expertos de ayuda al diagnóstico, fundamentalmente médico.

En el ejemplo, se ve que es atrevido concluir que Pepe es un mafioso simplemente por el hecho de portar pistola. Pero imaginemos que posteriormente se conoce que Pepe tiene en su casa una foto en la que aparece con Don Vito Corleone en la Opera. Parece que aumenta la evidencia a favor de que Pepe es un mafioso. Si además, un testigo reconoce a Pepe como conductor del coche de Michael Corleone en la noche que este cometió el crimen, ya la evidencia es casi total a favor de que es un mafioso.

Pero el valor de evidencia también puede descender si, por ejemplo, posteriormente, alguien demuestra que el susodicho testigo es, en realidad, un mentiroso profesional y además, en la noche del crimen, el tal testigo se encontraba borracho.

Debe quedar claro, en este momento, que el uso cuidadoso de la abducción permite llegar a descubrir la verdad. No otra cosa es lo que hace un médico cuando observa los síntomas del paciente para concluir cual es su enfermedad. A su vez, la conclusión alcanzada ayuda, a su vez, a explicar los síntomas. La abducción no es más que una forma muy controlada de explicación.

Para ponderar correctamente la evidencia a favor o en contra de algún hecho deben conocerse dos cosas:

1. Cual es el peso de un hecho en favor o en contra de una conclusión.
2. Como combinar distintos valores de evidencia en una conclusión final.

Existe toda una profunda base matemática para tratar con valores numéricos que reflejen la incertidumbre. Se basa en la estadística y el cálculo de probabilidades. En particular, es fundamental la idea de probabilidad condicional, asociando un valor de probabilidad a cada regla IF-THEN. Por ejemplo:

$$\text{IF fumador}(x) \text{ THEN cancer-de-pulmon}(x) \text{ CON Prob}=0.35$$

Esto es lo mismo que expresar, en términos de probabilidad condicional:

$$P(\text{Cancer-de-pulmon}|\text{Fumador}) = 0.35$$

Los sistemas de razonamiento basado en la incertidumbre, generalmente disponen de una gran cantidad de valores de probabilidad condicional de la forma:

$$P(\text{Enfermedad}|\text{Síntoma}) = p$$

Y el mecanismo de inferencia trabaja con estos valores, combinándolos adecuadamente (según el Teorema de Bayes), hasta obtener finalmente un resultado que permita realizar un diagnóstico (o permita rechazar un diagnóstico previo). Los detalles de todo este proceso son demasiado complejos para este curso introductorio.

9 Ejercicios de autocomprobación

Problema 1.

Se pide modelizar todos los componentes de un sistema de producción, que, de la forma más general posible, pueda resolver integrales como

$$\int 6x(3x^2 + 1)^2 + 3x^2 \operatorname{sen} x^3 + \frac{2x}{x^2} dx$$

teniendo en cuenta

$$\int (u + v) dx = \int u dx + \int v dx$$

$$\int (f(x))^a f'(x) dx = (a + 1) \frac{f(x)}{a + 1} + c \quad (a \neq -1)$$

$$\int \frac{f'(x)}{f(x)} dx = \ln f(x) + c$$

$$\int f'(x) \operatorname{sen} f(x) dx = -\operatorname{cos} f(x) + c$$

$$\int a u dx = a \int u dx$$

y la siguiente tabla de derivadas:

Función	Derivada
$y=a$	$y'=0$
$y=x$	$y'=1$
$y=u^a$	$y'=a u^{a-1} u'$
$y=u+v$	$y'=u'+v'$
$y=au$	$y'=a u'$

donde u, v , y $f(x)$ son funciones de x , y a es una constante,

Solución al problema 1.

La siguiente es una posible solución al problema. Otras soluciones serían igualmente válidas. La Base de Hechos inicial sería:

$$\text{BH} = \{(\operatorname{der} x \ 1 \ x) \ (\operatorname{cte} 3 \ x) \ (\operatorname{cte} 6 \ x) \ (\operatorname{cte} 1 \ x) \ (\operatorname{cte} 2 \ x)\}$$

donde $(\operatorname{der} x \ y \ z)$ representa el hecho de que y es la derivada de x respecto de z , y $(\operatorname{cte} x \ y)$ representa el hecho de que x es una constante respecto a la variable y . La meta es conseguir en la Base de Hechos:

$$\begin{aligned}
& (\text{int } (+ (+ (* (\text{elev } (+ (* 3 (\text{elev } x 2)) 1) 2) \\
& \quad \quad \quad (* 6 x)) \\
& \quad \quad \quad (* 3 (* (\text{elev } x 2) (\text{sen } (\text{elev } x 3)))))) \\
& \quad \quad (/ (* 2 x) (\text{elev } x 2))) \\
& \quad \$1 \\
& \quad x)
\end{aligned}$$

donde $(\text{int } x y z)$ representa el hecho de que la integral de x respecto de z es y , y $\$1$ representa la solución al problema. Las variables se van a representar precedidas de un $\$$. La Base de Reglas estaría formada por las reglas:

- R1. $(\text{int } \$f1 \$i1 \$v) \wedge (\text{int } \$f2 \$i2 \$v) \rightarrow (\text{int } (+ \$f1 \$f2) (+ \$i1 \$i2) \$v)$
R2. $(\text{der } \$f \$d \$v) \wedge (\text{cte } \$c \$v) \rightarrow (\text{int } (* (\text{elev } \$f \$c) \$d) (+ (/ (\text{elev } \$f (+ \$c 1)) (+ \$c 1)) c) \$v)$
R3. $(\text{der } \$f1 \$d1 \$v) \rightarrow (\text{int } (/ \$d1 \$f1) (+ (\ln \$f1) c) \$v)$
R4. $(\text{der } \$f1 \$d1 \$v) \rightarrow (\text{int } (* \$d1 (\text{sen } \$f1)) (- c (\cos \$f1)) \$v)$
R5. $(\text{int } \$f1 \$i1 \$v) \wedge (\text{cte } \$c \$v) \rightarrow (\text{int } (* \$c \$f1) (* \$c \$i1) \$v)$
R6. $(\text{cte } \$c \$v) \rightarrow (\text{der } \$c 0 \$v)$
R7. $(\text{der } \$f1 \$d1 \$v) \wedge (\text{cte } \$c \$v) \rightarrow (\text{der } (* \$c \$f1) (* \$c \$d1) \$v)$
R8. $(\text{der } \$f1 \$d1 \$v) \wedge (\text{cte } \$c \$v) \rightarrow (\text{der } (\text{elev } \$f1 \$c) (* \$c (* (\text{elev } \$f1 (- \$c 1)) \$d1)) \$v)$
R9. $(\text{der } \$f1 \$d1 \$v) \wedge (\text{der } \$f2 \$d2 \$v) \rightarrow (\text{der } (+ \$f1 \$f2) (+ \$d1 \$d2) \$v)$

donde $(\text{elev } x y)$ es la función elevar a x a y . La estrategia de control sería un motor de inferencias hacia atrás, ya que se tiene un gran factor de ramificación hacia adelante y pequeño hacia atrás.

Problema 2.

Supóngase un mundo de los bloques, como el visto en clase, con las siguientes características adicionales:

- Cada bloque tiene un color en cada cara, que puede ser blanco, negro, azul o rojo.
- Hay dos brazos de robot.
- Cada brazo de robot puede levantar un bloque por una cara C si la cara C está hacia arriba, libre, y el otro brazo está libre o tiene cogido un bloque por una cara C' tal que C y C' tienen el mismo color.
- Un bloque sólo puede estar encima de otro si las dos caras en contacto tienen el mismo color.
- Cualquiera de los dos brazos de robot que esté libre puede empujar por una cara a un bloque que esté encima de la mesa y libre, de forma que el bloque girará y la cara empujada pasará a estar hacia arriba.
- Cualquiera de los dos brazos de robot que esté libre puede pintar un bloque de cualquiera de los cuatro colores.

- Puede haber muchos brazos de robot.
- El color rojo es incompatible con el azul y con el negro, y el color negro es el opuesto del blanco.
- Hay dos tipos de brazo de robot: los que levantan y los que empujan. Los que levantan están colgados de poleas, que deben resistir el peso del robot y el bloque. Los que empujan deben tener la fuerza necesaria para empujar el bloque.

Representar mediante marcos el conocimiento que se dispone del dominio anterior. Se debe mostrar al menos un método o demonio asociado a las restricciones mencionadas arriba.

Solución al problema 2.

Una posible jerarquía de marcos es la que se muestra en la siguiente tabla.

Marco	Atributo	Posibles valores
Soporte	Nombre	Identificador
Bloque	Subclase-de Peso Encima-de Libre Sujeto-por Cara-superior Cara-inferior Cara-lateral-1 Cara-lateral-2 Cara-lateral-3 Cara-lateral-4	Soporte Número Instancia de Soporte o Nada Verdadero o Falso Instancia de Robot o Nada Instancia de Cara Instancia de Cara Instancia de Cara Instancia de Cara Instancia de Cara Instancia de Cara
Mesa	Subclase-de Nombre	Soporte <i>mesa</i>
Cara	Nombre Color Opuesta-a	Identificador Instancia de Color Instancia de Cara
Color	Nombre Incompatible-con Opuesto-a	Identificador Instancia de Color Instancia de Color
Robot	Nombre Sujeta-a	Identificador Instancia de Bloque o Nada
Levantador	Subclase-de Peso Colgado-de	Robot Número Instancia de Polea
Empujador	Subclase-de Fuerza	Robot Número
Polea	Nombre Peso-máximo Sujeta-a	Identificador Número Instancia de Levantador

Podríamos plantear muchos métodos posibles, así como demonios. Como ejemplos, vamos a describir cómo implementar la acción Dejar y el demonio que comprueba si un robot levantador puede resistir el peso de un objeto. En el examen no era necesario llegar a este nivel de detalle, sino especificar qué haría cada uno.

Método robot.Levantar-Otro-Libre (self,bloque)

```
(* self es una instancia de robot levantador y bloque es una instancia de bloque *)
if robot.sujeta-a(self)=Nada AND
  bloque.libre(bloque)=Verdadero AND
  soporte.identificador(bloque.encima-de(bloque))=mesa AND
   $\exists R'$  instancia de Robot |  $R' \neq self$  AND
  (robot.sujeta-a(R')=Nada OR
  cara.color(bloque.cara-superior(robot.sujeta-a(R')))=
  cara.color(bloque.cara-superior(bloque)))
then robot.sujeta-a(self):=bloque
  bloque.sujeto-por(bloque):=self
  bloque.libre(bloque):=Falso
  bloque.encima-de(bloque):=Nada
```

Demonio levantador.sujeta-a.si-cambio (self,nuevo-valor)

```
(* se dispara cuando se intenta cambiar el valor del atributo sujeta-a
de cualquier robot levantador *)
if polea.peso-máximo(levantador.colgado-de(self))<
  levantador.peso(self) + bloque.peso(nuevo-valor)
then return levantador.sujeta-a(self):=nuevo-valor
else write('Error: robot no puede con objeto')
return levantador.sujeta-a(self)
```

Problema 3.

Los gatos y el ratón es un juego que se juega en un tablero de ajedrez (8x8) sobre las casillas negras. Un jugador mueve los gatos, que son cuatro fichas negras situadas inicialmente en las casillas negras de la primera fila. El otro jugador mueve el ratón, que es una ficha blanca que, inicialmente, está en cualquiera de las casillas negras de la última fila (fila 8). Los jugadores se van turnando en sus movimientos. El jugador de los gatos solo puede mover un gato en cada turno, ocupando una de las posibles dos casillas negras contiguas que se sitúan en diagonal hacia adelante con respecto a la que ocupa. El jugador del ratón puede mover en cada turno el ratón a una de las posibles cuatro casillas negras contiguas en diagonal a la que ocupa. Los gatos ganan la partida si consiguen impedir el movimiento del ratón, acorralando al ratón. El ratón gana si se encuentra en una fila con menor número que la del gato más atrasado, debido a que los gatos solo pueden ir hacia adelante y llegará un momento en el que no puedan mover.

Se pide formalizar este dominio por medio de un sistema de producción, que, dependiendo del papel con el que juegue el ordenador (gatos o ratón), efectúe los movimientos. El sistema de producción deberá también detectar la situación de fin de partida. Especificuense **todos** los componentes del sistema de producción.

Solución al problema 3.

Base de Hechos inicial=

$$\{(Ordenador-Juega-con,\$Papel), (1,1,Gato,Gato1),(1,3,Gato,Gato2), \\ (1,5,Gato,Gato3),(1,7,Gato,Gato4),(8,\$r,Ratón), (2,\$x,Vacío),(3,\$y,Vacío), \\ (4,\$x,Vacío),\dots,(8,\$z,Vacío), (0,\$p,Gato,Gato-Falso),(9,\$q,Gato,Gato-Falso), \\ (\$q,9,Gato,Gato-Falso),(\$p,0,Gato,Gato-Falso)\}$$

donde $\$Papel = \text{Gatos o Ratón}$; $\$r = 2, 4, 6$ u 8 ; $\$x = 2, 4, 6$ y 8 ; $\$y = 1, 3, 5$, y 7 ; $\$z = 2, 4, 6$, y 8 menos la posición donde se sitúe inicialmente el ratón; $\$p = 0, 2, 4, 6$, y 8 ; y $\$q = 1, 3, 5, 7$, y 9 . Cuando se estudien las reglas de fin, se estudiará por qué se han introducido las casillas ficticias últimas referentes a los gatos.

La Base de Hechos final es la dada por alguna de las situaciones de fin, especificadas por las reglas de fin.

La Base de Reglas:

Reglas de los Gatos

R1. $(Ordenador-Juega-con,Gatos), (\$i,\$j,Gato,\$Gato), (\$k,\$l,Vacío),$
 $\$i = \$k + 1, \$j = \$l - 1, \$i < 8, \$j > 1$
 $\rightarrow -(\$i,\$j,Gato,\$Gato) - (\$k,\$l,Vacío) + (\$k,\$l,Gato,\$Gato) + (\$i,\$j,Vacío)$
 R2. $(Ordenador-Juega-con,Gatos), (\$i,\$j,Gato,\$Gato), (\$k,\$l,Vacío),$
 $\$i = \$k + 1, \$j = \$l + 1, \$i < 8, \$j < 8$
 $\rightarrow -(\$i,\$j,Gato,\$Gato) - (\$k,\$l,Vacío) + (\$k,\$l,Gato,\$Gato) + (\$i,\$j,Vacío)$

Reglas del Ratón

R3. $(Ordenador-Juega-con,Ratón), (\$i,\$j,Ratón), (\$k,\$l,Vacío),$
 $\$i = \$k + 1, \$j = \$l - 1, \$i < 8, \$j > 1$
 $\rightarrow -(\$i,\$j,Ratón) - (\$k,\$l,Vacío) + (\$k,\$l,Ratón) + (\$i,\$j,Vacío)$
 R4. $(Ordenador-Juega-con,Ratón), (\$i,\$j,Ratón), (\$k,\$l,Vacío),$
 $\$i = \$k + 1, \$j = \$l + 1, \$i < 8, \$j < 8 \rightarrow$
 $-(\$i,\$j,Ratón) - (\$k,\$l,Vacío) + (\$k,\$l,Ratón) + (\$i,\$j,Vacío)$
 R5. $(Ordenador-Juega-con,Ratón), (\$i,\$j,Ratón), (\$k,\$l,Vacío),$
 $\$i = \$k - 1, \$j = \$l - 1, \$i > 1, \$j > 1 \rightarrow$
 $-(\$i,\$j,Ratón) - (\$k,\$l,Vacío) + (\$k,\$l,Ratón) + (\$i,\$j,Vacío)$
 R6. $(Ordenador-Juega-con,Ratón), (\$i,\$j,Ratón), (\$k,\$l,Vacío),$
 $\$i = \$k - 1, \$j = \$l + 1, \$i > 1, \$j < 8 \rightarrow$
 $-(\$i,\$j,Ratón) - (\$k,\$l,Vacío) + (\$k,\$l,Ratón) + (\$i,\$j,Vacío)$

Reglas de control de fin cuando ganan Gatos

En esta regla se comprueba que todo lo que rodea al Ratón son gatos. Por esto, se ponían en la Base de Hechos Inicial todas las ternas correspondientes a las casillas negras que rodean al tablero por fuera.

R7. $(\$i,\$j,Ratón), (\$g1,\$g2,Gato,\$Gato1), (\$g3,\$g4,Gato,\$Gato2),$
 $(\$g5,\$g6,Gato,\$Gato3), (\$g7,\$g8,Gato,\$Gato4), \$i = \$g1 - 1, \$j = \$g2 - 1,$
 $\$i = \$g3 - 1, \$j = \$g4 + 1, \$i = \$g5 + 1, \$j = \$g6 - 1,$
 $\$i = \$g7 + 1, \$j = \$g8 + 1 \rightarrow \text{FIN (ganan Gatos)}$

Reglas de control de fin cuando gana el Ratón

R8. $(\$i,\$j,Ratón), (\$g1,\$g2,Gato,\$Gato1), (\$g3,\$g4,Gato,\$Gato2),$
 $(\$g5,\$g6,Gato,\$Gato3), (\$g7,\$g8,Gato,\$Gato4), \$i < \$g1, \$i < \$g2,$

$i < g3, i < g4, \text{Gato1} \neq \text{Gato2} \neq \text{Gato3} \neq \text{Gato4} \neq \text{Gato-Falso}$
→ FIN (gana Ratón)

10 Ejercicios propuestos no resueltos

1. Intentar modelizar en un sistema de marcos un dominio compuesto por las figuras geométricas (círculo, cuadrado, triángulo, etc). Téngase en cuenta el mecanismo de herencia, e intente utilizar otras características de los marcos, como los demonios.
2. Observe la red semántica de la figura 12. Compáre el proceso de equiparación necesario para deducir cuál es el tipo de pistola que utiliza McCluskey, con el mismo proceso referido a la pistola de Harry el Sucio. Intente aclarar la cuestión de por qué existe un arco INST-DE que relaciona el arma de Michael Corleone con COLT-45 y por qué no es necesario un arco semejante para la pistola de McCluskey.
3. Representar mediante marcos el conocimiento que se dispone de la asignatura Inteligencia Artificial. Se deberán describir, al menos, veinte marcos (entre clases, subclasses e instancias) y una profundidad de cuatro niveles en alguna rama. Se deben especificar algunos atributos.

Búsqueda Heurística

11 Introducción a la búsqueda

En esta sección se van a tratar las técnicas básicas de búsqueda, partiendo de una forma de representar todos los problemas, el espacio de estados, tratado en la sección 12, pasando por las técnicas de búsqueda cuando no se dispone de información que guíe la solución de los problemas, búsqueda no informada, tratada en la sección 13, y finalizando en las técnicas de resolución de problemas que se pueden utilizar cuando se dispone de conocimiento que puede guiar la búsqueda, búsqueda informada o heurística, descritas en las secciones 14 y 15.

12 Espacio de estados

El espacio de estados es una técnica que permite formalizar problemas. Un espacio de estados se define en función de los dos elementos:

- Conjunto de estados. Representa el mundo por el que la técnica de búsqueda va a moverse para intentar resolver los problemas. Supóngase, por ejemplo, que se desea construir un sistema de integración simbólica, que sea capaz de, dada como entrada una integral indefinida, obtener su función primitiva. En este caso, los elementos de ese conjunto, los estados, serían todas las posibles integrales que se podrían formar, junto a fórmulas matemáticas intermedias. Así, $\int 3x^2 dx$, $\int \text{sen}(4x) dx$, o $x^3 + c$ serían estados del conjunto de estados.
- Conjunto de operadores. Representan la forma de moverse en ese conjunto de estados. Cada operador va a poder representarse como un arco que va de un estado a otro del conjunto de estados, con el significado de: si el método de resolución de problemas está en algún momento en un determinado estado, y se ejecuta un operador, entonces se puede pasar a otro estado. En el caso del dominio anterior, los operadores podrían ser todas las acciones que se pueden realizar en ese dominio, de forma que se puede pasar de una fórmula matemática a otra (de un estado a otro). Por ejemplo, podría haber un operador:

$$\text{OP1. } \int 3x dx \rightarrow 3 \int x dx$$

que permite pasar del estado que aparece en su izquierda, $\int 3x dx$, al estado que aparece en su derecha, $3 \int x dx$.

Los operadores tienen dos partes diferenciadas: la parte izquierda o precondiciones, que son las condiciones que se tienen que cumplir en un determinado estado para poder ser ejecutado el operador; y la parte derecha o postcondiciones, que representan la descripción del estado al que se puede transitar. Por ejemplo, se podrían tener, entre otros, los siguientes operadores:

- OP2. $\int (u + v)dx \rightarrow \int u dx + \int v dx$
- OP3. $\int (f(x))^a f'(x)dx \rightarrow (a + 1)\frac{f(x)}{a+1} + c \quad (a \neq -1)$
- OP4. $\int \frac{f'(x)}{f(x)}dx \rightarrow \ln f(x) + c$
- OP5. $\int f'(x) \operatorname{sen} f(x) dx \rightarrow -\operatorname{cos} f(x) + c$
- OP6. $\int a u dx \rightarrow a \int u dx$
- OP7. $y=a \rightarrow y'=0$
- OP8. $y=x \rightarrow y'=1$
- OP9. $y=u^a \rightarrow y'=a u^{a-1} u'$
- OP10. $y=u+v \rightarrow y'=u'+v'$
- OP11. $y=au \rightarrow y'=a u'$

donde a es una constante, y $f(x)$, u , y v son funciones de x . Como se puede observar, el primer operador, OP1, es un operador muy particular, ya que permite transitar de un único estado, $\int 3x dx$, a otro único estado, $3 \int x dx$. En el caso de los demás operadores, son operadores generales, ya que permiten pasar de un estado a otro, sin especificar exactamente los estados inicial y final, sino especificando condiciones que se tienen que dar en un estado para pasar a otro estado. Así, el operador OP6, que es el caso más general del operador OP1, permite pasar de estados que equiparen con la descripción de su parte izquierda, como, por ejemplo: $\int 3x^2 dx$ o $\int 4(x^3 + x^2) dx$, a los estados correspondientes de acuerdo a lo que dicte la parte derecha. En los dos casos mencionados, el operador OP6 permitiría pasar, respectivamente, a los estados $3 \int x^2 dx$ o $4 \int (x^3 + x^2) dx$, donde, en un caso, $\{u=x^2, a = 3\}$ y, en el otro, $\{u=(x^3 + x^2), a = 4\}$. Es decir, hay una asignación a las variables del operador, u y a .

Un espacio de estados se puede representar, pues, por un grafo dirigido en el que los nodos son los estados y los arcos son los operadores instanciados (asignando valores a las variables del operador) que permiten pasar de un estado a otro. En el caso del ejemplo de las integrales, la figura 17 muestra una porción del espacio de estados.

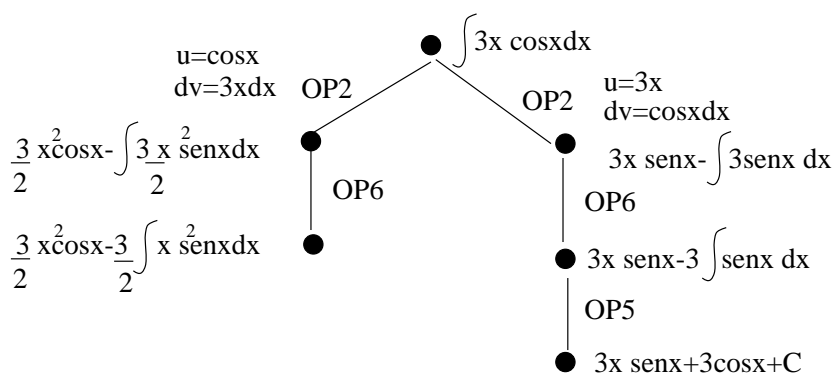


Figure 17: Parte del espacio de estados de las integrales.

Una vez descrito el espacio de estados, un problema se define como la tripleta: $\langle EE, EI, EF \rangle$, donde EE es el espacio de estados, EI es el estado inicial del problema, y EF es el estado final. En el caso del integrador simbólico, un estado inicial sería una determinada integral, como, por ejemplo, $\int 6x(3x^2 + 1)^2 + 3x^2 \operatorname{sen} x^3 + \frac{2x}{x^2} dx$ y un estado final sería su solución, $(\frac{3x^2+1}{3})^3 - \operatorname{cos} x^3 + \ln x^2 + c$.

En muchos casos, no se conoce el estado final, pero sí una descripción del mismo, como el caso del integrador, en el que no se conoce “a priori” la solución, pero sí se sabe cuándo se ha llegado a ella: cuando la fórmula que representa el estado no contiene el símbolo de integral.

Por tanto, la resolución de problemas bajo la formalización de espacio de estados consiste en recorrer el grafo que lo representa, partiendo del nodo inicial (estado inicial), hasta que se alcanza un nodo que, o bien es el estado final, o bien cumple la descripción del estado final deseado. Normalmente, se desarrolla un árbol de búsqueda para resolver el problema. Debido a que puede haber muchas formas de realizar esa búsqueda, y puede haber muchos arcos (operadores instanciados) que partan de un determinado nodo (estado), la dificultad de los problemas va a consistir en encontrar el camino que conecta los dos nodos que representan el estado inicial y final en grafos grandes. Supóngase, por ejemplo, que el espacio de estados es el del ajedrez. El conjunto de estados sería el formado por todas las posibles situaciones de juego, el conjunto de operadores serían las reglas del juego, el estado inicial es el tablero inicial, y el estado final es uno en que se haya ganado la partida. En este caso, la búsqueda consistiría en encontrar una estrategia que, dada cualquier jugada del contrario, nos haga ganar la partida. Dado que el conjunto de estados es inmenso, y el número de arcos que salen de cada nodo es muy grande, entonces la solución del problema del ajedrez es considerado como muy difícil, y, hasta ahora, no se ha podido generar dicha estrategia.

Se van a definir dos parámetros que van a servir para caracterizar numéricamente la complejidad del espacio de búsqueda: factor de ramificación (fr) y profundidad del árbol (p). El factor de ramificación de un nodo se define como el número de nodos sucesores que tiene. Como los sucesores de un nodo en el árbol de búsqueda son los nodos accesibles desde ese en el grafo del conjunto de estados, y los accesibles desde un nodo corresponden al número de operadores instanciados aplicables en ese nodo, el factor de ramificación se puede definir también como el número de operadores instanciados que se pueden aplicar en un determinado estado. Normalmente, se suele utilizar el factor de ramificación medio de todo el árbol para calcular la complejidad del problema. Este parámetro se obtiene de:

$$fr_{medio} = \frac{\sum_{i=1}^n fr(i)}{n}$$

donde n es el número de nodos del árbol y $fr(i)$ es el factor de ramificación del nodo i . Por otra parte, la profundidad de un nodo en el árbol representa el número de niveles que se han recorrido para llegar a él desde el nodo raíz del árbol, o bien por el número de operadores aplicados. El factor de ramificación da una idea de lo ancho del árbol de búsqueda, mientras que la profundidad da una idea de la altura del árbol.

13 Búsqueda no informada

Existen muchos espacios de estados donde no se dispone de información de por dónde se debe realizar la búsqueda. Es decir, para cada decisión de por qué arco ir desde cada nodo, no se conoce el mejor camino a la solución. En estos casos, existen algunos métodos conocidos de la teoría de grafos que permiten recorrer el espacio de estados para encontrar la solución. En esta sección se van a tratar el método de generar y comprobar, la búsqueda en amplitud, en profundidad, hacia atrás, bidireccional, y con islas.

13.1 Método de generar y comprobar

Cuando el espacio de estados no es muy grande, la técnica más sencilla para recorrerlo consiste en generar todos los posibles caminos desde el estado inicial y comprobar, para cada uno, si es el estado final o no. Supóngase, por ejemplo, en el dominio de las integrales que se define un subespacio de estados, en el que se enumeran todos los posibles caminos desde el estado inicial $\int 3x^5x dx$, teniendo sólo en cuenta los operadores OP6 y OP3. Entonces, la técnica de generar y comprobar, enumeraría todos los posibles caminos alcanzables desde el estado inicial en tres pasos (número de pasos de la solución) y, para cada uno, comprobaría si el estado correspondiente tiene el símbolo de integral o no. Podría generar, por ejemplo, las secuencias OP3-OP3-OP3, OP3-OP3-OP6, OP3-OP6-OP3, ..., OP6-OP6-OP6. Luego, comprobaría que la OP3-OP3-OP3 no llega a la solución, la siguiente tampoco, y así sucesivamente, hasta llegar a la OP6-OP6-OP3, que sí llega a la solución. Evidentemente, este tipo de técnica puede llegar a ser totalmente ineficaz en grandes espacios de estados, ya que podría tardar mucho tiempo en estudiar todos los posibles caminos hacia la solución.

13.2 Búsqueda en amplitud

La búsqueda en amplitud es una forma ordenada de recorrer los posibles caminos desde el estado inicial. Consiste en recorrer primero todos los nodos accesibles directamente desde el estado inicial, atravesando sólo un arco, que constituyen los sucesores del primer nivel. Si ninguno de ellos es la solución, se estudian, por orden, los nodos accesibles desde los nodos sucesores del primer nivel, siendo estos los nodos que configuran el segundo nivel. Así, se iría creando un árbol de búsqueda por niveles; no se estudiará un nodo de un determinado nivel, hasta que no se estudien los nodos de niveles anteriores. El algoritmo de este tipo de búsqueda se muestra en la tabla adjunta. La lista ABIERTA representa el conjunto de nodos generados en el árbol de búsqueda, pero de los que no se han generado sus sucesores. Funciona como el concepto informático de cola, en la que los primeros nodos generados serán los primeros en salir (FIFO, "first-in first-out"), al introducirse los nuevos nodos siempre al final de ABIERTA.

Procedimiento Amplitud (Estado-inicial Estado-Final)

1. Crear lista ABIERTA con el nodo inicial, I, que representa el Estado-inicial
2. EXITO=Falseo
3. Hasta que ABIERTA esté vacía O EXITO
 - Quitar de ABIERTA el primer nodo. Lo llamaremos N
 - Si N tiene sucesores
 - Entonces
 - Generar los sucesores de N
 - Crear punteros desde los sucesores hacia N
 - Añadir los sucesores al final de ABIERTA
 - Si el primer nodo de ABIERTA es el Estado-Final
 - Entonces EXITO=Verdadero
- Fin de bucle
4. Si EXITO
 - Entonces Solución=camino desde I a N por los punteros
 - Si no, Solución=fracaso

Veamos el funcionamiento con un ejemplo en el 8-puzzle. El 8-puzzle es un dominio sencillo que permite describir fácilmente las técnicas de búsqueda. Consiste en una matriz de 3x3, en la que cada casilla tiene un número diferente del 1 al 8, habiendo una casilla vacía. El conjunto de estados está formado por todas las combinaciones posibles de números en casillas. El conjunto de operadores son las formas de transformar un estado en otro. En concreto, se pueden realizar los siguientes movimientos:

- OP1. Abajo. Mueve la casilla vacía a la casilla inferior.
- OP2. Arriba. Mueve la casilla vacía a la casilla superior.
- OP3. Derecha: Mueve la casilla vacía a la casilla de la derecha.
- OP4. Izquierda: Mueve la casilla vacía a la casilla de la izquierda.

La figura 18 muestra un ejemplo de problema (estado inicial y final) del 8-puzzle. La figura 19 muestra el árbol de búsqueda generado por la búsqueda en amplitud para ese problema. En la figura, los números representan el orden de expansión de los nodos, las flechas son los apuntadores desde cada nodo hasta su nodo padre, las **x** representan los nodos de fallo, que corresponden a bucles de estado (nodos repetidos en el árbol de búsqueda en el camino que lleva a ese nodo desde el nodo raíz), la \checkmark representa el estado final, y, finalmente, los apuntadores en negrita son la solución del problema (la secuencia de operadores que hay que aplicar para resolverlo). La generación de los

sucesores de los nodos se realiza de acuerdo al orden en el que se han especificado los operadores. Así, el primer operador que se intenta aplicar es el OP1, luego OP2, OP3 y OP4.

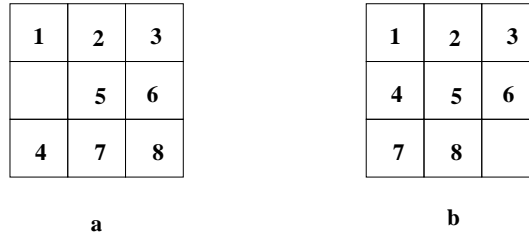


Figure 18: Estados inicial (a) y final (b) de un problema en el 8-puzzle.

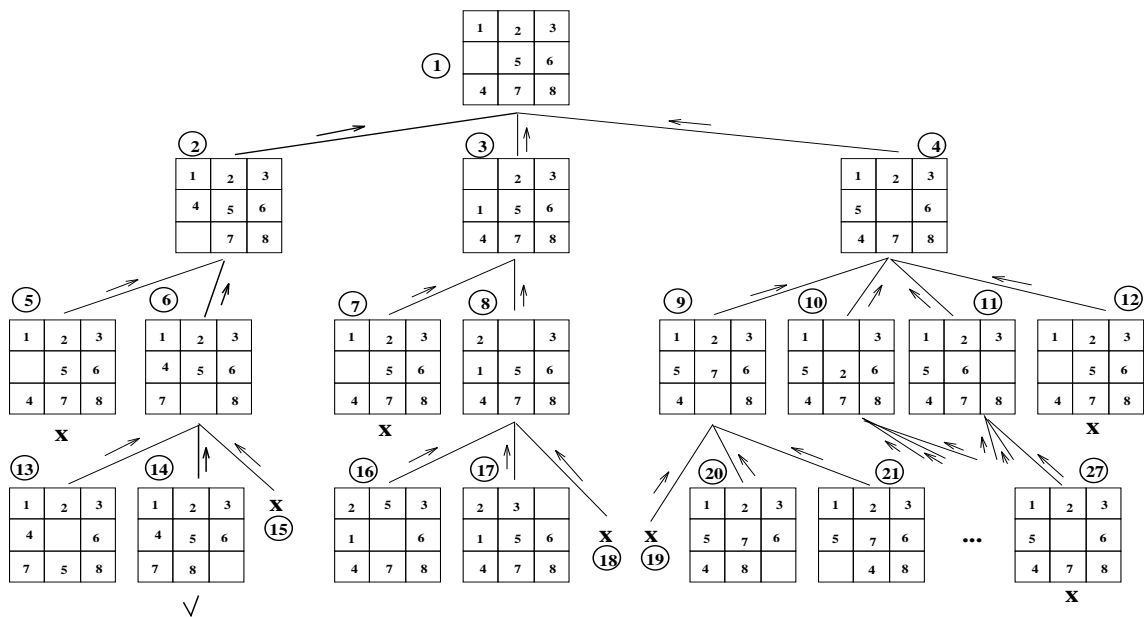


Figure 19: Árbol de búsqueda generado por amplitud en el problema de la figura 18.

Las ventajas que tiene la búsqueda en amplitud son:

- Si existe una solución y el factor de ramificación es finito, la encontrará.
- Si todos los arcos tienen igual coste (el coste de aplicar los operadores es el mismo), amplitud encuentra la solución óptima, si la hubiera.
- Es útil cuando las metas del problema son escasas y están cerca del estado inicial. En ese caso, como amplitud busca en la vecindad del estado inicial, las encontraría pronto.

Sin embargo, tiene el problema de que ocupa mucha memoria, ya que en cada paso almacena todos los sucesores de los nodos de un nivel, con lo que si se está en el nivel de profundidad p , y el factor de ramificación medio es fr , se tendrían almacenados fr^p nodos.

13.3 Búsqueda en profundidad

La búsqueda en profundidad elige el primer camino siempre, en lugar de expandir todos los sucesores de cada nivel. Así, el algoritmo, que se muestra en la tabla adjunta, explora el primer sucesor de cada nodo, realizando una búsqueda primero por la rama de la izquierda. Como podría generar indefinidamente nodos por la rama de la izquierda, se suele poner un tope de profundidad a partir del cual no se sigue estudiando. Cuando la búsqueda llega a un nodo con la profundidad máxima prefijada y no se ha llegado a la solución, se da marcha atrás, volviendo al nodo padre e intentando la búsqueda por otro camino. Esto es lo que se denomina la técnica de retroceso, fundamental para realizar una búsqueda en profundidad efectiva. Como se puede observar en el procedimiento, la lista ABIERTA se comporta como una pila, a diferencia del método de amplitud, ya que los últimos nodos generados serán los que se estudien primero, al introducirse al principio de ABIERTA (estrategia LIFO “last-in first-out”).

Procedimiento Profundidad (Estado-inicial Estado-Final)

1. Crear lista ABIERTA con el nodo inicial, I, y su profundidad=0
2. EXITO=Falseo
3. Hasta que ABIERTA esté vacía O EXITO
 - Quitar de ABIERTA el primer nodo.
 - Lo llamaremos N y a su profundidad P
 - Si $P < \text{Profundidad-máxima}$ Y N tiene sucesores
 - Entonces
 - Generar los sucesores de N
 - Crear punteros desde los sucesores hacia N
 - Añadir los sucesores al principio de ABIERTA con profundidad P+1
 - Si algún sucesor es el Estado-Final
 - Entonces EXITO=Verdadero
 - Fin de bucle
4. Si EXITO
 - Entonces Solución=camino desde I a N por los punteros
 - Si no, Solución=fracaso

Veamos el comportamiento con el mismo problema de la figura 18. En este caso, el árbol de búsqueda generado es el mostrado en la figura 20, donde también se ha especificado el orden de generación de los nodos. En este caso la búsqueda utilizando profundidad ha resuelto mejor el problema (ha expandido menor número de nodos). Sin embargo, esto no tiene por qué ser así en todos los casos. Las situaciones en las que funciona bien son:

- Las metas están alejadas del estado inicial y son numerosas. En ese caso, como amplitud va recorriendo poco a poco la vecindad del estado inicial tardaría mucho tiempo en llegar a la

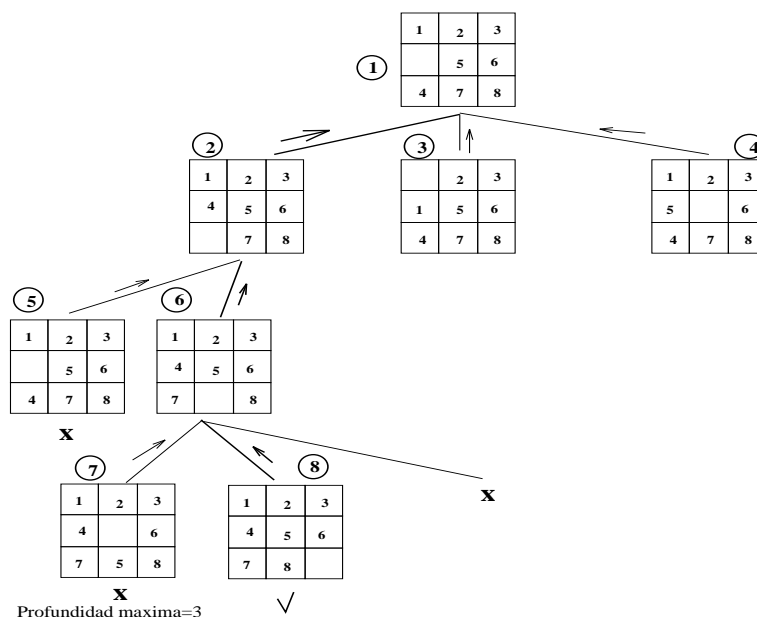


Figure 20: Árbol de búsqueda generado por profundidad en el problema de la figura 18.

solución, mientras que profundidad iría directo hacia la zona donde están las metas (lejos).

- Se dispone de poca memoria. Como profundidad almacena sólo los nodos del camino desde el nodo raíz hasta el último generado, requiere de media $fr * p$ nodos en ABIERTA, que es mucha menor memoria que el de amplitud. Por ejemplo, si $fr=10$ y $p = 5$, el método de búsqueda en amplitud guardaría $10^5 = 10.000$ nodos, frente a los $10 * 5 = 50$ nodos que guardaría profundidad.

Sin embargo, la técnica de profundidad no asegura encontrar la solución óptima.

La técnica de búsqueda en profundidad requiere disponer de la técnica de retroceso (“backtracking” en inglés). Las razones por las cuales se debe realizar retroceso son:

- Se ha llegado al límite de profundidad.
- Se han estudiado todos los sucesores de un nodo y no se ha llegado a la solución.
- Se sabe que el estado no conduce a la solución. Por ejemplo, cuando se genera un estado repetido (nodo 5).

13.4 Búsqueda hacia atrás

Hasta ahora se han estudiado procedimientos para, partiendo desde el estado inicial del problema, aplicando operadores, intentar llegar a las metas del mismo. Una forma alternativa de resolver el problema consiste en empezar la búsqueda en el estado final y, aplicando el inverso de los operadores, intentar llegar al estado inicial. En el caso del problema de la figura 18, el árbol generado por la búsqueda hacia atrás (o desde las metas) es el mostrado en la figura 21, en el que, como se puede observar, se ha comenzado desde el estado final y se han aplicado los operadores inversos de los que se emplean en la búsqueda hacia adelante. En este caso, los operadores inversos serían, por

ejemplo, el operador inverso *arriba* en lugar del operador *abajo*, o el operador inverso *izquierda* en lugar del operador *derecha*. En la búsqueda hacia atrás se puede elegir la técnica de búsqueda que se prefiera: generar y verificar, amplitud o profundidad. En el ejemplo se ha mostrado la de profundidad con una profundidad máxima de 3.

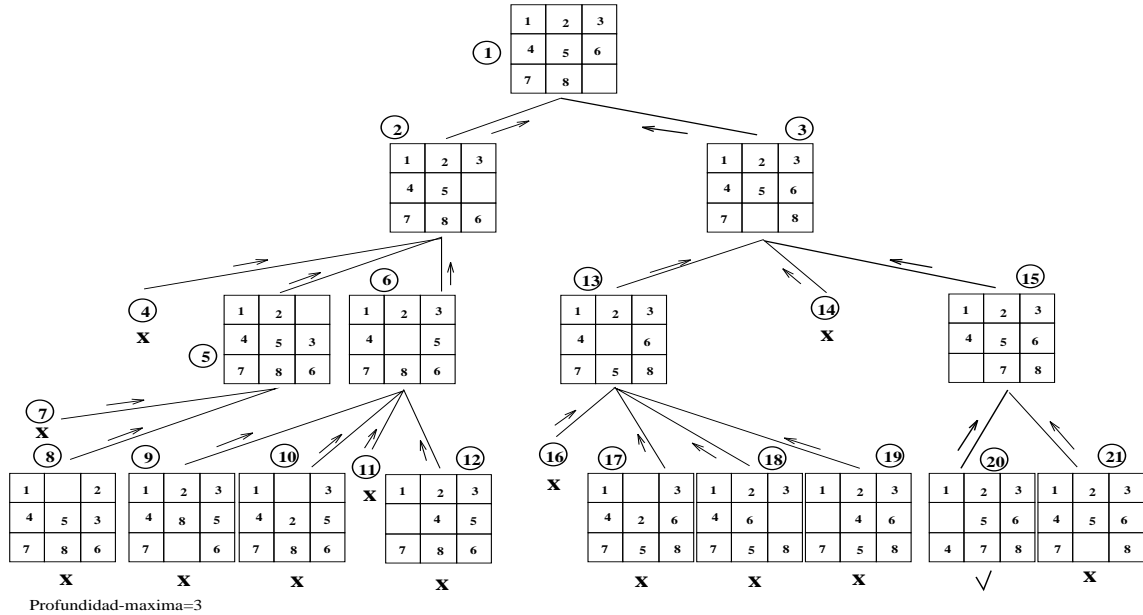


Figure 21: Árbol generado por búsqueda desde las metas en el problema de la figura 18.

13.5 Búsqueda bidireccional

Todas las técnicas estudiadas hasta el momento sólo tienen un sentido de búsqueda, que puede ser hacia adelante o hacia atrás. La búsqueda bidireccional permite realizar una búsqueda en los dos sentidos: hacia adelante y hacia atrás. Cada una de estas búsquedas se podrá realizar por medio de generar-y-comprobar, amplitud o profundidad. La tabla adjunta muestra el procedimiento de búsqueda bidireccional.

Procedimiento Bidireccional (Estado-inicial Estado-final)

1. ABIERTA-I=Estado-inicial; ABIERTA-M=Estado-final
EXITO=Falso
2. Hasta que EXITO
 - Quitar de ABIERTA-I el primer nodo, N
 - Generar sucesores de N, introducirlos en ABIERTA-I, y compararlos con todos los estados de ABIERTA-M
 - SI algún sucesor de N equipara con algún elemento de ABIERTA-M
ENTONCES EXITO=Verdadero
SI NO
 - Quitar de ABIERTA-M el primer nodo M
 - Generar sucesores de M, introducirlos en ABIERTA-M, y compararlos con todos los estados de ABIERTA-I
 - SI algún sucesor de M equipara con algún elemento de ABIERTA-I
ENTONCES EXITO=Verdadero
3. SI EXITO
ENTONCES Solución=camino desde nodo del Estado-inicial al nodo del Estado-final por los punteros
SI NO, Solución=fracaso

Supongamos que se desea resolver el problema de la figura 18 mediante una búsqueda bidireccional en la que la búsqueda hacia adelante se va a realizar por medio de la técnica de profundidad, y la búsqueda hacia atrás mediante la técnica de amplitud. La figura 22 muestra el árbol generado en el que los nodos están numerados de acuerdo al orden de generación. En la figura se observa que encuentra que dos nodos, uno perteneciente al árbol de búsqueda hacia adelante (nodo 9), y el otro perteneciente al árbol de búsqueda hacia atrás (nodo 7), son iguales, por lo que termina la búsqueda.

13.6 Búsqueda con islas

Si se dispone de información de estados intermedios del espacio de estados por los cuales debe pasar la búsqueda, entonces se puede utilizar la búsqueda con islas. Como se necesitan conocer estos estados intermedios, o islas, este tipo de búsqueda se puede también considerar dentro de la búsqueda informada. En el caso de que se conozcan las islas, se puede realizar una búsqueda entre el estado inicial y la primera isla, otra búsqueda entre la primera isla y la segunda, y así sucesivamente hasta la búsqueda entre la última isla y el estado final, tal como muestra esquemáticamente la figura 23.

En la figura 24 se muestra los dos árboles de búsqueda generados por la búsqueda con islas cuando se conoce que la búsqueda debe pasar por el estado representado por el nodo 6. Las dos búsquedas han sido en profundidad, aunque la búsqueda con islas permite realizar cualquier tipo de búsqueda entre dos estados tanto bidireccional como unidireccional.

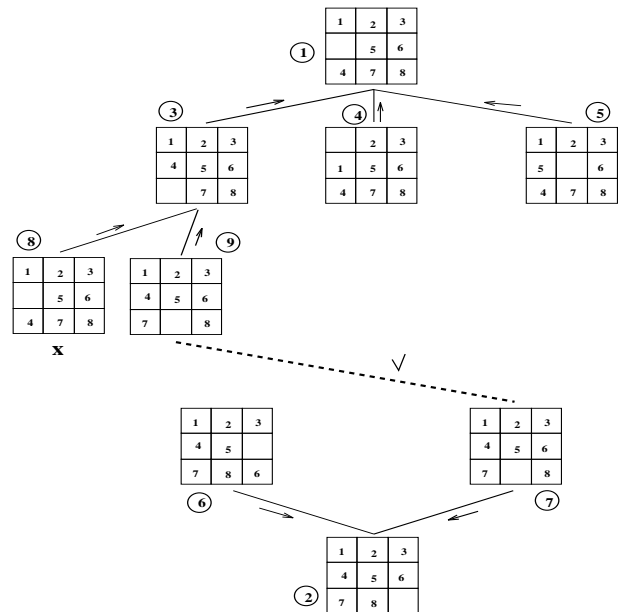


Figure 22: Árbol generado por búsqueda bidireccional en el problema de la figura 18.

13.7 Análisis de complejidad

En este apartado se va a estudiar la complejidad de cada uno de los tipos de búsqueda teniendo como parámetros de comparación el factor de ramificación, fr , y la profundidad, p . En la tabla 1 se muestra un resumen del máximo número de nodos de los árboles de búsqueda generados con cada técnica.

Las técnicas unidireccionales exploran, en el peor de los casos, fr^p nodos en cada nivel. Como hay p niveles de profundidad, para conocer el número máximo de nodos, se debe realizar el sumatorio para cada nivel. La técnica bidireccional realiza dos búsquedas unidireccionales que, si se encuentran en el medio, serán, cada una de profundidad $\frac{p}{2}$. Por tanto, el número de nodos máximo se calcula como la suma de dos búsquedas unidireccionales con la mitad de profundidad. Como el número de nodos crece exponencialmente con la profundidad, cuanta menos profundidad

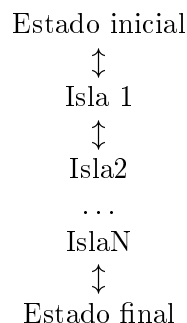


Figure 23: Árboles de búsqueda generados cuando se conocen estados intermedios por los que debe pasar la solución del problema.

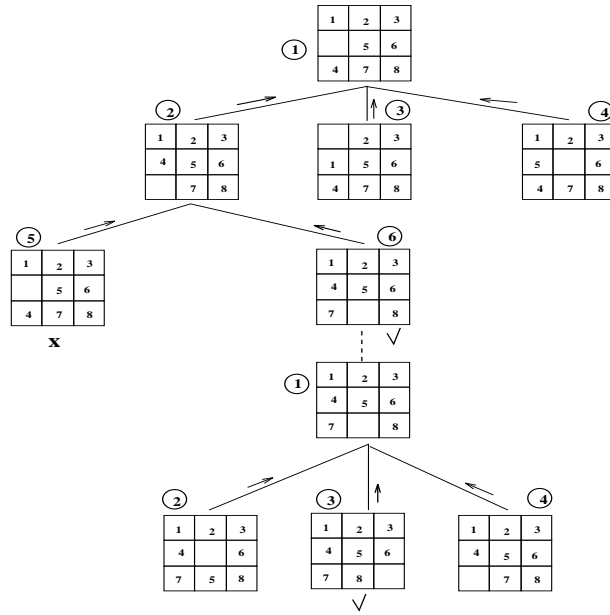


Figure 24: Árboles de búsqueda generados por la técnica de búsqueda con islas conociendo un estado intermedio (nodo 6).

Técnica de Búsqueda	Número máximo de nodos
Unidireccional	$\sum_{i=0}^p f r^i$
Bidireccional	$2 \sum_{i=0}^{\frac{p}{2}} f r^i$
Unidireccional (1 isla)	$2 \sum_{i=0}^{\frac{p}{2}} f r^i$
Unidireccional (k islas)	$(k + 1) \sum_{i=0}^{\frac{p}{k+1}} f r^i$
Bidireccional (k islas)	$2(k + 1) \sum_{i=0}^{\frac{p}{2(k+1)}} f r^i$

Table 1: Análisis de complejidad de las técnicas de búsqueda.

tenga el árbol de búsqueda, se generarán muchos menos nodos, por lo que la técnica de búsqueda bidireccional exploraría muchos menos nodos que la unidireccional. En el caso de realizar una búsqueda unidireccional con 1 isla, se realizan 2 búsquedas unidireccionales con profundidad $\frac{p}{2}$ si la isla se sitúa en medio del camino de solución. Se tendrá otro resultado si la isla no se sitúa en medio. Por tanto, el número de nodos es el mismo que en la búsqueda bidireccional. En el caso de disponer de k islas, y hacer búsquedas unidireccionales entre cada par de nodos, se realizan $k + 1$ búsquedas, cada una de las cuales va a tener $\frac{p}{k+1}$ de profundidad si las islas están situadas uniformemente en el camino de solución. Por último, si se hace búsqueda bidireccional con k islas, se tendrán $2(k + 1)$ búsquedas cada una de las cuales será de profundidad $\frac{p}{2(k+1)}$ con las mismas suposiciones anteriores.

14 Búsqueda heurística

En casi todos los espacios de estados existe información que permite guiar los procesos de búsqueda. Normalmente, esa información no es perfecta, es decir, no es un algoritmo que permita conocer de forma precisa cuál es el mejor camino para obtener la solución. Este tipo de información no perfecta que ayuda a resolver problemas en un espacio de estados, pero que no siempre acierta con el mejor camino se denomina **heurística**. Cuando se dispone de este tipo de información, las técnicas de búsqueda se pueden ver muy beneficiadas de su utilización. Para ello, en esta sección y en la siguiente se estudiarán técnicas de búsqueda heurística divididas en dos clases: técnicas en dominios en los que existe un único agente que toma decisiones, como, por ejemplo, el 8-puzzle; y técnicas que se utilizan cuando existen al menos dos agentes que toman decisiones en el mismo espacio de estados, como, por ejemplo, todos los juegos.

14.1 Técnica de escalada

La técnica de escalada es la evolución de la técnica de profundidad, en la que, en cada nodo, se dispone de una forma de evaluar cómo está de cerca o lejos de la solución. La forma más común de evaluar es la **función de evaluación**. Como ejemplo, en el 8-puzzle se podría definir una función de evaluación que fuera:

$$f(\text{nodo}) = \text{número de casillas bien colocadas}(\text{nodo})$$

que devuelve un número que representa cómo está de cerca un determinado estado de la solución. Cuanto mayor sea el número, más cerca se estará de la solución. Por tanto, si se tiene que elegir entre varios estados, se debería escoger aquél que tuviera un mayor valor de esta función. Es decir, es una función que se debe maximizar. También se podría haber definido la función complementaria de la anterior, que sería:

$$f'(\text{nodo}) = \text{número de casillas mal colocadas}(\text{nodo})$$

En este caso, la función se debe minimizar, puesto que cuantas menos casillas estén mal colocadas, más cerca se estará de la solución. Habiendo definido estas dos funciones, se va presentar el procedimiento básico de escalada, mostrado en la tabla adjunta, con el mismo ejemplo del 8-puzzle que se ha presentado en las anteriores secciones y mostrado en la figura 18.

Procedimiento escalada (Estado-inicial Estado-final)

1. $N = \text{Estado-inicial}$; $\text{EXITO} = \text{Falso}$
2. Hasta que EXITO
 - Generar los sucesores de N
 - SI algún sucesor es Estado-final
ENTONCES $\text{EXITO} = \text{Verdadero}$
SI NO,
 - Evaluar cada nodo con la función de evaluación
 - $N = \text{mejor sucesor}$
3. SI EXITO
ENTONCES $\text{Solución} = \text{camino desde nodo del Estado-inicial al nodo } N \text{ por los punteros}$
SI NO, $\text{Solución} = \text{fracaso}$

Se va a utilizar la función f , y, por tanto, el mejor sucesor se va a calcular como el que tenga valor máximo de la función f . El árbol que la técnica de escalada generaría sería el que aparece en la figura 25. Los nodos aparecen numerados según dos criterios. Dentro de los círculos aparece el orden de generación del nodo, y en los cuadrados aparece el valor de la función de evaluación para cada nodo.

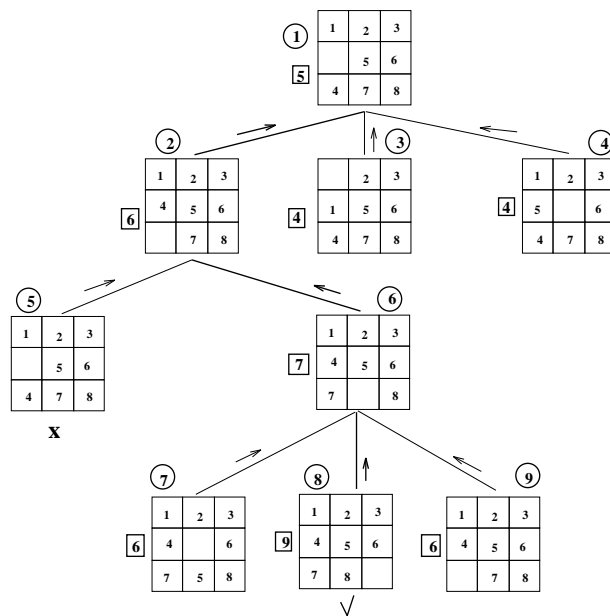


Figure 25: Árbol de búsqueda generado por la técnica de escalada para el problema de la figura 18.

La técnica de escalada exagera los problemas de la profundidad en el sentido de que no asegura que se alcance la solución óptima. Relacionado con esto, existen dos problemas que ocurren a menudo cuando se utiliza escalada:

- Puede haber máximos o mínimos locales. Esto ocurre, por ejemplo, cuando la función de evaluación elegida es maximizante y todos los sucesores de un determinado nodo tienen menor valor que el valor del nodo.
- Altiplanicies. Es un caso parecido al anterior y sucede cuando los mejores sucesores de un nodo tienen igual valor que el nodo.

Las soluciones que se suelen adoptar son:

- Retroceder. A la lista de razones por las que se debe retroceder, que provienen de la técnica de profundidad, se le añade que debe retroceder cuando ocurra cualquiera de los dos casos anteriores.
- Dar más de un paso. En lugar de retroceder, se generan todos los sucesores de los sucesores del nodo en cuestión. Si, aún así, no hay ningún sucesor de los sucesores que sea mejor que el nodo, se puede expandir un nivel más, hasta que se obtenga un valor mayor/menor que el nodo.

Una generalización de la técnica de escalada, denominada búsqueda en haz (en inglés “beam-search”), consiste en la selección en cada paso de más de un sucesor para expandir. En este caso, existe un parámetro k , que es el número fijo de nodos que se expanden en cada nivel. En el caso de escalada, $k = 1$. A continuación, se muestra el algoritmo de búsqueda en haz.

Procedimiento Búsqueda en haz (Estado-inicial Estado-final K)

1. ABIERTA=(Estado-inicial); EXITO=Falso
2. Hasta que EXITO
 - ABIERTA=Todos los sucesores de los nodos de ABIERTA
 - SI algún nodo de ABIERTA es Estado-final
ENTONCES EXITO=Verdadero
SI NO,
 - Evaluar cada nodo con la función de evaluación
 - ABIERTA=K mejores nodos de ABIERTA
- 3Si EXITO
Entonces Solución=camino desde nodo del Estado-inicial
al nodo N por los punteros
Si no, Solución=fracaso

En la figura 26 se muestra cómo expandiría este procedimiento un problema del 8-puzzle.

14.2 Técnicas de mejor-primero. A*

La técnica de escalada es una técnica local, ya que, cuando tiene que tomar una decisión de qué nodo estudiar a continuación, sólo toma en cuenta los nodos sucesores del nodo activo, sin tener en

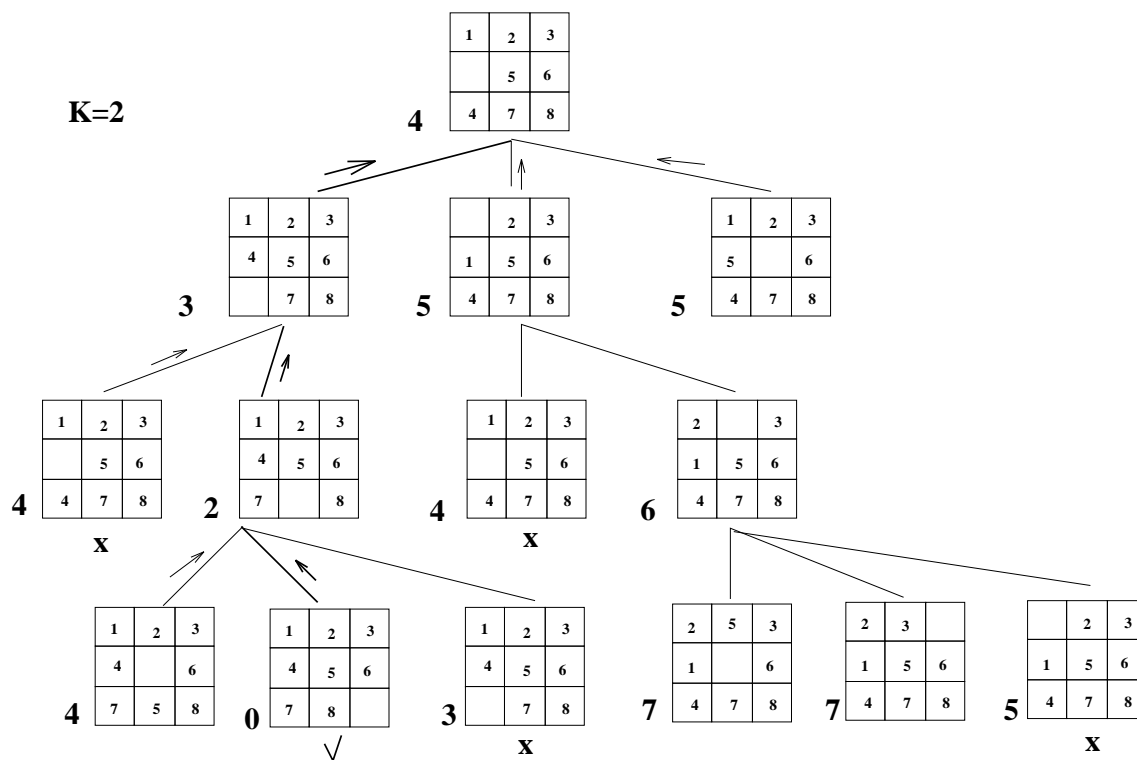


Figure 26: Ejemplo de solución del 8-puzzle utilizando la técnica de búsqueda en haz.

cuenta los otros nodos del árbol, generados pero no expandidos. Las técnicas de mejor-primero son técnicas globales en el sentido de que sí tienen en cuenta todos los nodos generados no expandidos del árbol a la hora de tomar la decisión de qué nodo expandir en cada momento. Para ello, mantienen dos listas de nodos: ABIERTA, que contiene los nodos generados, pero no estudiados (expandidos); y CERRADA, que contiene los nodos generados y estudiados. En cada decisión, tiene en cuenta todos los nodos de ABIERTA. La tabla adjunta presenta el procedimiento general de búsqueda mejor-primero.

Procedimiento Mejor-primero (Estado-inicial Estado-final)

1. Crear grafo de búsqueda G , con el nodo inicial, I , que representa el Estado-inicial
2. ABIERTA= I ; CERRADA=Vacío; EXITO=Falso
3. Hasta que ABIERTA esté vacía O EXITO
 - Quitar el primer nodo de ABIERTA, N
 - Introducir N en CERRADA
 - SI N es Estado-final
ENTONCES EXITO=Verdadero
SI NO
 1. Expandir N , generando el conjunto S de sucesores de N , que no son antecesores de N en el grafo
 2. Generar un nodo en G por cada s de S
 3. Establecer un puntero a N desde aquellos s de S que no estuvieran ya en G , añadiéndolos a ABIERTA
 4. Para cada s de S que estuviera ya en ABIERTA o CERRADA decidir si redirigir o no sus punteros hacia N
 5. Para cada s de S que estuviera ya en CERRADA decidir si redirigir o no los punteros de sus sucesores hacia s
 6. Reordenar ABIERTA
7. SI EXITO
ENTONCES Solución=camino desde I a N a través de los punteros
SI NO Solución=Fracaso

Antes de pasar a explicar paso por paso el funcionamiento del algoritmo mediante un ejemplo, estudiaremos en más detalle el paso 5f. En este paso, se reordena la lista ABIERTA y, para ello, se utiliza otra vez el concepto de función heurística visto en la subsección anterior. Es precisamente en ese paso en el que se diferencian los procedimientos de Mejor-primero. El ejemplo más conocido de procedimiento de este tipo es el algoritmo A*. En este algoritmo, se supone que la ejecución de cada operador lleva asociado un coste. Se trata, pues, de obtener la solución con el menor coste, es decir, el camino menos costoso que conecte el estado inicial y el estado final. La función de evaluación que se intenta minimizar tiene la forma general:

$$f(n) = g(n) + h(n)$$

donde $f(n)$ es la función de evaluación aplicada al nodo n , $g(n)$ es el coste estimado de ir desde el nodo raíz del grafo de búsqueda al nodo n , y $h(n)$ es el coste estimado para ir desde el nodo n al Estado-final. La función $g(n)$ refleja el coste que, hasta el momento, se ha seguido para pasar del nodo raíz al nodo n . Se calcula como la suma de los costes de los operadores utilizados en el camino menos costoso encontrado hasta el momento entre el nodo raíz y el nodo n . Como se verá más adelante, conforme se van estudiando otros nodos, se pueden encontrar caminos menos costosos para ir desde el nodo raíz hasta el n , por lo que este coste es estimado. Llamaremos $g^*(n)$ al coste real del camino menos costoso que hay entre el nodo raíz y el nodo n . Este valor sólo se

conocerá al final de la ejecución del algoritmo para los nodos que estén en el camino solución. Si se está, por ejemplo, en un espacio de estados configurado por estaciones de metro de una ciudad, y se desea conocer cuál es el recorrido de metro más corto (o por el que se tarda menos tiempo) para ir desde una estación a otra, $g^*(n)$ representa el camino más corto (o por el que se tarda menos tiempo) para llegar desde la estación origen a la final.

Por otra parte, $h(n)$ representa el conocimiento heurístico, ya que es una estimación de qué coste estimado se piensa va a tener el camino menos costoso desde el nodo n hasta el nodo meta. Al igual que en el caso anterior, $h^*(n)$ es el coste real mínimo desde el nodo n hasta el nodo meta, y que sólo se conocerá al final de la ejecución para los nodos situados en el camino de la solución (camino más corto entre estado inicial, nodo raíz, y estado final, nodo meta). En el ejemplo de las estaciones de metro, una posible $h(n)$ podría ser la distancia en línea recta entre las dos estaciones, puesto que da una idea de la distancia real que hay por metro. En el caso de que se desee trabajar con tiempos, se podría utilizar la siguiente función de evaluación:

$$h(n) = \frac{d(n, m)}{v_{max}}$$

donde $d(n, m)$ es la distancia en línea recta entre el nodo n y el nodo meta m , y v_{max} es la velocidad máxima que puede alcanzar el metro.

El algoritmo A^* tiene dos propiedades muy importantes: es completo, es decir, si existe una solución al problema la encontrará; y es admisible, es decir, si hay una solución óptima la encontrará. Esta última propiedad depende de una restricción: $h(n)$ deber ser menor o igual que $h^*(n)$ en todos los nodos del árbol de búsqueda. En el caso del metro, se cumple en las dos funciones definidas, por distancia o por tiempo. En el caso de la distancia, la línea recta siempre va a ser menor o igual que la distancia por línea de metro. En el caso del tiempo, como la distancia es la mínima también y la velocidad es la máxima y está en el denominador y hace, por tanto, que la fracción sea la mínima, entonces también está asegurado.

Para ver el funcionamiento del procedimiento A^* , se va a mostrar con el grafo de la figura 27, que puede representar conexiones entre varias ciudades, un trozo del mapa del metro, o cualquier otro dominio en el que se pueda definir un coste asociado a cada operador (arcos del grafo). En concreto, se va a suponer que el grafo es dirigido y se puede ir desde el nodo 1 al 2, pero no del 2 al 1. El problema consiste en hallar el camino de coste mínimo entre el nodo 1 y el nodo 5. El coste estimado $h(n)$ está representado dentro de los cuadrados al lado de cada nodo, mientras que el coste de utilizar cada operador viene representado en el grafo por los costes en los arcos.

Las figuras 28, 29, 30, y 31 muestran los sucesivos pasos del algoritmo y el grafo que se va generando. Los nodos en gris son los que están en CERRADA, los nodos en blanco los que están en ABIERTA, los valores de $g(n)$ para cada nodo están arriba a la derecha de cada nodo, los valores de $h(n)$ aparecen abajo a la derecha, y, por último, los valores de $f(n)$ están a la izquierda de cada nodo. En la figura 28, el algoritmo selecciona el nodo 1, que es el único de ABIERTA, lo mete en CERRADA y lo expande, generando los sucesores, 2, 3, y 4, que no estaban en el grafo de búsqueda (pasos 5a y 5b). Desde cada uno establece un puntero (paso 5c), que servirá para conocer, en cada momento, cuál es el camino más corto desde cada nodo al nodo raíz, y los mete en ABIERTA. Los pasos 5d y 5e no son aplicables, pues ningún nodo estaba ya en el grafo de búsqueda. Por último, el paso 5f reordena ABIERTA de mejor nodo a peor nodo. Para ello, necesita saber cuál es el valor de la $f(n)$ de cada nodo:

- $g(n)$ es el coste del mejor camino encontrado hasta el momento desde el nodo raíz a cada nodo. Como todos los caminos van directos al nodo raíz, es fácil de calcular y $g(2) = 200$, $g(3) = 60$, y $g(4) = 40$.

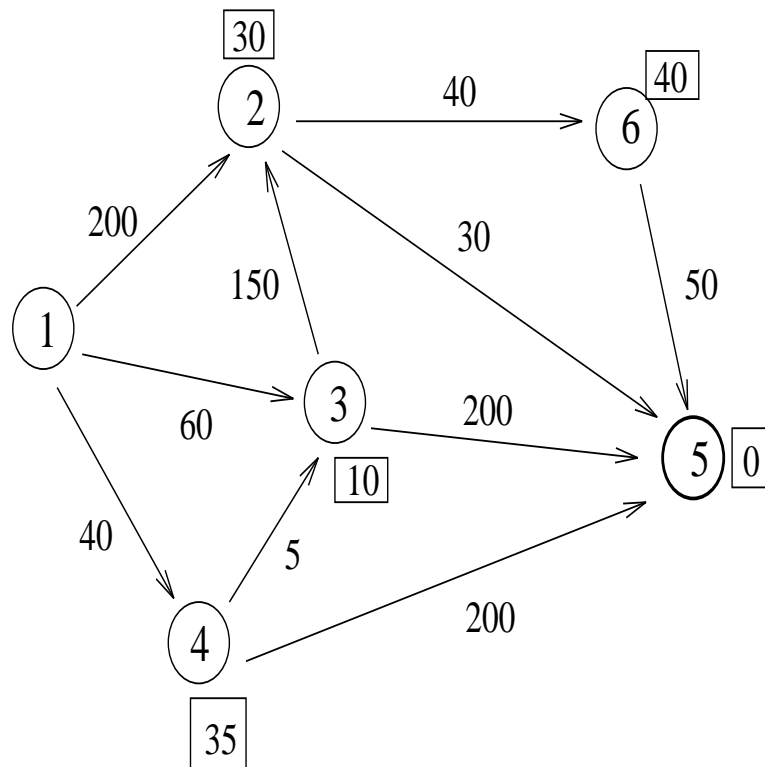


Figure 27: Grafo de entrada para el algoritmo A*.

- $h(n)$ viene dada en la descripción inicial y es $h(2) = 30$, $h(3) = 10$, y $h(4) = 35$. Con estos valores, se calculan las $f(n)$ como:

$$f(2) = g(2) + h(2) = 200 + 30 = 230$$

$$f(3) = g(3) + h(3) = 60 + 10 = 70$$

$$f(4) = g(4) + h(4) = 40 + 35 = 75$$

Por tanto, ABIERTA quedaría ordenada como (3 4 2). Con esto termina el ciclo 1, y se pasa al ciclo 2.

El ciclo 2, cuyo resultado es mostrado en la figura 29, expande el nodo 3, genera sus sucesores, nodos 2 y 5, mete en ABIERTA al nodo 5, que no estaba en el grafo, y establece punteros desde el nodo 5 al nodo 3 (pasos 5a, 5b, y 5c). No se crea un puntero desde el nodo 2 porque ya estaba en el grafo. A continuación, se estudia si redirigir el puntero desde el nodo 2 (que era el único sucesor que estaba en ABIERTA, según el paso 5d) al nodo 3. Se redirecciona el puntero si el camino desde el nodo raíz al nodo 2 pasando por el nodo 3 es más corto que el camino desde el nodo raíz al nodo 2 directamente. Es decir, se tiene que estudiar si el nuevo camino abierto hacia el nodo 2, por el nodo 3, es mejor que el que ya tenía. Esta decisión se toma mirando la suma de los costes de los caminos. Desde el nodo 2 al nodo 1 directamente el coste es de 200, mientras que a través del nodo 3 es de $60 + 150 = 210$. Por tanto, no interesa redirigir el puntero hacia el nodo 3. El paso 5e no es aplicable, ya que ningún sucesor del nodo 3 estaba en CERRADA, y se va al paso 5f, en el que se reordena ABIERTA. Como se hace conociendo el valor de $g(n)$ y $h(n)$ de cada nodo de ABIERTA, se recalculan los valores. Los nodos 2 y 4 no han cambiado de valores, con lo que

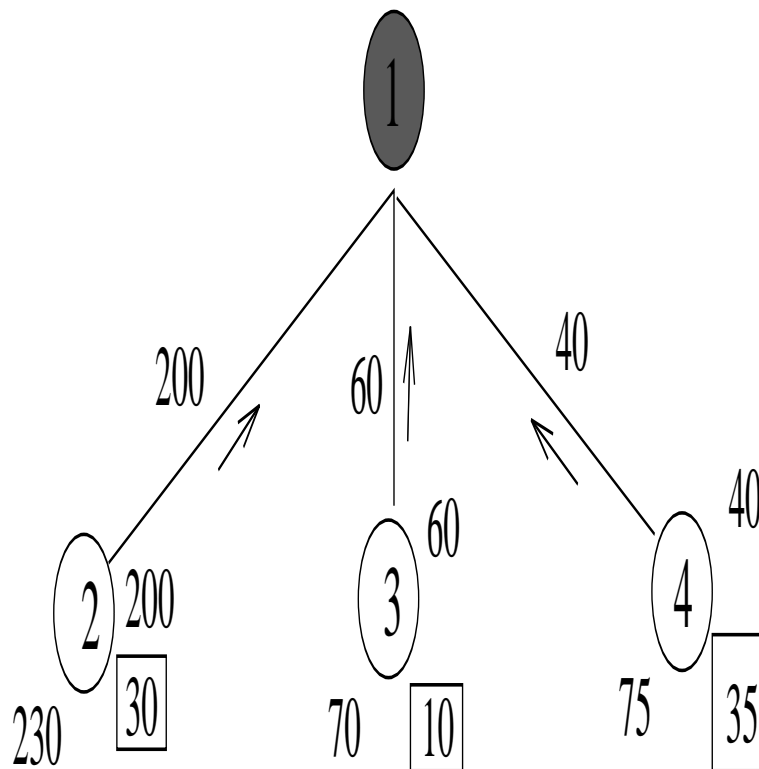


Figure 28: Primer ciclo del funcionamiento del A* para el ejemplo de la figura 27.

$f(2) = 230$ y $f(4) = 75$. El nodo 5 es nuevo y $f(5) = g(5) + h(5)$. $g(5)$ se calcula como la suma de los costes de los caminos al nodo 1 siguiendo los punteros (que indican el camino más corto). En este caso, $g(5) = 200 + 60 = 260$. Por otra parte, $h(5) = 0$ puesto que es el nodo meta, y, entonces, $f(5) = 260 + 0 = 260$. Al reordenar ABIERTA en el paso 5f, quedaría como (4 2 5). Como se puede observar, se ha introducido el nodo meta en el grafo de búsqueda, pero, sin embargo, todavía no se ha terminado el algoritmo, puesto que no es el mejor nodo de ABIERTA. Esto realmente significa que todavía no se ha encontrado el camino más corto desde el nodo inicial al nodo meta.

En el tercer ciclo, mostrado en la figura 30, se repiten los procesos anteriores, quitando el nodo 4 de ABIERTA, metiéndolo en CERRADA, generando sus sucesores (nodos 3 y 5). No establece ningún puntero, puesto que ya estaban en ABIERTA o CERRADA. En el paso 5d estudia si redirige o no los punteros de sus sucesores hacia él. En el caso del nodo 3, $g(3) = 60$, mientras que ahora puede ir por un camino de coste $40 + 5 = 45$. Por tanto, le conviene redirigir el puntero hacia el nodo 4. Igualmente ocurre en el nodo 5. Antes $g(5) = 260$, mientras que ahora puede ser $200 + 40 = 240$ a través del nodo 4. En el paso 5e, se estudia si redirigir o no el puntero de los sucesores de los nodos sucesores del nodo 4 que estuvieran en CERRADA (nodo 3), en este caso, el nodo 2 y el 5. Antes, $g(2) = 200$ mientras que, a través del nodo 3, sería $150 + 5 + 40 = 195$, por lo que sí le interesa redirigir el puntero hacia el nodo 3. El nuevo $g(5) = 240$ y, a través del nodo 3, sería $200 + 5 + 40 = 245$, por lo que no le interesa redirigir el puntero al nodo 3. Al final del ciclo, en el paso 5f, se reordena ABIERTA, donde $f(2) = 225$ y $f(5) = 240$. Por tanto, ABIERTA quedaría (2 5).

En el siguiente ciclo (figura 31), se quita el nodo 2 de ABIERTA, se mete en CERRADA, se expanden sus sucesores, nodos 5 y 6, y se crea un puntero desde el nodo 6 (único que no estaba en el grafo), pasos 5a, 5b, y 5c. En el paso 5d se estudia si redirigir o no el puntero desde el nodo 5 (que

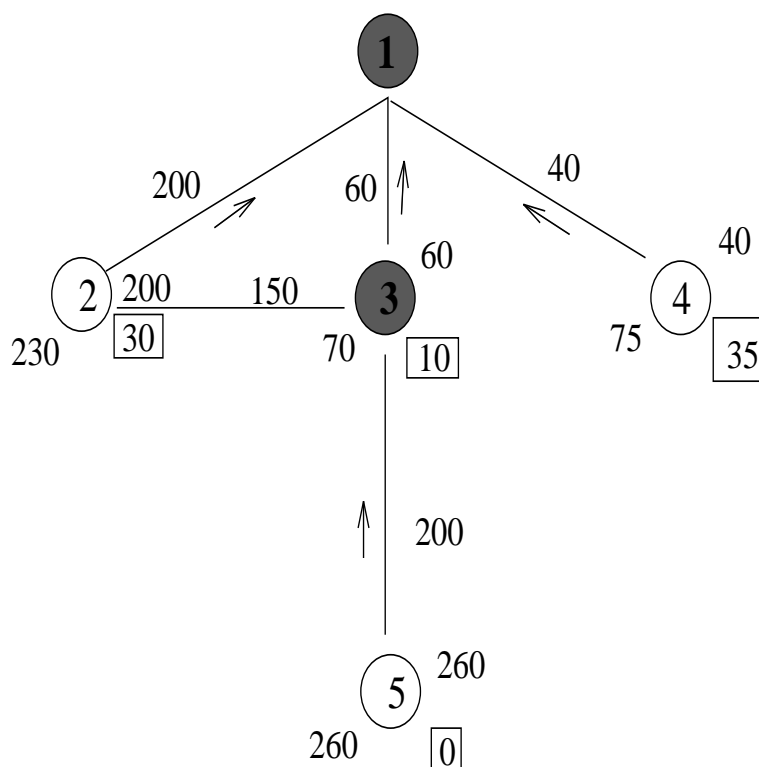


Figure 29: Segundo ciclo del funcionamiento del A* para el ejemplo de la figura 27.

estaba en ABIERTA) al nodo 2. Antes $g(5) = 240$ y, ahora sería $30+150+5+40=225$, por lo que sí le interesaría redirigir el puntero. El paso 5e no es aplicable, y en el paso 5f se reordena ABIERTA, sabiendo que $f(6) = g(6)+h(6) = (40+150+5+40)+40 = 275$ y $f(5) = g(5)+h(5) = 225+0 = 225$. Por tanto, al reordenar ABIERTA, quedaría (5 6). En el siguiente ciclo, al preguntar si el mejor nodo de ABIERTA, el nodo 5, es nodo meta, se terminaría el algoritmo, devolviendo como solución el camino desde el nodo raíz al nodo meta siguiendo los punteros: 1-4-3-2-5, que es el camino de coste mínimo. Se puede ver que $h(n) \leq h^*(n) \forall n$, y, por tanto, se cumple que el algoritmo es el A*.

15 Búsqueda en situaciones con contrincantes

Al tratar cierto tipo de problemas, cuyo paradigma son los juegos de antagonismo, es necesario utilizar unos procedimientos de representación y, posteriormente, búsqueda un poco distintos a los empleados hasta ahora. El desarrollo de este tipo de problemas se lleva a cabo por dos antagonistas que efectúan su acción cada uno alternativamente por turno. Este tema se va a centrar sobre aquellos problemas en los que todos los antagonistas conocen en cada momento perfectamente el estado de los otros antagonistas. A este tipo de problemas se les denomina de información completa. Ejemplos de este tipo son el ajedrez y las damas, mientras que el dominó y las cartas no lo son, debido a que se desconoce cuáles son las fichas/cartas del(os) contrario(s). Por tanto, se van a eliminar de esta unidad didáctica aquellos problemas cuyo resultado está determinado, aunque sea parcialmente, mediante el azar. Aunque las técnicas aquí descritas podrían generalizarse para incluirlas dentro de un esquema de solución de juegos de azar.

Más específicamente, se van a tratar problemas de suma nula; es decir, aquellos en los que lo

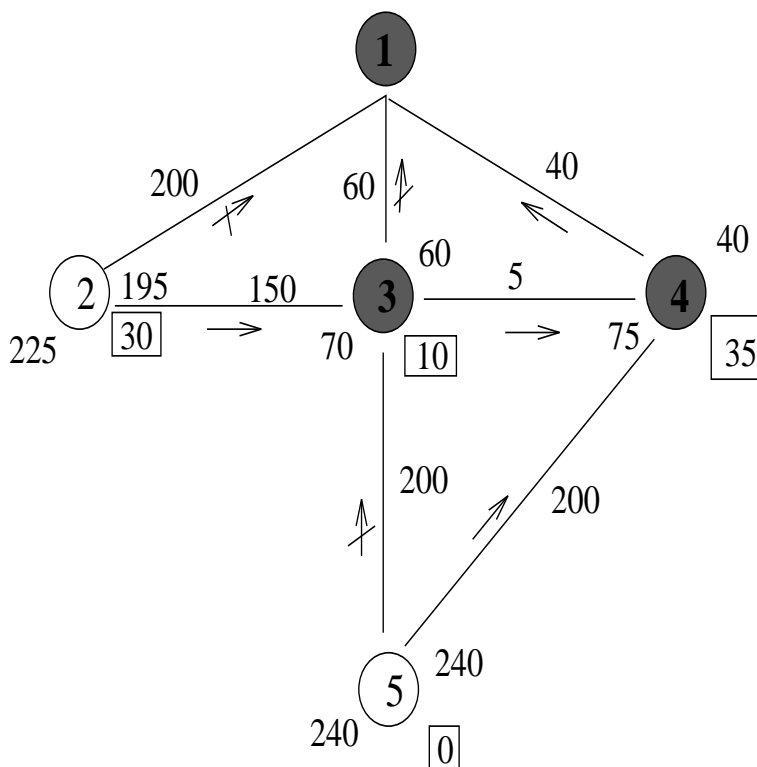


Figure 30: Tercer ciclo del funcionamiento del A* para el ejemplo de la figura 27.

que gana uno de los antagonistas, es justamente lo que pierde el otro, pudiendo darse una situación de equilibrio o empate entre ambos antagonistas. Juegos como las damas, el ajedrez, el “tres en raya”, el GO, el NIM y muchos otros, son ejemplos de este tipo de problema.

Para analizar y modelizar este tipo de problemas, se pueden utilizar sistemas que son, en el peor de los casos, muy similares a los sistemas de producción. Por ejemplo, en el ajedrez, la base de hechos global contendría una representación de la posición de todas las piezas sobre el tablero. Las reglas de producción modelizarían los movimientos legales del juego. La aplicación de estas reglas a la base de hechos inicial, y a sus sucesores, y a los sucesores de los sucesores, y así sucesivamente, genera lo que se denomina el grafo total de todas las posibles partidas, o el árbol concreto de una partida concreta.

15.1 Árboles alternados

La mejor forma de representar este tipo de problemas es mediante los, así denominados, árboles alternados. En efecto, las situaciones competitivas entre dos antagonistas, se caracterizan por un conjunto de posiciones y un conjunto de reglas para pasar de una posición a otra, “moviendo” alternativamente los adversarios. Estas situaciones, pueden representarse adecuadamente, mediante un árbol en el que cada nodo representa una posición. Los sucesores de un nodo proporcionan las posiciones a las que se puede acceder, usando el conjunto de reglas permitido, a partir de la posición que representa ese nodo. Este acceso se representa por arcos que unen ese nodo con sus sucesores. Cada nivel del árbol contiene las acciones posibles para uno de los antagonistas. Así, el antagonista A “moverá” desde posiciones de los niveles pares 0 (nodo raíz), 2, 4, ..., y el antagonista B lo hará desde posiciones correspondientes a niveles impares 1, 3, 5, Un árbol de este tipo es un árbol

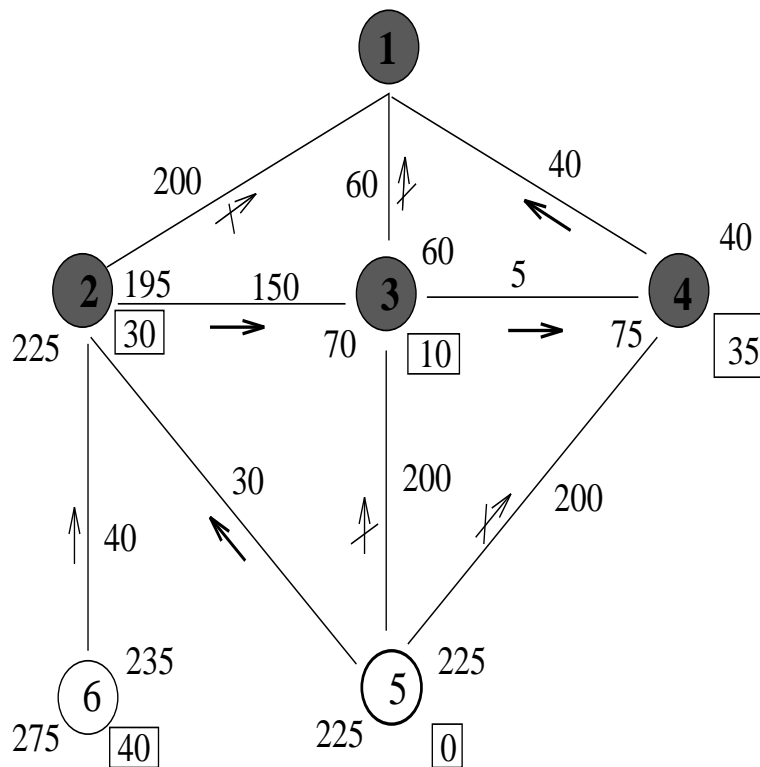


Figure 31: Cuarto ciclo del funcionamiento del A* para el ejemplo de la figura 27.

cuyos nodos son las distintas posiciones y los arcos o aristas las acciones posibles.

Obviamente, los árboles alternados presentan los defectos de los árboles libres y en particular la aparición de ciclos en forma de desarrollos periódicos ilimitados. Aunque en el caso de los juegos de dos personas los ciclos no sólo son raros, sino que están limitados de hecho por reglas “ad hoc”. De este modo, los árboles alternados se consideran el procedimiento idóneo para representar árboles de juegos.

La mayoría de los juegos sencillos, así como las secuencias finales de juegos complejos, se pueden representar mediante este tipo de árboles alternados habida cuenta de que en este contexto, la prueba de la ganancia, o empate, se puede hallar sin tener que generar un árbol de búsqueda muy profundo. El árbol solución que da la ganancia o el empate, proporciona entonces una estrategia de juego para uno de los dos jugadores.

Para juegos complejos, tales como el ajedrez o las damas, el árbol alternado de búsqueda, está absolutamente fuera de lugar. Si el juego de damas tiene alrededor de 10^{40} nodos, el de ajedrez es de 10^{120} nodos, lo que supondría tardar unos 10^{21} siglos para generar completamente el árbol, en el supuesto de que se tardara sólo $\frac{1}{3}$ de nanosegundo en generar cada sucesor, lo que, por otra parte, es mucho suponer. Por consiguiente, para juegos complejos se tiene que aceptar la imposibilidad de finalizar la búsqueda; es decir, se debe desechar la idea de demostrar que es posible obtener una victoria o tablas en el juego, excepto quizá para los finales de partida.

Para solucionar este problema se toman dos decisiones:

- Explorar sólo hasta una determinada profundidad, denominada profundidad máxima, p_{max} .
- Evaluar por medio de una función heurística, los nodos hoja del árbol obtenido, de modo que proporcione un valor estimativo que especifique la distancia desde el nodo correspondiente a

la meta, que es la victoria de la computadora.

A continuación, se presentan técnicas que permiten determinar cuál es la mejor decisión a tomar (mejor “jugada”) cuando se emplea esta técnica mixta de búsqueda y evaluación. Se estudiará el algoritmo básico, minimax, seguido de un método más eficiente de encontrar la misma solución que el minimax, el método de poda Alfa-Beta.

15.2 Método minimax para árboles alternados

El objetivo de la búsqueda en el árbol del juego, debe ser encontrar un “buen” primer movimiento, haciendo dicho movimiento “in mente” sin que el contrincante dé la réplica, y buscar de nuevo un buen primer movimiento desde esta nueva situación. Para cada una de estas búsquedas, se puede utilizar cualquiera de los procedimientos ya vistos como profundidad, amplitud, o escalada, teniendo en cuenta que ahora las condiciones de finalización deben ser modificadas. Entre estas condiciones de finalización, se pueden introducir algunas artificiales; es decir, ajenas al juego como tal. Estas condiciones artificiales pueden basarse en factores tales como límite de tiempo, espacio de almacenamiento en memoria, profundidad del nodo terminal más profundo del árbol de búsqueda, y otros por el estilo. Por otra parte, es habitual no concluir, por ejemplo en ajedrez, si cualquiera de los nodos es una posición “viva”; esto es, posiciones en las cuales no se vislumbra una ventaja inmediata de alguno de los antagonistas, como se estudiará más adelante.

Una vez concluido el árbol de búsqueda (subárbol del árbol total de juego) se debe extraer un estimado del “mejor” primer movimiento. Este estimado puede obtenerse aplicando una función de evaluación estática, a las “hojas” del árbol de búsqueda. La función de evaluación mide el valor de la posición del nodo terminal, y se basa en ciertas características que racionalmente se presume que influyen en dicho valor. Es habitual convenir que las situaciones favorables a A tengan un valor positivo de la función de evaluación, mientras que las favorables a B tengan un valor negativo. Valores próximos a cero se reservan para las posiciones dudosas.

Una vez dicho lo anterior, un buen primer movimiento, puede obtenerse por un procedimiento llamado del Minimax. Se expande el árbol de búsqueda hasta llegar a los nodos que estén en una profundidad fijada como máxima, p_{max} . En ese momento, se evalúan todos los nodos “hoja” de ese árbol. Si se supone que es A quien elige entre los nodos terminales, se inclinará ante varias alternativas por el nodo mejor para él/ella, es decir, aquél de mayor evaluación. Por lo tanto, al nodo antecedente de las hojas se le asignaría un valor igual al mayor valor de las evaluaciones de las hojas. Si fue B el que eligió entre los terminales, y se supone que el contrario elegirá su mejor movimiento, éste será el de menor evaluación. Por consiguiente, al nodo antecedente de nodos terminales, se le asignaría un valor igual al mínimo de las evaluaciones de los terminales.

Una vez que a todos los predecesores de los nodos terminales se le han asignado esos valores “de retroceso”, se continúa hacia la raíz del árbol, dando valores “de retroceso” a otro nivel, suponiendo que A elegiría el nodo con menor valor, y así sucesivamente, nivel a nivel, hasta que finalmente, en el nodo inicial, A elegirá como mejor primer movimiento el correspondiente al sucesor de mayor valor. La utilidad de este método se basa en la presunción de que los valores de retroceso asignados al nodo inicial son medidas de más garantía del último valor relativo de esas situaciones, que los valores que se obtendrían aplicando directamente la función de evaluación estática a esas posiciones. Los “valores de retroceso” están, después de todo, basados, por una parte, en una visión de futuro del árbol del juego y, consiguientemente, dependen de circunstancias o características que sólo se ponen en evidencia al final del juego. Por otra parte, están basados en la suposición “Radzsky” quien decía: “no consideres a tu enemigo más estúpido de lo que tú te creas”.

El adversario de números positivos (que hemos llamado A) se denomina “maximizante” o MAX. Su contrincante, el de números negativos, “minimizante” o MIN. El maximizante buscará caminos que le lleven a números positivos altos, sabiendo que su adversario intentará forzar el juego a situaciones con evaluaciones estáticas muy negativas.

En el árbol de la figura 32, el maximizante intentará alcanzar una posición con un valor estático de 8. Pero el maximizador sabe que el minimizador no se lo permitirá, ya que el minimizador puede escoger un movimiento que desvíe su acción hacia la posición con un valor de 1 (figura 33). Por lo general, la decisión del maximizador debe tener en cuenta la actitud del minimizador en el nivel siguiente. Si se adelanta la búsqueda un paso más, el minimizador actúa entonces, con toda seguridad, de acuerdo con las opciones del maximizador en el nivel siguiente. Esto continúa hasta que se alcanza el límite de la exploración y el evaluador estático provee una base directa para seleccionar entre las distintas alternativas. En el ejemplo, las evaluaciones estáticas en la parte inferior determinan que las opciones disponibles para el jugador minimizante conducen a valores efectivos de 2 y 1 en el nivel justo encima (figura 33). Conociendo estos valores efectivos, el maximizador puede determinar la mejor acción en el siguiente nivel superior. Claramente, el maximizador se mueve hacia el nodo desde el cual el minimizador no puede hacer nada mejor que mantener el valor esperado de 2 (figura 34). De nuevo, los valores en un nivel determinan la acción y el valor efectivo en el siguiente nivel superior.

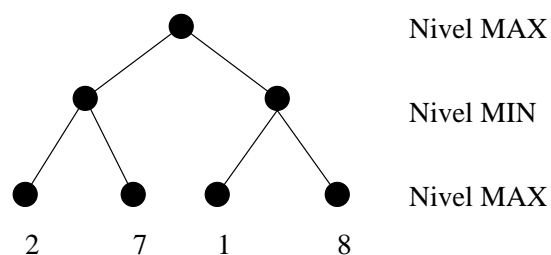


Figure 32: Niveles maximizante y minimizante en un árbol de representación de un problema competitivo.

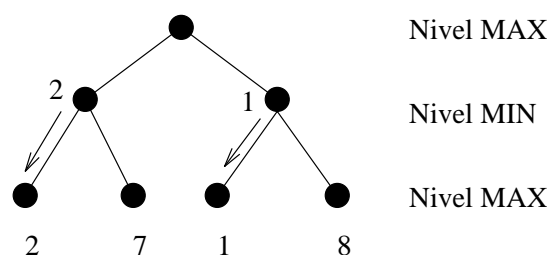


Figure 33: Niveles maximizante y minimizante en un árbol de representación de un problema competitivo.

Para ilustrar este procedimiento de minimax, se va a utilizar el juego conocido por los nombres de “Tres en raya” o “Cruces y círculos” o “Tic-tac-toe” en diferentes versiones. La descripción del juego, tal y como se va a usar aquí, puesto que hay muchas variantes, es la siguiente. El juego se desarrolla en un tablero de 3x3 casillas en las que los jugadores, por turno, van depositando sus fichas. Uno juega con cruces (x) y el otro con círculos (o). Gana el primero que consigue colocar tres de sus fichas en línea ya sea horizontal, vertical o diagonalmente. Si se han colocado todas las

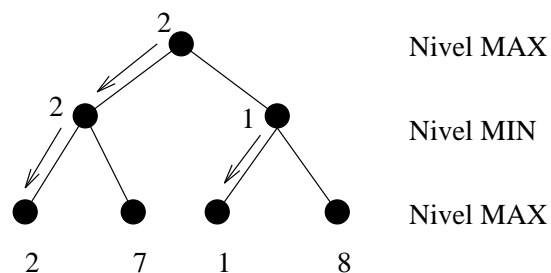


Figure 34: Niveles maximizante y minimizante en un árbol de representación de un problema competitivo.

fichas, llenando el tablero, y no hay ningún ganador, han empatado. Supóngase que A juega con x y B con o. Se desarrolla una búsqueda por niveles hasta que se generan los nodos de nivel p y, entonces, se aplica una función de evaluación estática a las posiciones de esos nodos. En lo que sigue se utilizará una función de evaluación, $f_{ev}(s)$, para una situación s , que viene dada por las tres posibilidades siguientes:

1. Si s es ganadora para A,
 $f_{ev}(s) = \infty$ (∞ denota el mayor número positivo posible).
2. Si s es ganadora para B,
 $f_{ev}(s) = -\infty$.
3. Si s no es ganadora para cualquiera de los jugadores,
 $f_{ev}(s) = \text{abiertas(A)} - \text{abiertas(B)}$

donde abiertas(X) es el número de filas, columnas o diagonales en las que el jugador X puede ganar; es decir, el número de líneas que no contienen una ficha del contrario.

Así, si s es la mostrada en la figura 35, se tiene que $f_{ev}(s) = 6 - 4 = 2$.

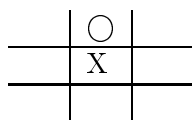


Figure 35: Ejemplo de situación en el juego del “tic-tac-toe”.

El procedimiento mediante el cual la información del valor atraviesa de abajo a arriba el árbol del problema se denomina MINIMAX, ya que la puntuación de cada nodo es o bien la máxima, o la mínima de los valores en los nodos inmediatamente inferiores. Este procedimiento se muestra en la tabla adjunta.

Procedimiento Minimax (Situación Profundidad)

SI Profundidad = p_{max}
 ENTONCES devolver evaluación (Situación)

SI NO SI ganadora (Situación)
 ENTONCES devolver $+\infty$

SI NO SI perdedora (Situación)
 ENTONCES devolver $-\infty$

SI NO SI empate (Situación)
 ENTONCES devolver 0

SI NO
 S = sucesores (Situación)
 L = lista de llamadas al MINIMAX ($S_i \in S$, Profundidad + 1)
 SI nivel-max (Profundidad)
 ENTONCES devolver max (L)
 SI NO devolver min (L)

Puesto de otra manera, el valor numérico de cada nodo del árbol o valor minimax de un nodo es:

$$f(n) = \begin{cases} +\infty & \text{si } n \text{ es una situación ganadora para computadora} \\ -\infty & \text{si } n \text{ es una situación perdedora para computadora} \\ 0 & \text{si } n \text{ es una situación de empate} \\ f_{ev}(n) & \text{si } p = \text{Profundidad máxima} \\ \max_{S_i \in S(n)} f(S_i) & \text{si } n \text{ es nodo max y } p < p_{max} \\ \min_{S_i \in S(n)} f(S_i) & \text{si } n \text{ es nodo min y } p < p_{max} \end{cases}$$

Hay que tener en cuenta que toda la idea de minimización se basa en la traducción de la situación del problema en un sólo número, el valor estático. Esto lleva aparejado un grave problema: un número no dice nada sobre cómo fue determinado. Además, la minimización puede resultar cara, dependiendo del evaluador estático y del generador de movimientos utilizados.

En las figuras 36, 37, y 38, se muestra el árbol generado por una búsqueda a nivel 2 ($p_{max} = 2$). Las evaluaciones estáticas se dan debajo de los nodos hoja y los “valores de retroceso” se presentan a la izquierda de los nodos. Nótese que no aparecen todos los sucesores del nodo raíz, ya que los que faltan son simétricos respecto a los que se han expandido, y, por tanto, devolverían el mismo valor que su simétrico devuelve. Si estuviera ejecutándose en una computadora, se deberían expandir estos nodos, a no ser que el algoritmo de generación de nodos no generara nodos correspondientes a situaciones simétricas. El algoritmo minimax devolvería, en este caso, la primera jugada como mejor jugada a realizar de acuerdo a la profundidad máxima seleccionada y al orden en el que se han generado los sucesores de cada nodo.

Si s es una situación desde la cual hay d movimientos legales s_1, \dots, s_d ($d > 1$), el problema es elegir la mejor acción. Se supone que la mejor acción es la que logre el valor más grande posible cuando la situación competitiva finalice, si el oponente también elige acciones que son las mejores

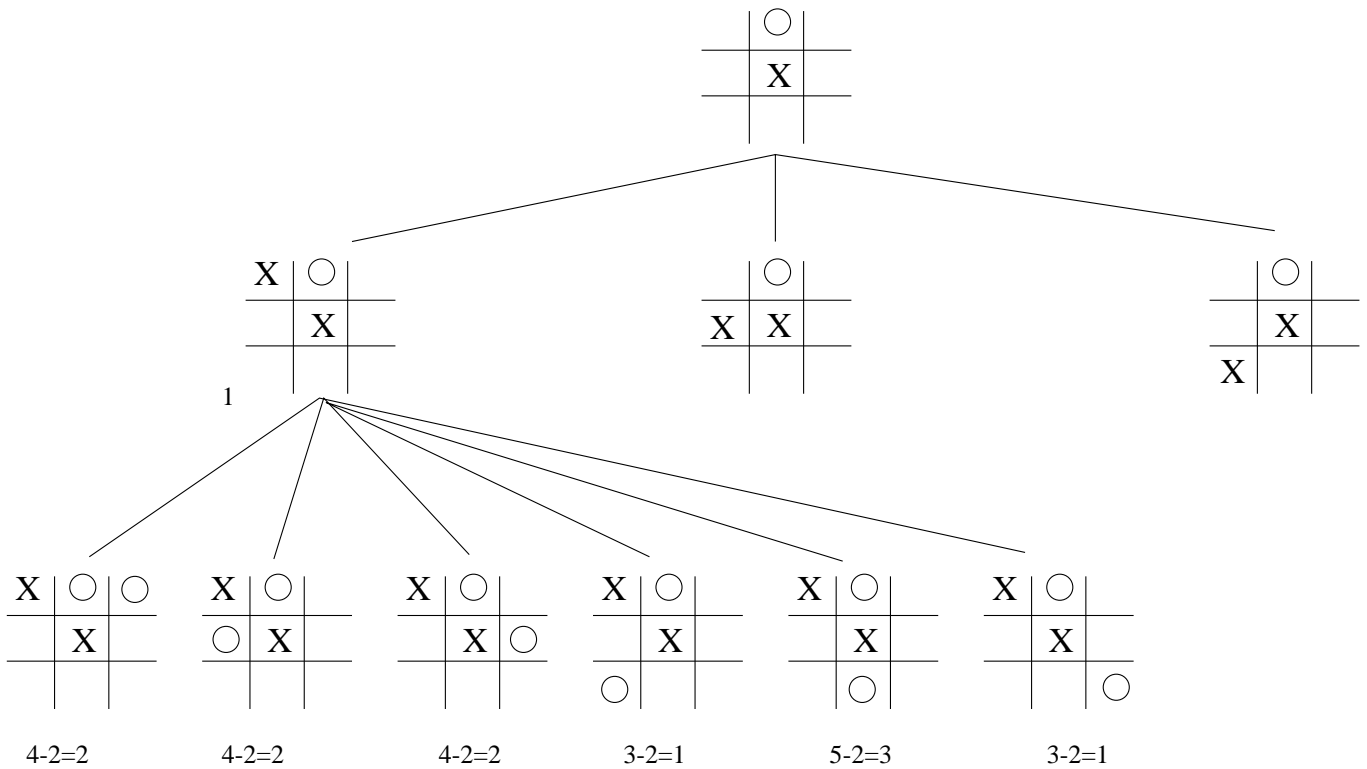


Figure 36: Ejemplo del Minimax aplicado al Tres en raya. Expansión del primer nodo sucesor del nodo raíz.

para él. Supóngase que $E(s)$ es el mayor valor posible que puede conseguirse desde la situación s contra la estrategia defensiva óptima. Dado que el valor, para el individuo que está en turno de acción desde s , después de mover a la posición s será $-E(s)$, se tiene

$$E(s) = \begin{cases} e(s) & sid = 0 \\ \max(-E(s_1), \dots, -E(s_d)) & sid > 0 \end{cases}$$

Este formalismo es equivalente al minimax y se denomina negamax.

Ambas formulaciones, “negamax” y “minimax”, son equivalentes, pero, en ocasiones, es más fácil razonar sobre el juego usando el esqueleto de trabajo “minimax”. La razón es que, a veces, es más difícil confundirse si continuamente se evalúan las posiciones del modelo competitivo desde el punto de vista de un participante. Por otra parte, la formulación “negamax” tiene ventajas cuando se intenta probar cosas acerca de las situaciones competitivas, porque no se tiene que trabajar con dos casos separados.

La función $E(s)$ es el valor máximo final que puede conseguirse si ambos adversarios actúan óptimamente, pero hay que señalar que esto refleja una estrategia bastante conservadora que no siempre será la mejor contra adversarios malos o no óptimos que se pueden encontrar en el mundo real. Por ejemplo, supóngase que hay dos acciones, posiciones s_1 y s_2 , donde s_1 asegura un empate (valor 0) pero no puede ganar, mientras s_2 da una oportunidad de victoria o derrota dependiendo de si el oponente juega o no óptimamente. Puede estar mejor realizada la acción s_2 , que es la única oportunidad de ganar, a menos que se esté convencido de la competencia del oponente. En la terminología de situaciones competitivas, una acción s_2 puede ser “refutada” si el oponente puede hacer una réplica a s_2 que es al menos tan buena como su mejor réplica a s_1 . Esta técnica de

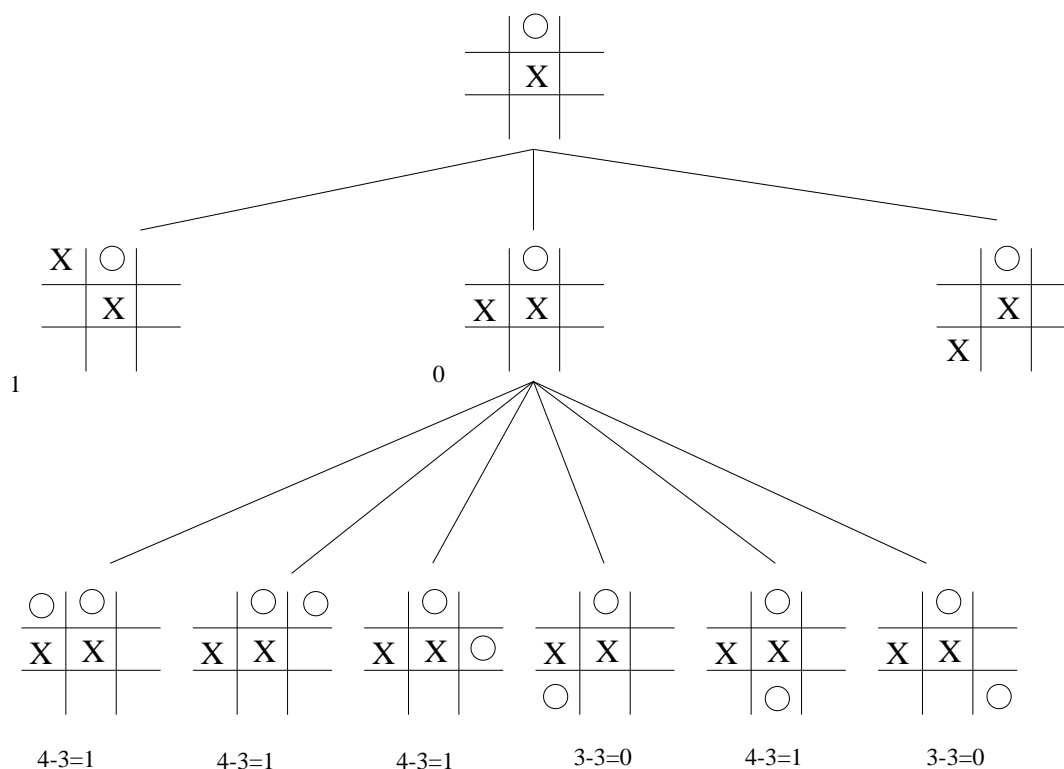


Figure 37: Ejemplo del Minimax aplicado al Tres en raya. Expansión del segundo sucesor del nodo raíz.

razonamiento lleva a una técnica computacional que evita mucha de la computación hecha por el minimax.

Aunque el minimax resuelve el problema de encontrar cuál es la mejor decisión a tomar dado un conjunto de posibles acciones y contracciones del contrario, realiza una búsqueda denominada de “fuerza bruta”, debido a que expande y estudia todos los sucesores de cada nodo hasta los nodos de la profundidad máxima fijada. Es posible mejorar esta “búsqueda” de “fuerza bruta” usando la técnica Alfa-Beta, ignorando acciones que son incapaces de ser mejores que otras que ya se conocen.

15.3 Método Alfa-Beta: un procedimiento de poda

15.3.1 Introducción

El primer estudio publicado de un método de poda de árboles representando problemas competitivos, lo hicieron Newel, Shaw y Simon; pero, no estaba muy claro que emplearan los “atajos”. McCarthy fue el primero en darse cuenta de la potencialidad de esta poda, quien acuñó el nombre de Alfa-Beta, debido a los límites que usa el algoritmo, y el primero que escribió un programa en LISP que contenía esta técnica, llegando a pensar que era un dispositivo heurístico posiblemente inexacto. Esto no era así en absoluto, puesto que se trata de un sencillo procedimiento algorítmico, basado en el método de separación y evaluación progresiva, “branch-and-bound”, que produce los mismos valores que el minimax.

La técnica Alfa-Beta presenta tres características que lo hacen imprescindible en Inteligencia Artificial.

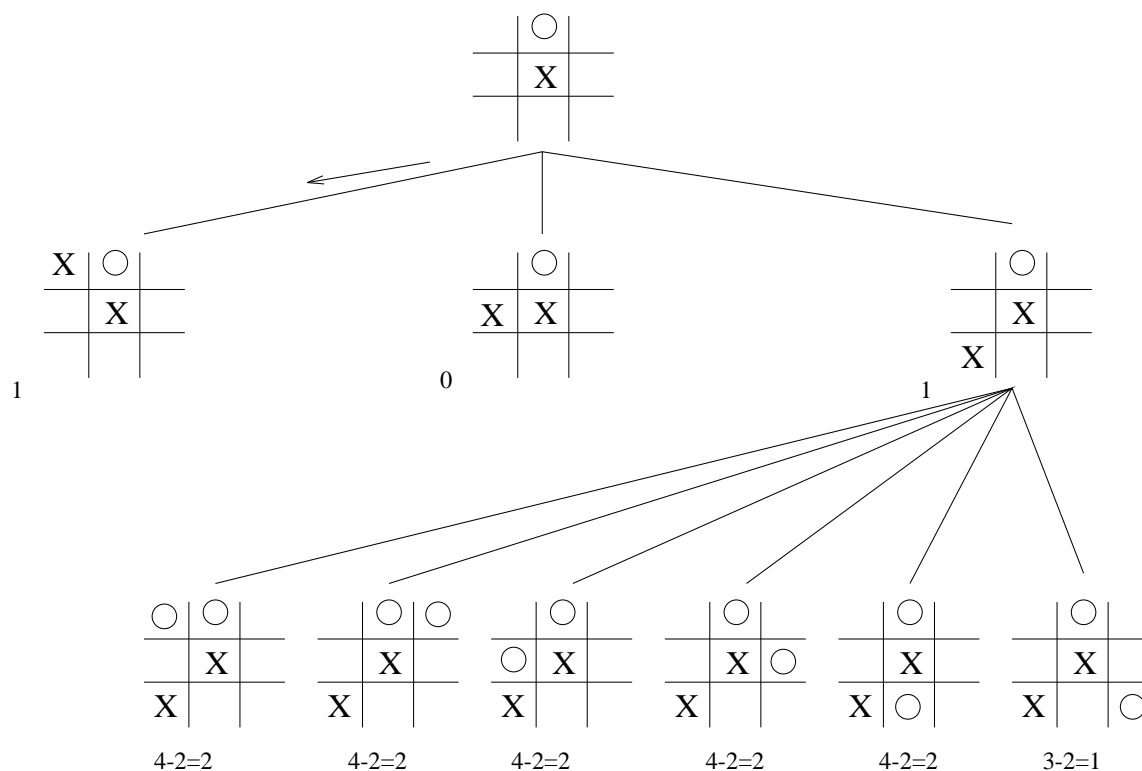


Figure 38: Ejemplo del Minimax aplicado al Tres en raya. Expansión del tercer sucesor del nodo raíz, y selección de la jugada a realizar.

- En primer lugar, complementa y suple el método de “minimax”, puesto que permite a la computadora soslayar la generación de secuencias irrelevantes de movimientos en la búsqueda “minimax”. Más en concreto, Alfa-Beta le indica al “minimax” dónde debe detener la investigación de los sucesores de una acción o de una secuencia de acciones dadas, siempre y cuando uno de los sucesores que se encuentre, sea una acción de “refutación”.
- En segundo término, como se sabe, la inmensa mayoría de los métodos de búsqueda, separan completamente el proceso de generación del árbol de búsqueda, del proceso de evaluación de cada posición, de tal forma que únicamente una vez que se completa la generación del árbol, es cuando se inicia la evaluación de los nodos. Esta dicotomía da como resultado una estrategia muy poco eficiente. Esta ineficiencia se obvia si se ejecuta la evaluación de las hojas del árbol, calculando, simultáneamente, hacia atrás, los valores de los nodos, con la generación del árbol, obteniéndose una reducción notable, a veces de varios órdenes de magnitud, en el total de la búsqueda necesaria para conseguir el mismo mejor movimiento.
- Por último, esta técnica parece bastante difícil de comunicar verbalmente o en lenguaje matemático convencional. Además, parece necesitarse bastante pensamiento para convencerse de que el método es correcto, en especial para justificar en lenguaje corriente los “atajos”. Sin embargo, el método es fácilmente comprensible y se demuestra correcto en lenguaje algorítmico. De este modo, es un buen ejemplo de cómo, a veces, un enfoque “dinámico” de la descripción de un método es conceptualmente superior al enfoque “estático” de las matemáticas convencionales.

15.3.2 El método de poda

La poda Alfa-Beta se basa en la idea de disponer de dos valores que conformen una ventana a la cual deben pertenecer los valores de $f_{ev}(n)$ para que sean considerados. En los nodos n MAX, según el algoritmo minimax, se debe maximizar el valor de los nodos sucesores. En estos nodos, se utiliza el parámetro $\alpha(n)$ que determina el máximo de los valores de los nodos sucesores encontrado hasta el momento. Asimismo, como los nodos n MIN tratan de minimizar el valor de los nodos, utilizan el parámetro $\beta(n)$ que va a ser, en cada momento, el mínimo de los valores encontrados de los nodos sucesores de ese nodo.

Existen dos formas de podar, dependiendo del nodo en el que se esté estudiando. En los nodos MAX, la condición de poda es:

$$\alpha_p \geq \beta_{p-1}$$

es decir, si el α de un nodo MAX de profundidad p es mayor o igual que el β del nodo MIN padre (profundidad $p - 1$) se pueden podar los sucesores del nodo MAX no estudiados todavía. Esto es debido a que, como valor inferior, el nodo MAX va a devolver ese valor de α (α_p). Ya que el nodo superior trata de minimizar, va a calcular:

$$\beta_{p-1} = \begin{cases} \beta_{p-1} & \text{si } \alpha_p \geq \beta_{p-1} \\ \alpha_p & \text{si } \alpha_p < \beta_{p-1} \end{cases}$$

con lo que siempre va a elegir el β_{p-1} . Por lo tanto, los nodos no estudiados todavía en el nodo MAX no es necesario estudiarlos, porque no cambian el valor β del nodo padre.

Simétricamente, la condición de poda en los nodos MIN es:

$$\beta_p \leq \alpha_{p-1}$$

La explicación es análoga al caso anterior. El algoritmo recursivo del Alfa-Beta podría especificarse de la forma expresada en la tabla adjunta, donde la llamada inicial es de la forma:

Alfa-Beta (Situación-Inicial Menor-Número Mayor-Número 0)

Procedimiento Alfa-Beta (Situación Alfa Beta Profundidad)

SI Situación está considerada como empate, Devolver 0

SI Situación es ganadora, Devolver mayor-numero

SI Situación es perdedora, Devolver menor-numero

SI Profundidad = Profundidad-maxima, Devolver evaluacion (Situación)

SI nivel-max-p

para todo s_i sucesor de Situación y Alfa < Beta

Alfa-beta = ALFA-BETA(s_i ,Alfa,Beta,Profundidad+1)

Alfa = max(Alfa,Alfa-beta)

Devolver Alfa

SI NO

para todo s_i sucesor de Situación y Beta > Alfa

Alfa-beta = ALFA-BETA(s_i ,Alfa,Beta,Profundidad+1)

Beta = min(Beta,Alfa-beta)

Devolver Beta

Ejemplo:

Con el árbol de la figura 39, va a realizarse una búsqueda minimax con la poda alfa-beta. En este árbol los nodos círculo son nodos de nivel MAX, y, por tanto, se les denomina nodos MAX, y los nodos cuadrado pertenecen a niveles MIN, y, por tanto, se les denomina nodos MIN. Dentro de cada nodo aparece su identificador y debajo de los nodos hoja aparece el valor que devolvería la función de evaluación si se evaluara dicho nodo.

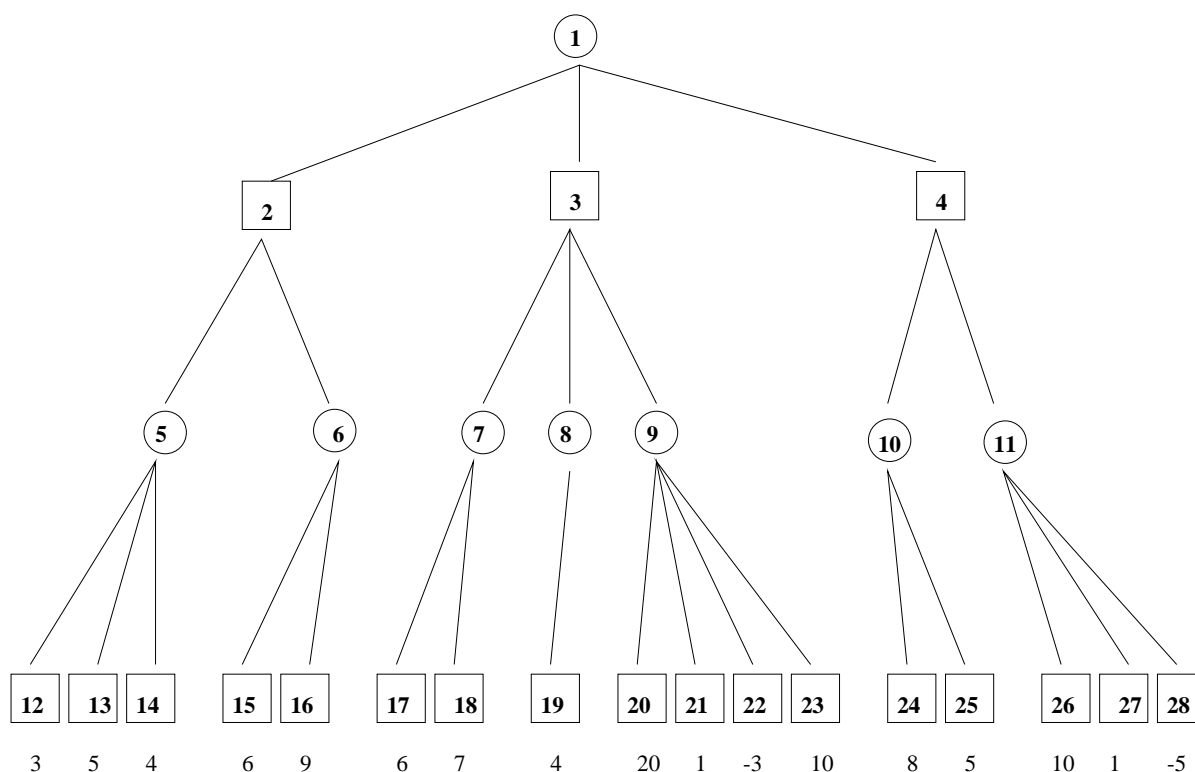


Figure 39: Arbol completo hasta la profundidad máxima. Este árbol muestra los valores de las hojas.

El algoritmo se inicia con el nodo inicial y los valores para α y β siguientes, teniendo en cuenta que no se puede representar el ∞ en las computadoras:

α = menor número posible (m); β = mayor número posible (M)

Los valores de m y M dependen del lenguaje de programación, y de la computadora utilizada, normalmente. Como puede observarse en la figura 40, se produce una poda en el nodo 6 ya que el α en ese nodo (6) es mayor que el valor de β (5) en el nodo padre. En la figura 41, puede observarse cómo se produce una poda en el nodo 3 debido a que el valor de β (5) es igual al de α (5) del nivel superior, con lo que, por mucho que se estudien los hijos del nodo 3 no se va a cambiar el valor del nodo 1. Por último, en la figura 42, vuelve a producirse una poda debido a que, en el nodo 11 el valor de α (10) es mayor que el valor de β (8) en el nodo padre.

La técnica Alfa-Beta es completamente general, pudiendo utilizarse tanto para árboles de cualquier profundidad finita, como para árboles de profundidad irregular. Las interrupciones que

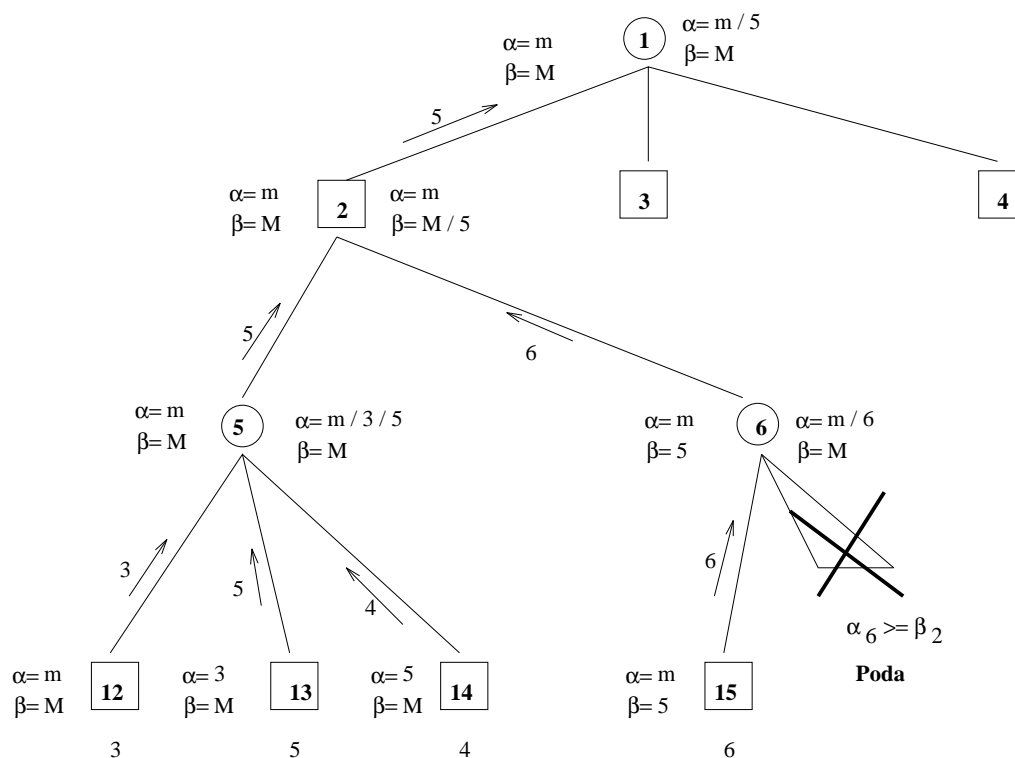


Figure 40: Estudio de la parte izquierda del árbol de la figura 39.

se producen en los nodos conjuntivos se llaman “alfa-interrupciones”, diciendose también que existe un “alfa-atajo” en tanto que las interrupciones que se producen en los nodos disyuntivos se llaman “beta-interrupciones”, o también se dice que existe un “beta-atajo”.

15.3.3 Mejoras sobre el Alfa-Beta

Desde que se dio a conocer el Alfa-Beta como método de poda han surgido muchos estudios, críticas, mejoras, e innovaciones a dicho método. Algunas son:

Efecto horizonte: El descubrimiento de este problema se debe a Berliner y radica en el estudio del Alfa-Beta hasta una profundidad fija. Supóngase, en ajedrez, que se captura con la reina de la computadora, en una jugada que da lugar a un nodo hoja (profundidad máxima), a la reina enemiga, que estaba defendida por un peón. La situación expresada por el nodo hoja devolverá un valor minimax muy alto puesto que se dispone de una reina y el contrario no. Sin embargo, esta situación es ficticia ya que, en la siguiente jugada, el peón enemigo capturará la reina.

El efecto horizonte está provocado, pues, por el estudio a profundidad fija, debido a que no se “ve” qué es lo que ocurre más allá del horizonte establecido. La solución que se da normalmente a este problema consiste en realizar un estudio de profundidad variable o búsqueda secundaria como se tratará más adelante.

Equilibrio: Este problema está relacionado con el anterior puesto que el efecto horizonte

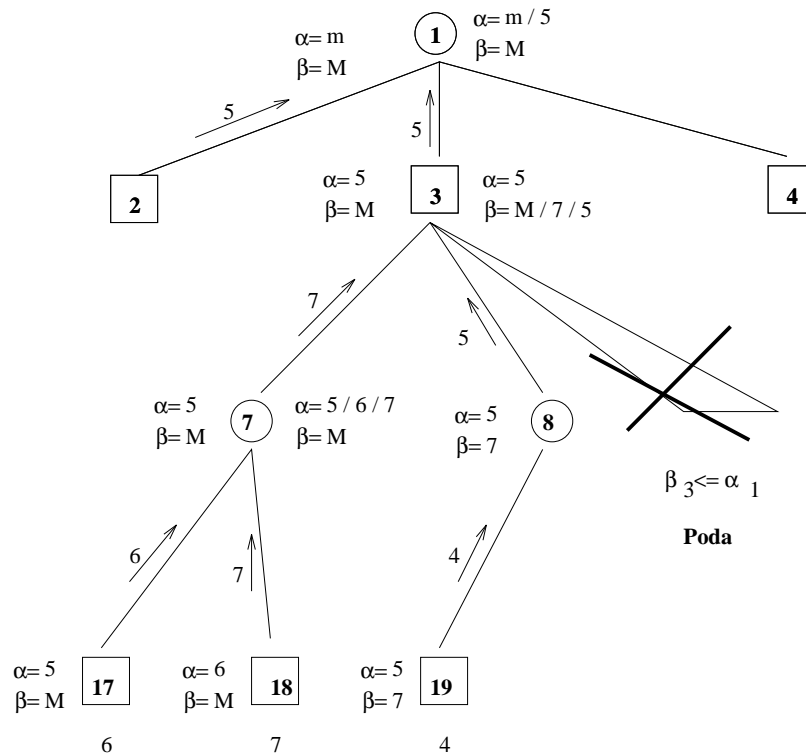


Figure 41: Continuación de búsqueda de la figura 40.

provoca falta de equilibrio. Supongamos que se tiene el árbol de la figura 43. Puede observarse que el nodo de la izquierda devuelve un valor de 8. Si se siguiera explorando un nivel más, se encontraría que devolvería un valor de -20 (figura 44). Por lo tanto, este valor no es estable ya que fluctúa mucho. La solución a la falta de equilibrio es también la búsqueda secundaria.

Búsqueda secundaria o de profundidad variable: En este tipo de búsqueda, no se estudia el árbol hasta una profundidad fija sino que se varía la profundidad de búsqueda. A los nodos en los que se estudia a más profundidad que a la profundidad máxima se les denomina “extensiones selectivas” y representan casos forzados. La forma de saber cuándo una situación es un caso forzado suele ser dependiente del dominio. Por ejemplo, en el ajedrez se puede estudiar por debajo de aquellos nodos en los que se capture una pieza. Esto presenta problemas como la dificultad para proporcionar todos o casi todos los casos interesantes, o representar información estática y no dinámica.

Otra solución consiste en utilizar información independiente del dominio como son las “extensiones singulares”. Una jugada es singular si devuelve un valor mucho mayor que sus jugadas hermanas (con el mismo padre). Es decir, si $n \in \text{suc}(n')$ y

$$f(n) \gg \max_{n'' \in \{\text{suc}(n') - n\}} f(n'')$$

se dice que n es un nodo singular. En esos nodos es donde se realizará una búsqueda secundaria. La búsqueda secundaria sólo expande aquellos nodos por debajo del nodo interesante que comporten

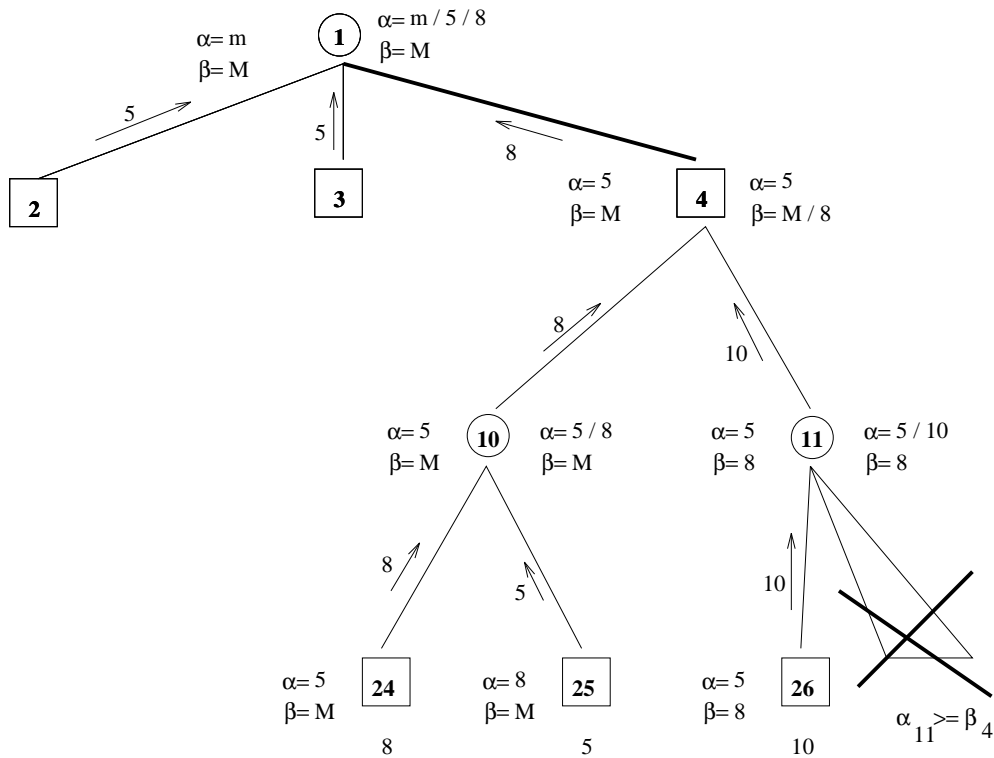


Figure 42: Continuación de búsqueda de la figura 41.

algo importante a la partida como, en ajedrez, nodos relacionados con capturas y, o, jaques.

Profundizamiento iterativo o progresivo: Slate y Atkin proponen este método que es muy utilizado en juegos en los que el tiempo tiene un papel importante de restricción, como en ajedrez, o cuando interesa que el programa determine qué jugada hacer en un determinado tiempo. El método consiste en estudiar hasta una determinada profundidad p . Si todavía queda tiempo, se estudian k niveles más (profundidad $p + k$). Así sucesivamente hasta que ya no quede más tiempo. Aunque pueda parecer que se pierde mucho tiempo cada vez que se baja un nivel, por tener que recalcular el árbol, se ha demostrado que no hay tal pérdida de tiempo debido a la ordenación de los nodos resultante de las iteraciones anteriores. El tema de la ordenación de los nodos aparece tratado más adelante.

Movimiento nulo: Goetsch y Campbell lo plantean como un método para ayudar a la poda del árbol Alfa-Beta. En los nodos MAX, se evalúa la situación de ese nodo o, lo que es lo mismo, se evalúa la situación correspondiente a no mover. Si esa situación pertenece al grupo de las situaciones en las que no mover representa la mejor jugada, entonces representa un límite inferior de lo que va a devolver ese nodo, puesto que va a ser la peor jugada. Por tanto, si ese valor es mayor o igual que el β del nodo padre, se puede podar sin estudiar ninguno de sus hermanos. Al tipo de

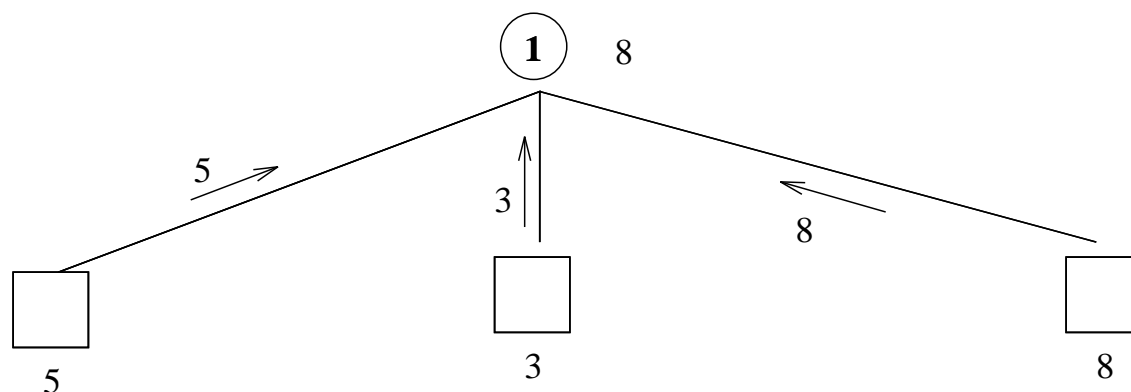


Figure 43: Ejemplo de árbol de búsqueda

situaciones en las que mover es la peor jugada se les denomina, en el argot, posiciones “zugzwang”.

Movimiento asesino: Este método es muy utilizado para podar más nodos. Si en un determinado nodo s_i , se han estudiado sus sucesores y el mejor sucesor ha sido s_{ik} , en el siguiente nodo hermano de s_i , s_{i+1} , se intenta estudiar primero la jugada correspondiente a la s_{ik} . Si fue la mejor en el nodo hermano, la heurística dice que es muy posible que también sea la mejor en el nodo $s_i + 1$. Este método fue planteado por Huberman.

Ventana: En lugar de empezar con la ventana inicial ($a=-\infty$, $b=+\infty$), puede comenzarse con una ventana más pequeña (a , b). Esto hace que se produzca mayor número de podas, con lo que el sistema puede profundizar más. Hay que tener cuidado, sin embargo, puesto que si la ventana es demasiado pequeña se pueden estar desechando ramas importantes para el resultado final.

Ordenación de nodos en el Alfa-Beta: Slagel y Dixon, en un trabajo publicado en 1969 introdujeron el concepto de orden de generación/estudio de nodos que puede mejorar la eficiencia de la técnica Alfa-Beta. Esta idea sugiere la posibilidad de mejorar la eficiencia del procedimiento Alfa-Beta mediante la ordenación de los sucesores de una posición por su valor estático, para producir el mayor número de cortes “alfa” y “beta”. Así, si los sucesores de mayor valor, en una posición “MAX”, y los peores sucesores, en una posición “MIN”, se ponen delante de los demás, se conseguirán cuanto antes todas las podas posibles. Esto está basado en el hecho de que las podas se conseguirán cuanto antes se alcancen valores altos/bajos para los α/β . Para ordenar los sucesores de mejor a peor se puede utilizar una función de evaluación más sencilla que la que se utiliza normalmente en los nodos hojas que proporcione una primera indicación de cómo son de buenos/malos los nodos sucesores de cada nodo.

Se puede demostrar que el número de hojas de un árbol de profundidad p generado por una búsqueda óptima Alfa-Beta es aproximadamente igual al número de hojas que se habrían generado, sin el método Alfa-Beta, a la profundidad $p/2$. Por consiguiente, para la misma capacidad de almacenamiento, el método Alfa-Beta con sucesores en el orden perfecto permitiría doblar la profundidad de búsqueda. Como es lógico, el orden perfecto es imposible de lograr, dado que si fuera posible, la búsqueda sería absolutamente innecesaria; sin embargo, se debe buscar la mejor

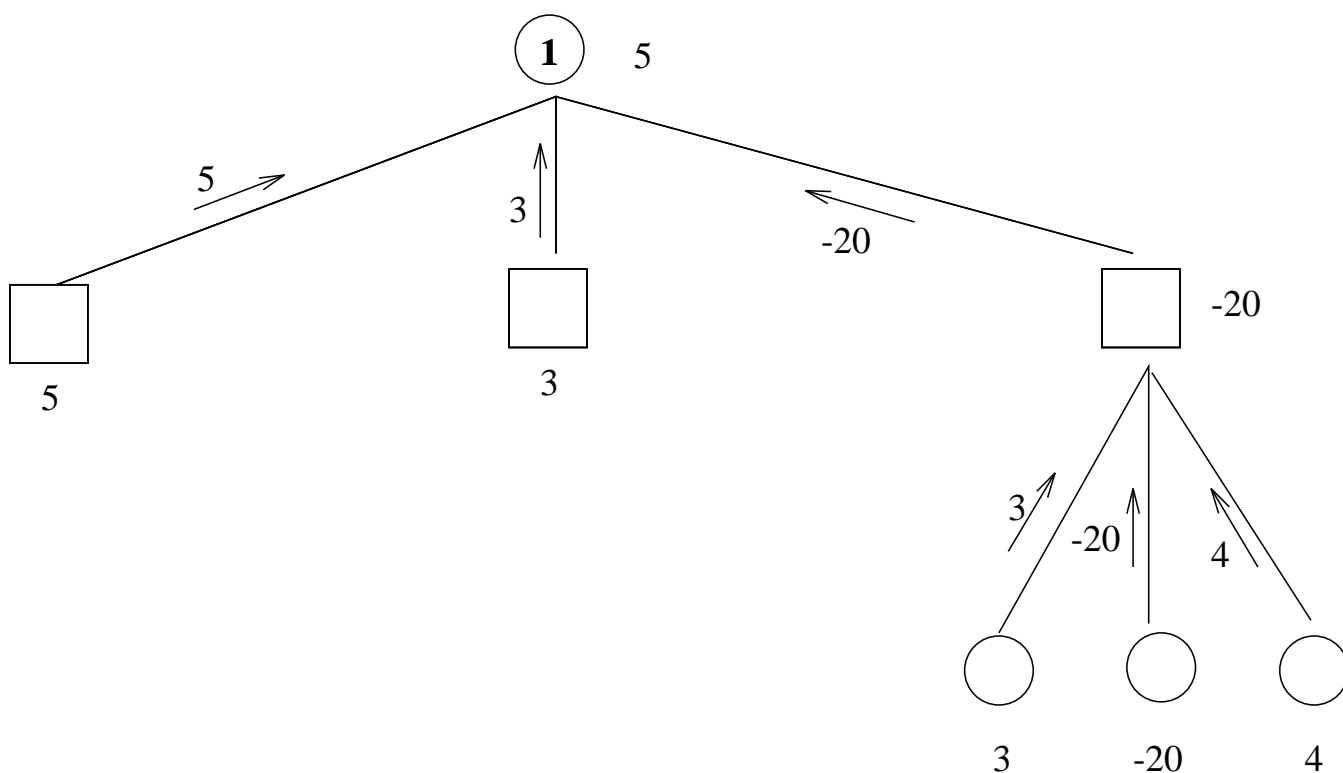


Figure 44: Ejemplo de falta de equilibrio producido por el efecto horizonte.

aproximación con miras a reducir el alto coste.

15.4 Algoritmo SSS*

Stockman en 1979 propuso el algoritmo sss* (“State Space Search”), de búsqueda en espacio de estados, como oposición al carácter secuencial del algoritmo Alfa-Beta. Este algoritmo hace una búsqueda mejor-primero sobre un espacio de estados que constituye el árbol de búsqueda.

Antes de describir el algoritmo, conviene definir dos conceptos:

- G es árbol de juego si:
 - ▷ G es no vacío
 - ▷ Todos los sucesores de un nodo MAX son nodos MIN
 - ▷ Todos los sucesores de un nodo MIN son nodos MAX.
- T es un árbol solución de un árbol de juego G si:
 - ▷ La raíz de T, R_T , es la raíz de G, R_G .
 - ▷ Si n es un nodo MAX, no terminal y pertenece a T, entonces uno solo de sus sucesores debe estar en T. Será, además, aquél que tenga mayor valor.
 - ▷ Si n es un nodo MIN, no terminal y pertenece a T, entonces todos sus sucesores deben pertenecer a T.

En la figura 45, pueden observarse los tres subárboles solución (T_1 , T_2 , y T_3) del árbol de juego mostrado en la figura 39.

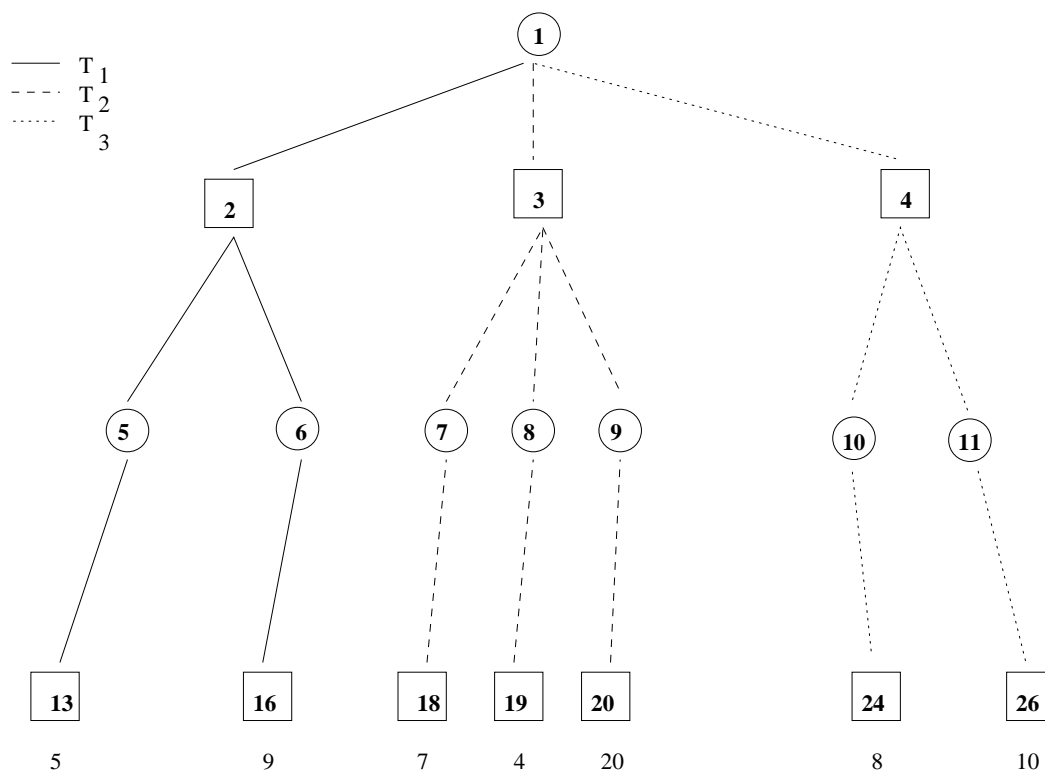


Figure 45: Ejemplos de árboles solución del árbol de la figura 39.

De la anterior definición se extraen las siguientes consecuencias:

- $f(R_T) = \min_{n \in H(R_T)} f(n)$
siendo $H(R_T)$ el conjunto de nodos hoja del árbol T . Puesto que los nodos MAX sólo tienen un sucesor, desde la raíz del árbol se contemplan sólo decisiones MIN, con lo que el valor de la raíz será el mínimo de los valores de sus hojas.
- $f(R_G) = \max_{n \in \text{succ}(R_G)} f(n) = \max_{T \in S(G)} f(R_T)$
siendo $S(G)$ el conjunto de árboles solución de G . Esto es debido a que, para cada decisión suya, sólo existe un sucesor (la jugada a realizar) y para cada decisión del contrario están todas las posibles respuestas.
- Cada T representa una estrategia para MAX, puesto que para cada posible alternativa de MAX, conserva sólo la mejor alternativa (la que tiene que seleccionar), mientras que para cada alternativa del contrario, conserva todas (no sabe cuál escogerá).

Un estado del sss* tiene la estructura (n, e, h) , donde n es el nodo correspondiente del árbol de búsqueda, e es el estado de ese nodo que puede ser activo o estudiado y h es el valor de la función heurística para ese nodo.

El algoritmo retira de la lista ABIERTA (que contiene todos aquellos estados no expandidos) aquel estado que tenga la mayor h . Después de comprobar si es el estado final, se va a una tabla

donde están contemplados todos los posibles casos y las acciones correspondientes. Según el caso, se ejecutan las acciones especificadas para el mismo y se vuelve al principio. Formalmente, el algoritmo sería el mostrado en la tabla adjunta.

<p>Procedimiento sss* Introducir en ABIERTA el estado inicial: $(n = \epsilon(\text{raíz del árbol}), e=\text{activo}, h = +\infty)$ Terminado = FALSO Repetir hasta que Terminado = VERDADERO Eliminar el primer estado de ABIERTA $(p = (n, e, h))$ SI $n = \epsilon$ (raíz del árbol) y $e=\text{estudiado}$, ENTONCES Terminado = VERDADERO SI NO, expandir el nodo p, aplicando el operador Γ (Tablas 2 y 3). Devolver jugada que introdujo el h (valor minimax del juego)</p>

La tabla 2 muestra el operador Γ que rige el comportamiento del algoritmo sss* en el caso de que el estado del nodo en cuestión esté ACTIVO. En el paso 3, los empates se resuelven prefiriendo los nodos que estén situados más a la izquierda en el árbol de búsqueda. La tabla 3 muestra el operador Γ en el caso de que el estado sea ESTUDIADO.

Caso	Condiciones satisfechas por el estado (n,s,h)	Acción
1	s=Activo tipo(n)=MAX n es no-terminal	$\forall n' \in \text{suc}(n)$, Añadir los estados (n', Activo, h) , al principio de Abierta
2	s=Activo tipo(n)=MIN n es no-terminal	Añadir (n', Activo, h) al principio de Abierta, donde n' es el primer sucesor de n
3	s=Activo n es terminal	Introducir $(n, \text{Estudiado}, \min\{h, f(n)\})$ delante de los estados de Abierta con menor h

Table 2: El Operador Γ (tres primeros casos).

Según Stockman, sss* no estudia nodos que no estudie el Alfa-Beta. Sin embargo, Roizen y Pearl reforman la tabla puesto que, sino, habría casos en los que no se cumpliría esta propiedad. Por otra parte, el sss* tiene el gran problema de requerir mucha memoria para la lista ABIERTA, y ha habido intentos de reducir esta ocupación de memoria, estableciendo un máximo M de ocupación de memoria (número de estados en ABIERTA), y dando lugar al algoritmo ITERSSS*. También se ha intentado proporcionar, durante la búsqueda, al Alfa-Beta información para guiar la búsqueda posterior.

Caso	Condiciones satisfechas por el estado (n,s,h)	Acción
4	s=Estudiado tipo(n)=MAX n tiene hermanos sin estudiar	Añadir (n',Activo,h) al principio de Abierta, donde n' es el primer hermano sin estudiar de n
5	s=Estudiado tipo(n)=MAX n no tiene hermanos sin estudiar	Añadir (padre(n),Estudiado,h) al principio de Abierta
6	s=Estudiado tipo(n)=MIN	Añadir (padre(n),Estudiado,h) al principio de Abierta. Eliminar de Abierta todos los nodos sucesores en el árbol del nodo padre(n)

Table 3: El Operador Γ (tres últimos casos).

En las tablas 4 y 5, se muestra una traza de la ejecución del algoritmo sss* para el ejemplo de la figura 39. En la primera columna está el número de ciclo; en la segunda el caso según las tablas 2 y 3; y en la tercera, la lista ABIERTA en cada ciclo. La jugada seleccionada es la que introdujo el nodo raíz en el penúltimo ciclo: la representada por el nodo 4, que es la misma jugada seleccionada por el Alfa-Beta.

Ciclo	Caso	Abierta
1	1	(1,A, ∞)
2	2	(2,A, ∞) (3,A, ∞) (4,A, ∞)
3	1	(5,A, ∞) (3,A, ∞) (4,A, ∞)
4	3	(12,A, ∞) (13,A, ∞) (14,A, ∞) (3,A, ∞) (4,A, ∞)
5	3	(13,A, ∞) (14,A, ∞) (3,A, ∞) (4,A, ∞) (12,E,3)
6	3	(14,A, ∞) (3,A, ∞) (4,A, ∞) (13,E,5) (12,E,3)
7	2	(3,A, ∞) (4,A, ∞) (13,E,5) (14,E,4) (12,E,3)
8	1	(7,A, ∞) (4,A, ∞) (13,E,5) (14,E,4) (12,E,3)
9	3	(17,A, ∞) (18,A, ∞) (4,A, ∞) (13,E,5) (14,E,4) (12,E,3)
10	3	(18,A, ∞) (4,A, ∞) (17,E,6) (13,E,5) (14,E,4) (12,E,3)
11	2	(4,A, ∞) (18,E,7) (17,E,6) (13,E,5) (14,E,4) (12,E,3)
12	1	(10,A, ∞) (18,E,7) (17,E,6) (13,E,5) (14,E,4) (12,E,3)

Table 4: Ejemplo de resolución del árbol de la figura 39 utilizando el algoritmo sss* (primera parte).

15.5 Algoritmo B*

Tanto en el Alfa-Beta como en el sss*, la $h(n)$ devolvía un sólo valor y era única. Berliner expuso un algoritmo de búsqueda, válido para juegos de una persona (juegos contra la naturaleza o solitarios) y de dos personas, que utiliza dos funciones de evaluación: una que suministra un umbral inferior o valor pesimista, $p(n)$, y otra que devuelve un umbral superior u optimista, $o(n)$. Cada nodo

Ciclo	Caso	Abierta
13	3	(24,A, ∞) (25,A, ∞) (18,E,7) (17,E,6) (13,E,5) (14,E,4) (12,E,3)
14	3	(25,A, ∞) (24,E,8) (18,E,7) (17,E,6) (13,E,5) (14,E,4) (12,E,3)
15	6	(24,E,8) (18,E,7) (17,E,6) (13,E,5) (25,E,5) (14,E,4) (12,E,3)
16	4	(10,E,8) (18,E,7) (17,E,6) (13,E,5) (14,E,4) (12,E,3)
17	1	(11,A,8) (18,E,7) (17,E,6) (13,E,5) (14,E,4) (12,E,3)
18	3	(26,E,8) (27,E,8) (28,E,8) (18,E,7) (17,E,6) (13,E,5) (14,E,4) (12,E,3)
19	6	(26,E,8) (27,E,8) (28,E,8) (18,E,7) (17,E,6) (13,E,5) (14,E,4) (12,E,3)
20	5	(4,E,8) (18,E,7) (17,E,6) (13,E,5) (14,E,4) (12,E,3)
21	6	(1,E,8)

Table 5: Ejemplo de resolución del árbol de la figura 39 utilizando el algoritmo SSS* (segunda parte).

tendrá, por tanto, una ventana que serán los valores superior e inferior entre los que debe estar comprendido el valor. La condición de fin del algoritmo es que

$$\forall n \in \text{suc}(R_G) \mid p(n) \geq \max_{n' \in \text{HE}(n)} o(n')$$

donde R_G es la raíz del árbol de juego, $\text{HE}(n)$ es el conjunto de los hermanos de n , $p(n)$ es el valor pesimista del nodo n , o umbral inferior, y $o(n')$ es el valor optimista del nodo n' , o umbral superior, o, lo que es lo mismo, se trata de que el valor pesimista del mejor sucesor del nodo raíz sea mayor o igual que el valor optimista de sus hermanos.

Para conseguir esto, existen dos tipos de estrategias:

- PROVEBEST, que intenta incrementar el $p(n)$ del mejor sucesor del nodo raíz.
- DISPROVEREST, que intenta decrementar el $o(n)$ del segundo sucesor del nodo raíz.

Por otra parte, las reglas de “backup” son:

$$p(n) = \max_{n' \in \text{suc}(n)} o(n')$$

$$o(n) = \max_{n' \in \text{suc}(n)} p(n')$$

Berliner describe el algoritmo de forma no estructurada, mientras que Palay lo hace de forma estructurada. La descripción de este último consta de dos procedimientos. El primero actúa sobre los nodos sucesores de la raíz del árbol de búsqueda y el segundo actúa sobre los niveles inferiores del árbol. El algoritmo queda de la forma especificada por los procedimientos B*-NIVEL-ALTO y B*-NIVEL-BAJO.

Procedimiento B*-NIVEL-ALTO (Situación)

1. Expandir el nivel alto del árbol desde Situación
2. Mientras $p(\text{Mejor-nodo}) < \max_{n' \in HE(\text{Mejor-nodo})} o(n')$
 - (a) Seleccionar estrategia PROVEBEST o DISPROVEBEST
 - (b) B*-NIVEL-BAJO (nodo-a-explorar-según-estrategia)
3. Devolver Mejor-nodo

Procedimiento B*-NIVEL-BAJO (Nodo)

1. SI Nodo no expandido ENTONCES EXPANDIR-NODO (Nodo)
2. $O = - \max_{n' \in \text{suc}(\text{Nodo})} p(n')$
3. $P = - \max_{n' \in \text{suc}(\text{Nodo})} o(n')$
4. Mientras $O = o(\text{Nodo})$ Y $P = p(\text{Nodo})$
 - (a) Siguiente-nodo = siguiente nodo a explorar del conjunto $\text{suc}(\text{Nodo})$
 - (b) B*-NIVEL-BAJO (Siguiente-Nodo)
 - (c) $O = - \max_{n' \in \text{suc}(\text{Nodo})} p(n')$
 - (d) $P = - \max_{n' \in \text{suc}(\text{Nodo})} o(n')$
5. $o(\text{Nodo}) = O$
6. $p(\text{Nodo}) = P$
7. Devolver $o(\text{Nodo})$ y $p(\text{Nodo})$

Palay también describe una serie de pautas para utilizar una u otra estrategia (PROVEBEST/DISPROVEREST) dependiendo del caso en el que se encuentre la búsqueda. Demuestra también que, pese a lo que pueda parecer, existen casos en los que el mejor nodo no se expande, porque se utiliza siempre la estrategia DISPROVEREST.

Pese a poder utilizarse para juegos de una y dos personas, se ha demostrado que no es adecuado para el primer tipo con lo que, normalmente, se utiliza para juegos de adversarios.

15.6 Algoritmo Max^n

Cuando el número de jugadores es mayor que dos, hay que realizar modificaciones para los algoritmos anteriores. Luckhardt e Irani, en 1986, plantearon un algoritmo válido para juegos de n personas, no cooperativo y de información perfecta (completa). Más tarde, Luckhardt amplió el mismo algoritmo, Max^n , para juegos cooperativos (dos o más jugadores forman grupos con intereses comunes).

La principal idea del algoritmo consiste en que en lugar de devolver un sólo valor (Alfa-Beta y SSS^*) o dos valores (B^*), la función de evaluación devuelva un vector con tantos componentes como jugadores haya. Cada componente i del vector representa la estimación de la victoria del jugador i . En cada nivel del árbol, se escoge, de los vectores de los nodos sucesores, aquél que maximice la componente del vector correspondiente al jugador que tiene la decisión en ese nivel.

Así, por ejemplo, en el árbol de la figura 46 se observa que en el nivel de profundidad 2 le toca decidir al jugador número 3 ya que son sus jugadas, por lo que maximiza la tercera componente del vector. Así, entre los nodos 11 y 12 prefiere el 12, ya que maximiza la tercera componente del vector. Análogamente, deciden el jugador dos y uno en los niveles superiores respectivos, tal como muestran las figuras 47, 48, y 49.

También plantean una optimización de este algoritmo que consiste en retrasar la evaluación de las componentes del vector hasta el momento en el que sean necesarias para decidir. Así, en el nivel 3 sólo se obtendrían las terceras componentes del vector, en el nivel 2 las segundas de los vectores elegidos, y en el nivel 1 las primeras componentes.

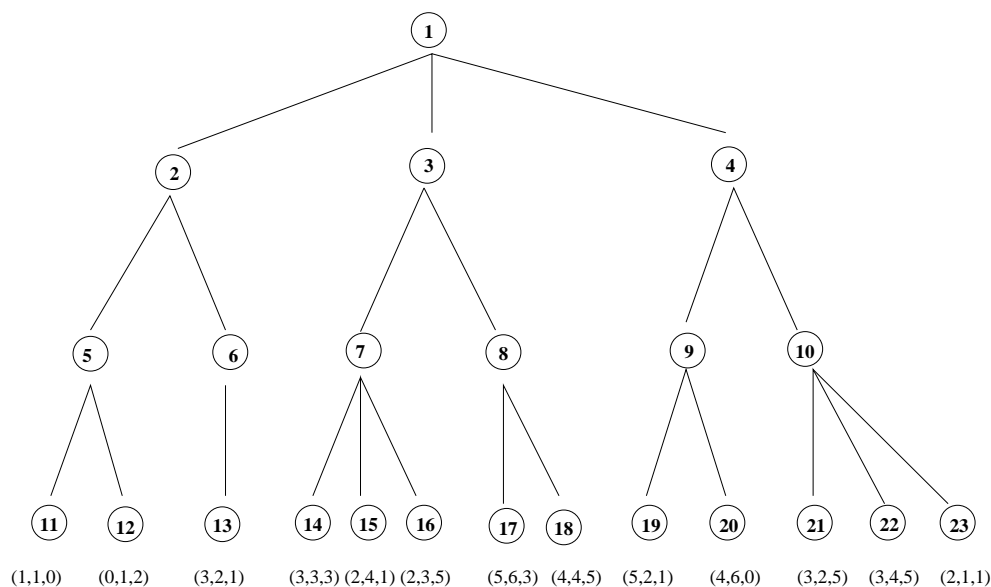
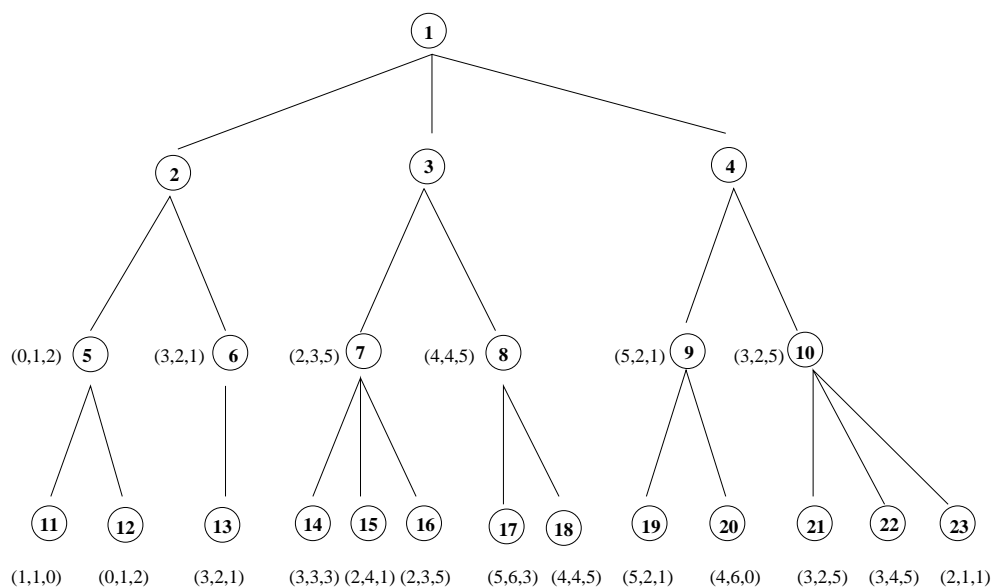


Figure 46: Ejemplo de funcionamiento del Max^n .

Ultimamente, McAllester ha presentado otro método de estudio de árboles de juego basado en los “números conspiratorios”. Dichos números representan, para cada nodo, el número de nodos cuyo valor habrá que cambiar para lograr cambiar el valor de ese nodo.

Figure 47: Ejemplo de funcionamiento del Maxⁿ.

16 Ejercicios de autocomprobación

Problema 1.

Supóngase que se tiene un dominio en el que se tiene un vehículo que debe ir de un punto a otro de una ciudad. Se definen los puntos como las intersecciones entre tramos de calles. El objetivo es que el vehículo vaya del punto origen al punto destino por el camino por el que menos se tarde.

Se conocen, además, los siguientes datos:

- Si el número de coches en un tramo es menor que 10, el tráfico es disperso. La velocidad, entonces, será de 50 km/h, a no ser que llueva, que será de 30 km/h, haya niebla, 20 km/h, o nieve, 15 km/h.
- Si el número de coches en un tramo está entre 10 y 100, el tráfico es medio. La velocidad, entonces, será de 40 km/h, a no ser que llueva o nieve que será de 25 km/h, o haya hielo en el suelo, que será de 10 km/h. Si hay hielo en un tramo, éste será poco recomendable.
- Si el número de coches en un tramo es mayor que 100, el tráfico será alto, y la velocidad será de 30 km/h. Si hay obras, la velocidad será de 15 km/h y si llueve 10 km/h.

Supóngase que existe un sistema de producción que describe este dominio, en el que la resolución del conjunto conflicto se realiza por medio del A*. Dada la ciudad de la figura 50, donde los pares de números representan coordenadas, describese el comportamiento de la búsqueda utilizando los algoritmos A* y escalada para ir desde el punto I al punto M por el camino por el que tarde menos tiempo, donde la información relativa a los tramos es la especificada en la tabla adjunta.

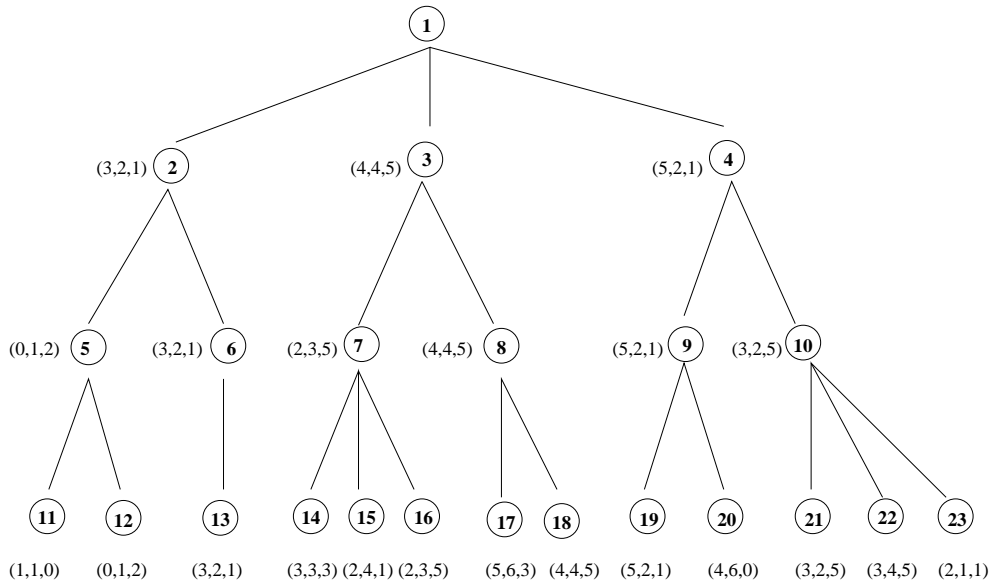


Figure 48: Ejemplo de funcionamiento del Maxⁿ.

Atributo	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Coches	9	15	9	150	25	5	50	200	5	60
Lluvia	x									x
Nieve		x				x				
Obras				x	x					
Niebla	x		x							
Hielo			x		x					

Solución al problema 1.

Debido a que se pide obtener el camino por el que se tarde menos tiempo, la función de evaluación $f(n)$ será $f(n) = g(n) + h(n)$, donde $g(n)$ se calculará como la suma de los tiempos en los trayectos obtenidos hasta el momento desde el nodo raíz del árbol de búsqueda hasta el nodo n , y $h(n)$ se calcula como el tiempo estimado para llegar al nodo meta desde el nodo n . Más concretamente, el coste para ir de un nodo n a otro m se calcula como:

$$\frac{e(n, m)}{v(n, m)}$$

donde $v(n, m)$ es la velocidad por la que se puede circular por el tramo correspondiente, y $e(n, m)$ es la distancia euclídea de n a m , y se calcula como:

$$e(n, m) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Por otra parte, $h(n)$ debe ser siempre menor o igual que la real, con lo que se puede definir como $h(n) = \frac{e(n, meta)}{50}$, ya que 50 es la mayor velocidad por la que se puede circular. Por tanto, las $h(n)$ de cada nodo serán:

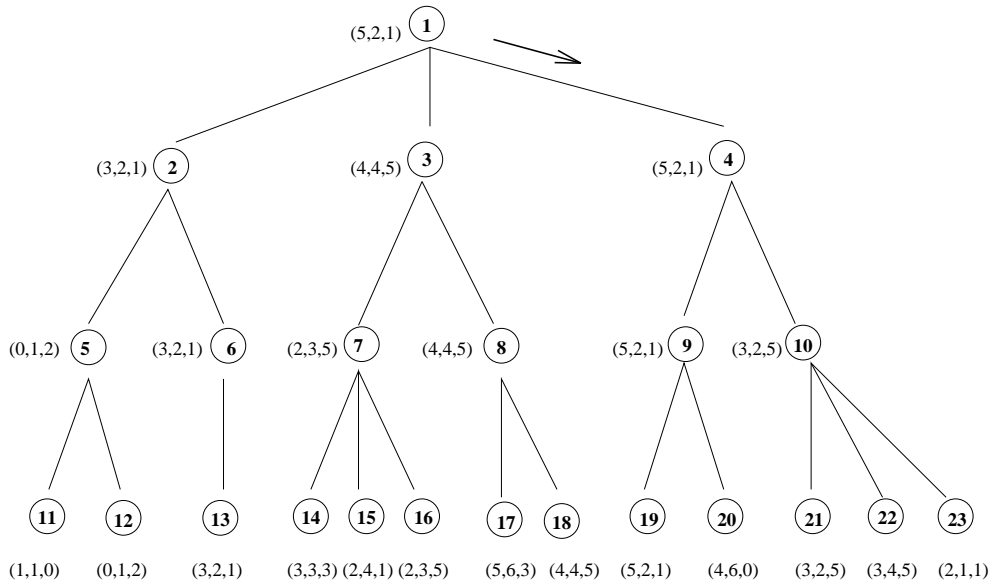


Figure 49: Ejemplo de funcionamiento del Maxⁿ.

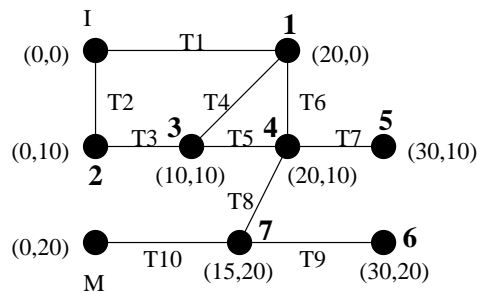


Figure 50: Ejemplo de ciudad

	Nodos			
	1	2	3	4
$h(\text{nodo})$	$\frac{20\sqrt{2}}{50} = 0.56$	$\frac{10}{50} = 0.2$	$\frac{10\sqrt{2}}{50} = 0.28$	$\frac{\sqrt{300}}{50} = 0.35$
	Nodos			
	5	6	7	Meta
$h(\text{nodo})$	$\frac{\sqrt{800}}{50} = 0.56$	$\frac{30}{50} = 0.6$	$\frac{15}{50} = 0.3$	$\frac{0}{50} = 0$

Las velocidades se calculan mediante las reglas descritas y serán las siguientes (puede haber variaciones, según se entiendan las reglas):

	Tramos				
	T1	T2	T3	T4	T5
Velocidad	20	25	20	15	10
Tiempo	$\frac{20}{20} = 1$	$\frac{10}{25} = 0.4$	$\frac{10}{20} = 0.5$	$\frac{10\sqrt{2}}{15} = 0.95$	$\frac{10}{10} = 1$

	Tramos				
	T6	T7	T8	T9	T10
Velocidad	15	40	30	50	25
Tiempo	$\frac{10}{15} = 0.67$	$\frac{10}{40} = 0.25$	$\frac{\sqrt{125}}{30} = 0.37$	$\frac{15}{50} = 0.3$	$\frac{15}{25} = 0.6$

Los sucesivos árboles expandidos aparecen en las figura 51, 52, 53, 54, 55, y 56. En la derecha de los nodos se han representado la $g(n)$ y, debajo, la $h(n)$. En la izquierda se han representado las $f(n)$.

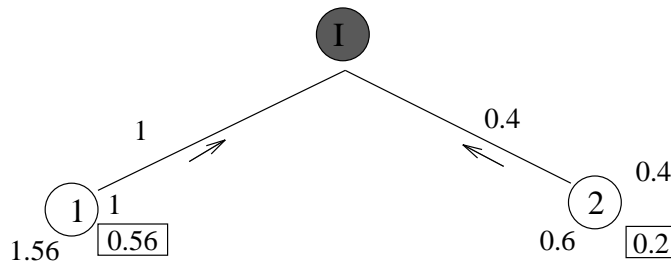


Figure 51: Paso 1 del algoritmo A* sobre el ejemplo de la figura 50.

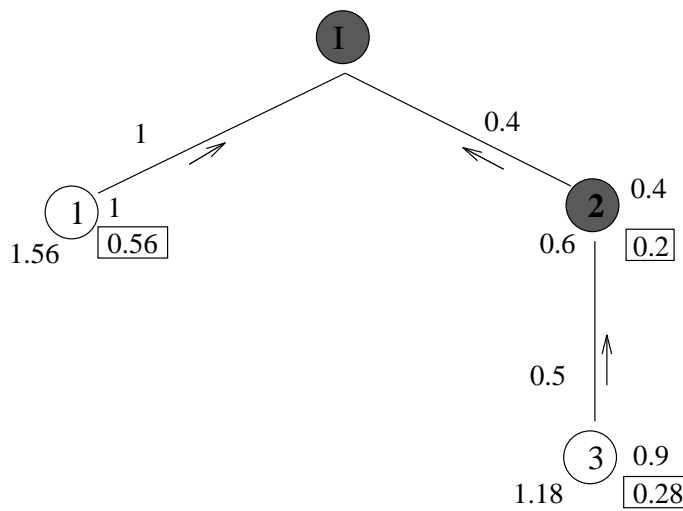


Figure 52: Paso 2 del algoritmo A* sobre el ejemplo de la figura 50.

Problema 2.

Dado el juego del "tic-tac-toe" descrito en el texto, suponiendo que la máquina juega con 0, a partir de la configuración inicial dada por la figura 57, determinar la mejor jugada mediante la utilización

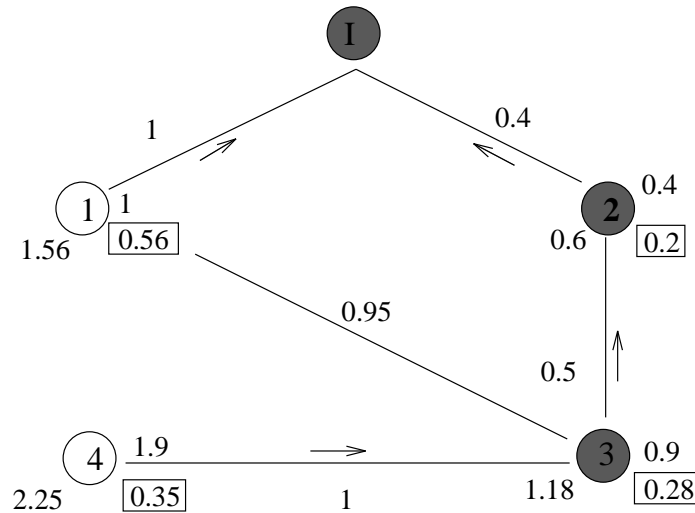


Figure 53: Paso 3 del algoritmo A* sobre el ejemplo de la figura 50.

del algoritmo Alfa-Beta una búsqueda 3-ply (profundidad del árbol de búsqueda = 4). La función de evaluación es:

$$f_{ev}(n) = n^2 \text{líneas libres para O} - n^2 \text{líneas libres para X}$$

siendo línea = fila o columna o diagonal. A continuación, obtener la mejor jugada mediante el algoritmo SSS*.

Solución al problema 2.

Las figuras adjuntas (58, 58, y 58) muestran los diferentes estados de solución de acuerdo al procedimiento Alfa-Beta. En la figura 58 se muestra la expansión por el procedimiento de la rama de la izquierda. En la figura 59, se estudia la rama del centro, realizándose una poda. Por último, la figura 60 muestra el estudio de la rama de la derecha. La solución del ejercicio con el SSS* se deja al lector.

Problema 3.

Dado el juego de los gatos y el ratón descrito en el capítulo de **Representación del Conocimiento**, si la estrategia de control fuera el alfa-beta, describir una posible función de evaluación, y el árbol de búsqueda que el alfa-beta generaría en el caso de la figura 61 cuando le toca mover al ratón, y la profundidad de búsqueda es tres (tres niveles por debajo del nodo raíz).

Solución al problema 3.

Vamos a utilizar una posible función de evaluación:

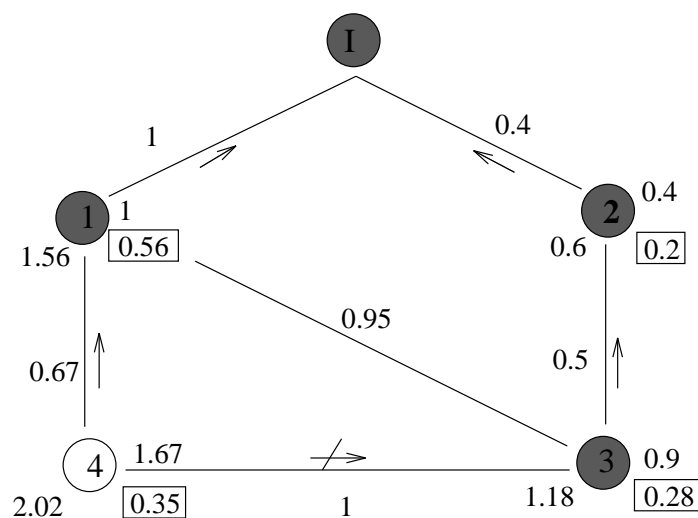


Figure 54: Paso 4 del algoritmo A* sobre el ejemplo de la figura 50.

$f(\text{situación}) = \text{número-jugadas-ratón}(\text{situación}) - \text{número-jugadas-gatos}(\text{situación})$
 El árbol resultante está representado en las figuras 62 y 63.

17 Lecturas recomendadas

Existen muchos y buenos textos en donde se tratan con diferentes niveles de profundidad las distintas técnicas que aquí se expusieron. Así, los libros de [Rich and Knight, 1994, Winston, 1984, Nilsson, 1987, Borrajo *et al.*, 1993] contienen descripciones de la mayor parte de los algoritmos tratados.

References

- [Borrajo *et al.*, 1993] Daniel Borrajo, Natalia Juristo, Vicente Martínez, and Juan Pazos. *Inteligencia Artificial. Métodos y Técnicas*. Centro de Estudios Ramón Areces, Madrid, 1993.
- [Nilsson, 1987] Nils Nilsson. *Principios de Inteligencia Artificial*. Ediciones Díaz de Santos, Madrid, 1987.
- [Rich and Knight, 1994] Elaine Rich and Kevin Knight. *Inteligencia Artificial*. McGraw-Hill, Inc., 1994. Segunda edición.
- [Winston, 1984] Patrick H. Winston. *Artificial Intelligence. Second Edition*. Addison-Wesley, Reading, Mass., 1984.

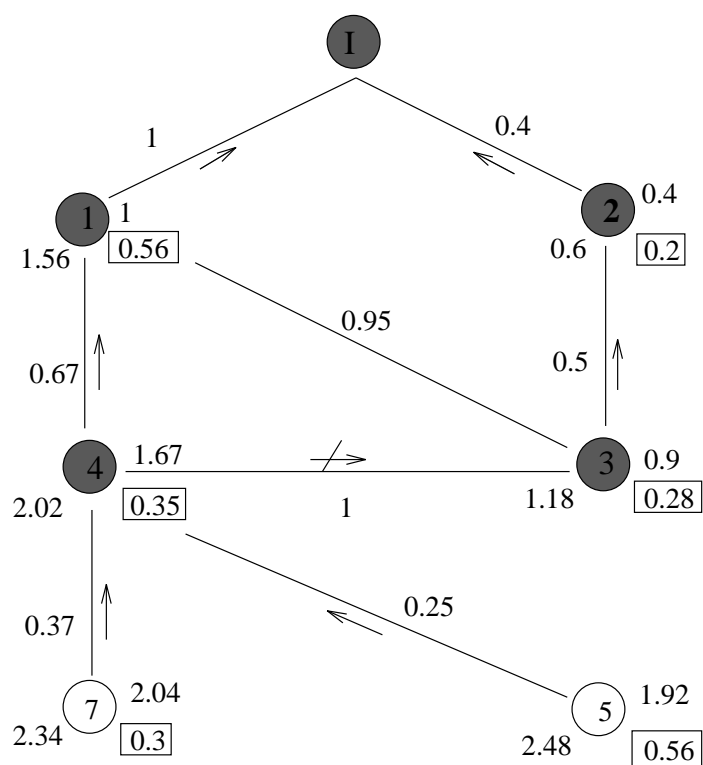


Figure 55: Paso 5 del algoritmo A* sobre el ejemplo de la figura 50.

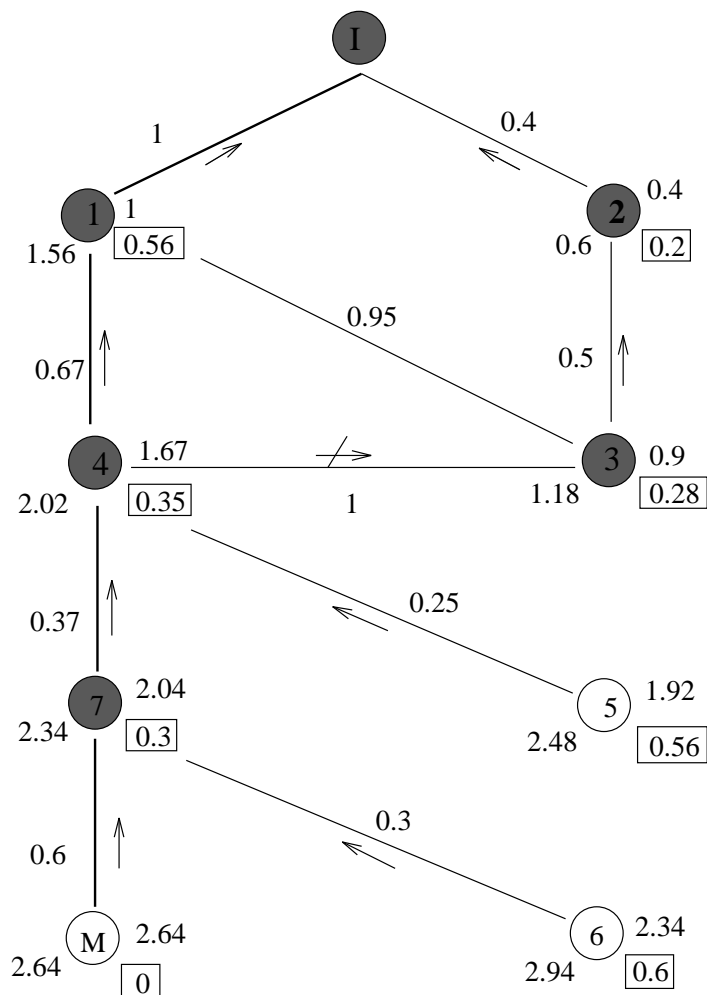


Figure 56: Paso 6 del algoritmo A* sobre el ejemplo de la figura 50.

X		X
○	○	X
		○

Figure 57: Ejemplo de situación en el juego del “tic-tac-toe”.

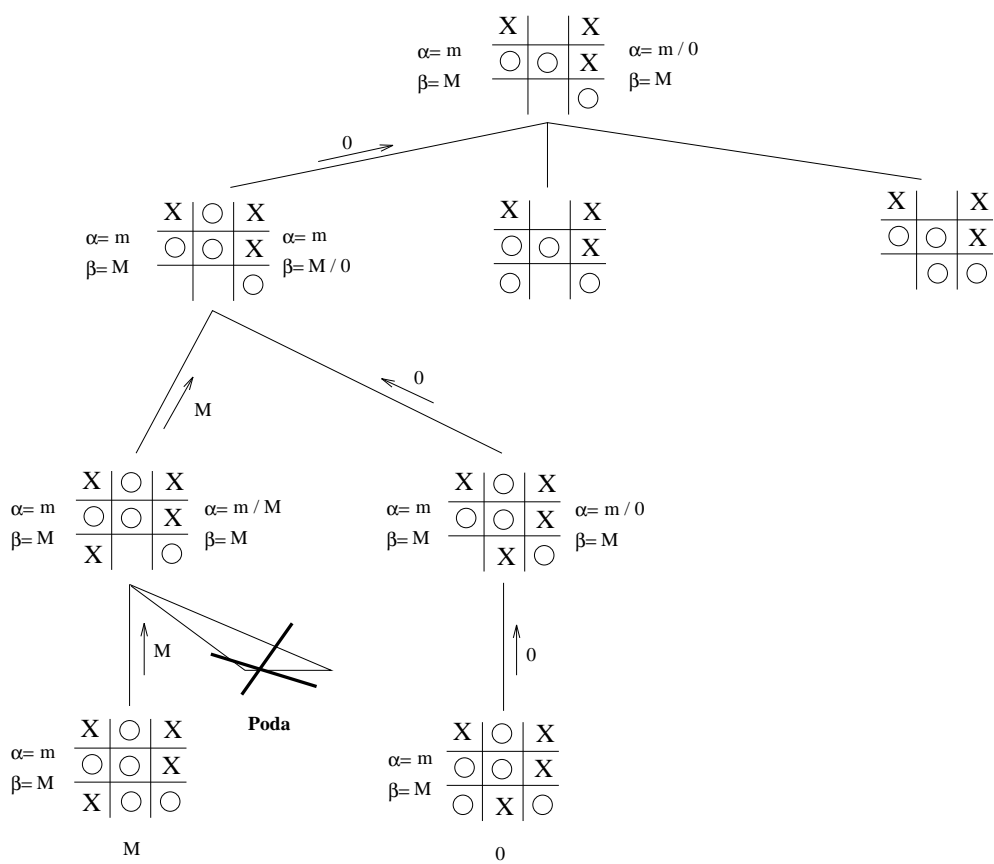


Figure 58: Estudio de la rama izquierda con el Alfa-Beta.

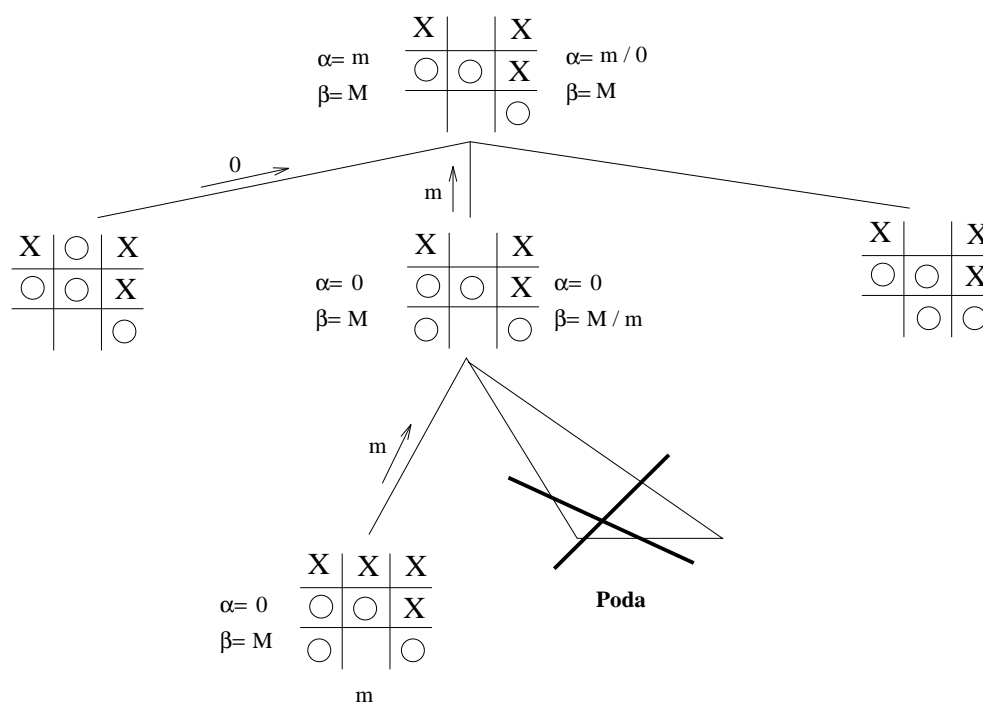


Figure 59: Estudio de la rama central con el Alfa-Beta.

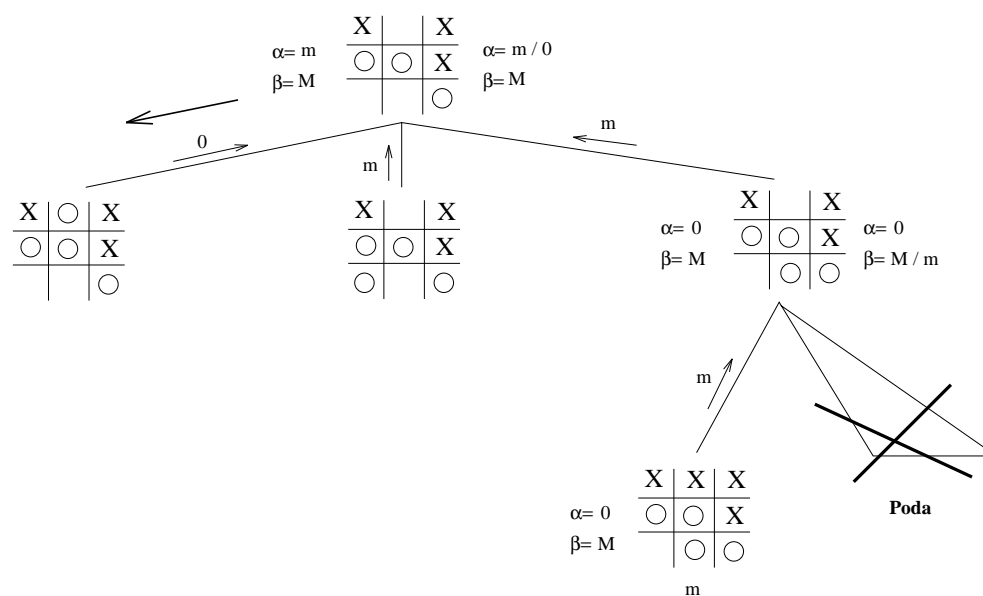


Figure 60: Estudio de la rama derecha con el Alfa-Beta.

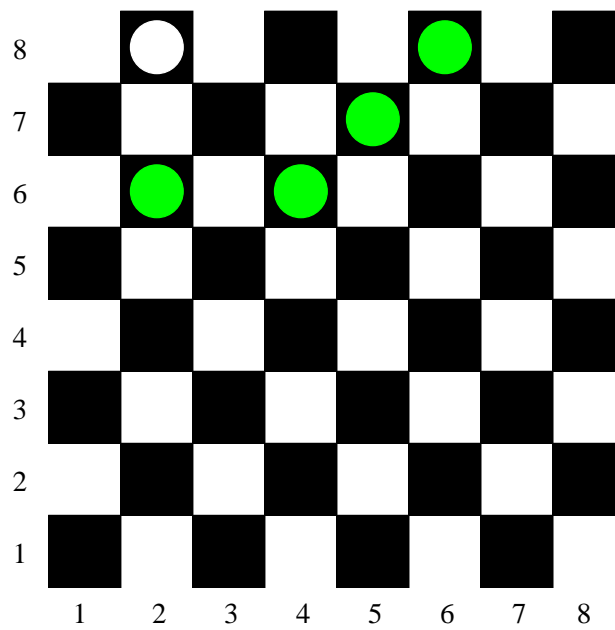


Figure 61: Ejemplo de tablero en el juego de los gatos y el ratón.

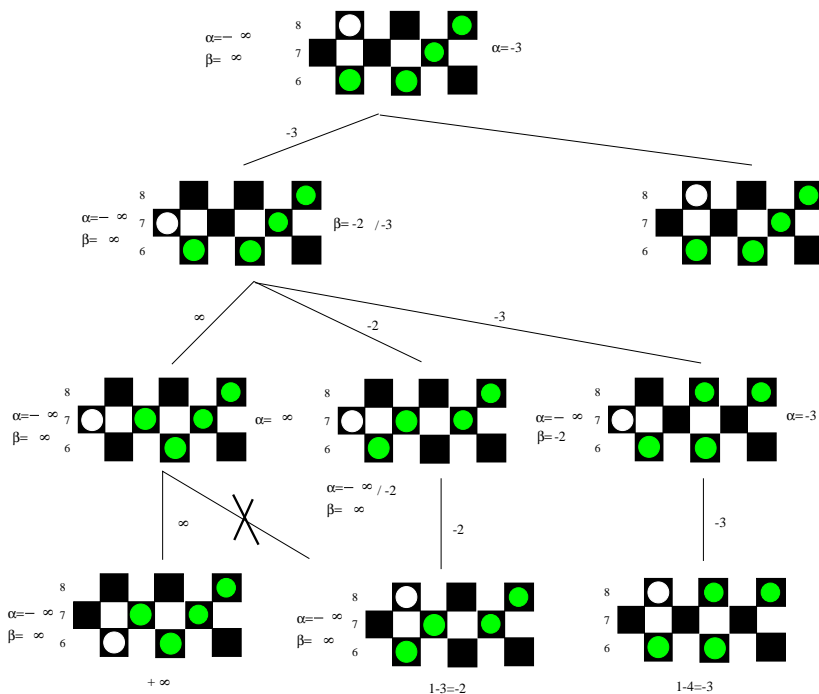


Figure 62: Ejemplo de funcionamiento del Alfa-Beta con el juego de los gatos y el ratón (rama izquierda del árbol).

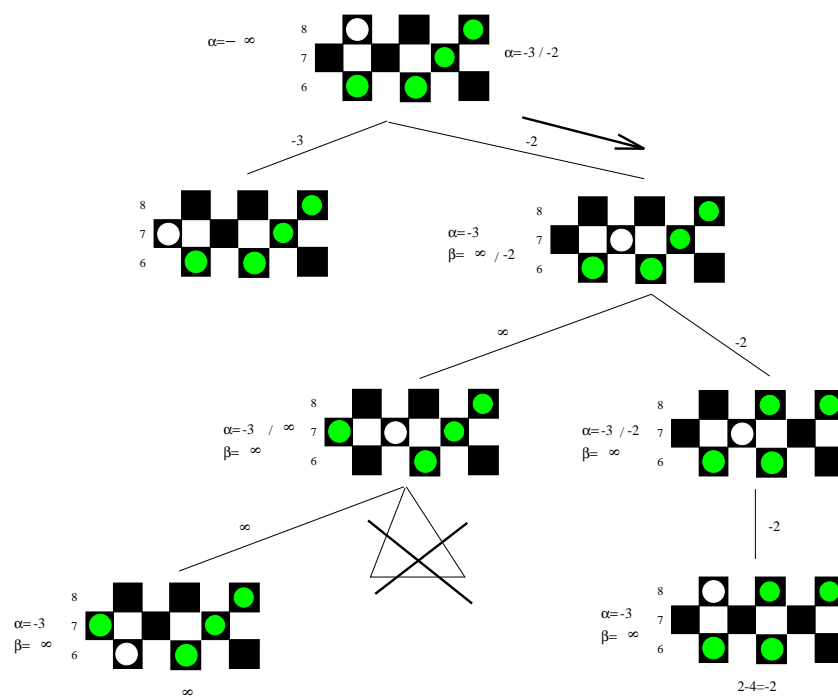


Figure 63: Ejemplo de funcionamiento del Alfa-Beta con el juego de los gatos y el ratón (rama derecha del árbol).