3. PROBLEMAS

"Cuanto más perseguimos el conocimiento, menos sabemos"
Lao Tse (500 a. de C)

Nos enfrentamos con un problema al encontrar dificultades, inquietudes u obstáculos que, en general, nos impiden hallar algo, llegar a una meta o cumplir un objetivo.

La solución de algunos problemas requiere amplio razonamiento para la generación y asimilación de estructuras en la memoria y así combinar el conocimiento que se posee. Existen diferentes vías para encontrar la solución a un problema; por ejemplo:

1. Aplicar una fórmula explícita.
2. Utilizar un algoritmo que converja a una solución, recursivo o no.
3. Generar posible solución y probar si satisface los requisitos
4. Dividir el problema e ir solucionando cada uno de los subproblemas.

{PRIVADO }Tabla 3.1 *Algoritmo Recursivo vs. Iterativo.*

Recursivo	*Iterativo*
Proc fib(n) *I=1* *MQ i <= n* *SI n > 2* *V: fib(n) = fib(n-1)+fib(n-2)* *F: fib(n) = 1* *1=1+1* *FMQ* *finproc()*	*proc fib(n)* *a,b=1* *i=2* *MQ i<=n* *c=a+b* *a=b* *b=c* *i=i+1* *FMQ* *Finproc()*

Las dos primeras vías se emplean continuamente en programas de computador, son modelos matemáticos o lógicos para hallar la solución a problemas[1]. Pero cuando no es posible éstas vías, se emplea alguna de las otras, en general la tercera: ensayo y error. El generar y probar puede ser un proceso no adecuado por el tiempo limitado del que se dispone. El razonamiento del ser humano en la búsqueda de solución a problemas crea métodos "extraños", "exhaustivos", "ilógicos", etc., algunos de los cuales se

[1] *Siempre que sea posible, la segunda vía es la mejor. Los algoritmos son efectivos pero no necesariamente eficientes por la complejidad de algunos problemas. Pero en algunos problemas se requiere de computador por el tipo de cálculos a realizar.*

analizan e implementan con técnicas de IA.

Un ejemplo de cómo pueden ser los algoritmos a involucrar en el computador se muestran en la tabla 3.1 para el mismo problema: calcular los números de Fibonaci, el uno recursivo, el otro iterativo.

El proceso recursivo es de complejidad O[fib(n)], es decir, de orden exponencial. El proceso iterativo es de complejidad O[n], es decir, lineal. Ambos procesos calculan los mismos números, pero, uno realiza menor número de operaciones aunque gasta más memoria.

{PRIVADO }3.1 TIPOS DE PROBLEMAS

Según un antiguo aforismo, *un problema bien planteado ya está medio resuelto*. Muchos de los problemas que se resuelven aplicando técnicas de IA, requieren modelamiento simbólico, definir las configuraciones posibles de los estados. Es decir, se plantea en términos de encontrar una configuración objetivo, a partir de una inicial, y luego se aplica transformaciones validas según el modelo del universo. La respuesta es la secuencia de transformaciones cuya aplicación sucesiva lleva a la configuración deseada. Controlar las acciones de un robot, un sistema o máquina o programación automática son ejemplos en que debe definirse una secuencia de operaciones (movimientos, pasos) para realizar una mínima tarea. Sin embargo, los sistemas se proponen resolver un tipo de problema.

El proceso de solución dependerá de las transformaciones posibles y la complejidad de los estados. Para construir un sistema inteligente básico capaz de resolver un problema específico, es necesario lo siguiente:

1. Definir el problema. Incluye condiciones iniciales y aspectos de las situaciones finales que podrían considerarse como soluciones acpetables.
2. Analizar el problema. Paso importante que facilita determinar los rasgoz que faciliten identificar el proceso o técnica apropiada para resolver el problema.
3. Identificar y representar el conocimiento necesario para resolver el problema.
4. Escoger el mejor proceso para resolver el problema.

Los ejemplos más característicos en IA son los juegos (son universos restringidos para modelar). En un juego, las configuraciones o estados corresponden al posicionamiento de las fichas en el tablero en cada momento. Cada configuración de los estados puede esquematizarse y representarse en forma simbólica. Las transformaciones permitidas corresponden a las reglas o movidas del juego, formalizadas como transiciones entre estados.

Para plantear formalmente un problema, se requiere precisar una representación simbólica de los estados y definir reglas del tipo condición->acción para cada una de

las transiciones válidas dentro del universo modelado. La acción de una regla indica cómo modificar el estado actual en un nuevo estado.

Como ya se dijo, algunos problemas se solucionan por procedimientos determinísticos que garantizan el éxito (algoritmos computables). Otros son problemas cuya solución requieren el planteamiento de estrategias dinámicas; y otros problemas se resuelven por **búsqueda**. *La búsqueda es el ingrediente crítico de la inteligencia*. La búsqueda exhaustiva consume recursos y tiempo. Una búsqueda no garantiza hallar una solución pero sí acercarnos a ella.

{PRIVADO }3.1.1 El rompecabezas-8{TC \l 33 ".1.1 El rompecabezas-8"}

Un ejemplo simple, muy familiar, es el rompecabezas-8. El rompecabezas-8 consiste de ocho fichas movibles en un conjunto de 3 X 3 cuadros. Un cuadro está siempre vacío para ser posible mover una de las fichas adyacentes a él, lo que puede interpretarse como movimiento del cuadro vacío.[2]

Considere el problema de cambiar la configuración inicial en una final o meta (ver figura 3.1). Una solución al problema lleva una apropiada secuencia de los movimientos de las fichas.

Por lo general, hay varias maneras de representar un problema. Seleccionar una buena representación es un arte. Para el rompecabezas-8 y otros problemas, fácilmente se identifican los elementos del problema que corresponden a esos tres componentes. Esos elementos del problema son los estados y reglas para hacer movimientos. En el rompecabezas-8, cada nuevo tablero (configuración) es un estado del problema. El conjunto de todas las posibles configuraciones es el espacio de estados[3] del problema o el espacio del problema. Muchos de los problemas en que nos interesamos tienen un espacio grande. El rompecabezas-8 tiene relativamente espacio pequeño; solamente son 362.880 (9!) diferentes configuraciones para las ocho fichas y el cuadro vacío.

Una vez los estados del problema conceptualmente son identificados, debe construirse

---

[2] *Desde niños nos guían a memorizar datos, fechas, nombres, fórmulas; pero no a definir, entender o comprender información, mucho menos plantear problemas. Los juegos ayudan al ser humano a adquirir destreza mental y desarrollar habilidades para cambiar de estrategia. Por ello, el rompecabezas ofrece la posibilidad de ejercitar la destreza y la lógica mental.*

[3] *El espacio de estados de un juego es un grafo cuyos nodos representan las configuraciones alcanzables (los estados válidos) y cuyos arcos explicitan las movidas posibles (las transiciones de estado). En principio, puede construirse cualquier espacio de estados partiendo del estado inicial, aplicando cada una de las reglas para generar los sucesores inmediatos, y así sucesivamente con cada uno de los nuevos estados generados (en la práctica, los espacios de estados suelen ser demasiado grandes para explicitarlos por completo). Cuando un problema se representa mediante un espacio de estados, la solución computacional corresponde a encontrar un camino desde el estado inicial a un estado objetivo.*

una representación o descripción de ellos para el computador[4]. Esta descripción, entonces se utiliza como BC. Para el rompecabezas-8, una descripción es una matriz 3*3 de los ocho números y el espacio en blanco. La BC inicial es la descripción del estado inicial del problema.

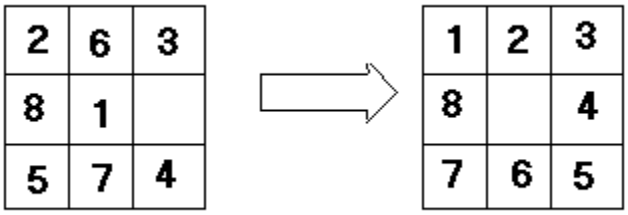


Figura 3.1. *Estado inicial y final para el rompecabezas-8.*

Un movimiento transforma un estado del problema en otro estado. Para el rompecabezas-8 se tiene los siguientes movimientos:

* mover blanco a la izquierda;
* mover blanco a la derecha;
* mover blanco arriba;
* mover blanco abajo.

Esos movimientos son modelados por condiciones que operan sobre la descripción de los estados de manera apropiada.

Cada condición tiene precondiciones que deben satisface una descripción de estados para ser aplicada. Así, la precondición para la regla asociada con "mover blanco abajo" se deriva de los requerimientos que el espacio vacío no debe estar en la fila tope inferior.

Un ejemplo, al aplicar la estrategia tentativa en el rompecabezas-8, cuando las reglas son seleccionadas de acuerdo con un esquema arbitrario el primer intento es mover el blanco a la izquierda, arriba, derecha o abajo. El retroceso ocurre: a) siempre que se genere un estado que ya ocurrió; b) se aplique un conjunto arbitrario de reglas sin llegar al estado meta; c) no existan más reglas para aplicar. En sí, se hace una búsqueda de soluciones.

En el rompecabezas-8 se busca producir un estado en particular del problema, llamado estado meta o final. Una mayor generalización es especificar algunas condiciones Verdad/Falso en estados del sistema que sirven para la condición final. La condición

[4] *Cualquier estructura de datos puede usarse para describir estados; éstas incluyen cadena de símbolos, vectores, conjuntos, arreglos, árboles y listas.*

final es base para la condición de terminación del sistema de producción.

En IA y fuera de ella, un ejemplo muy común al usar el conocimiento para hallar soluciones globales es el proceso *función creciente* que facilita hallar el máximo de esa función. En cualquier punto se procede en la dirección de la curva gradiente para encontrar un máximo.

En algunas funciones, el conocimiento de la dirección de la curva gradiente es suficiente para hallar la solución. Puede usarse en el proceso una función creciente directamente en un sistema de producción (SP). Sólo se requieren algunas funciones de valor real aplicables a la BC. La estrategia de control utiliza la función para seleccionar una regla. Ella selecciona la regla aplicable que produce por base de datos un *incremento* en el valor de la función. Esta función creciente debe ser tal que el valor máximo se halla al satisfacer la condición de terminación.

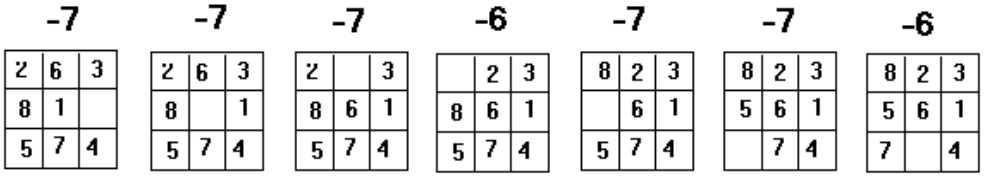


Figura 3.2. *Valores de función creciente para estados de rompecabezas-8.*

Al aplicar una función creciente al *rompecabezas-8*, puede usarse como función el número negativo de fichas fuera de lugar respecto al estado meta.

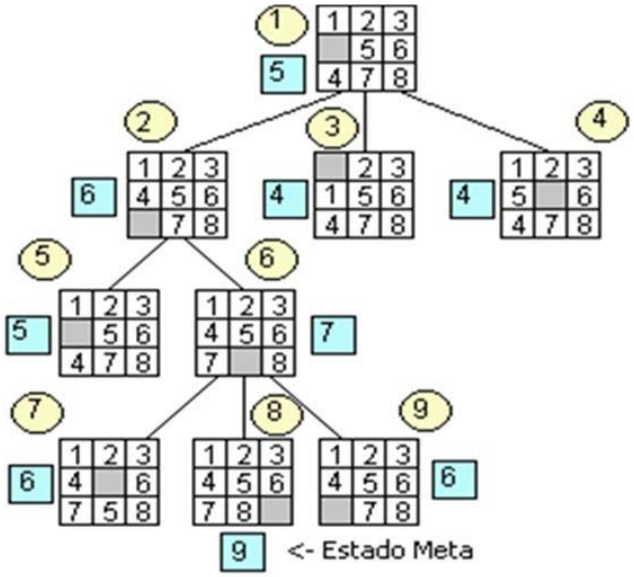
La figura 3.2 muestra una secuencia de estados hallados por un SP al buscar solución al rompecabezas-8, utilizando una función creciente (menor número de fichas que están fuera de la posición final). El valor de la función creciente de cada estado se halla en la parte superior de la figura. Al aplicar yna regla no incrementa el valor de la función. Al no existir reglas que incrementen el valor de la función, una regla cualquiera se aplica pero que no disminuya el valor. Desde luego existen funciones que presentan varios máximos (o mínimos) locales.

Diferentes obstrucciones por una función creciente ocurren en problemas de interés y presentan dificultades en el proceso, como tener mesetas y crestas (funciones fácilmente computables). ¿Cómo saber cuál es la mejor función mejor? No hay manera fácil de concretarse. Presumiblemente cualquier función conocida es mejor que alguna desconocida; de otro modo, se debe emplear una función que da mejores resultados. Pero existe la garantía de definir otra función que supere el rendimiento de la empleada[5]. Así, el uso de métodos crecientes para guiar la selección en SP de manera

[5] *La eficacia de cualquier técnica de investigación depende a la vez de la longitud de la vía de solución que implica, comparada con la longitud mínima posible, y del coste (en términos de tiempo y recursos exigidos) que supone hallar esta vía. A veces, es posible determinar analíticamente la longitud mínima de*

irrevocable es muy limitado. A menudo, la estrategia de control no puede seleccionar la mejor regla para aplicar a un estado.

Figura 3.3. *Árbol de movimientos del rompecabezas-8.*



Algunos problemas pueden tener una representación gráfica[6] que indique todas las alternativas para hallar la solución o no. La figura 3.3 muestra parte del árbol para el rompecabezas-8.

Cuando un problema puede representarse mediante un espacio de estados la solución computacional corresponde a encontrar un camino desde el estado inicial a su estado meta.

{PRIVADO }3.1.2 Las 8-reinas{TC \l 33 ".1.2 Las 8-reinas"}

La meta en el problema de las 8-reinas es colocar ocho reinas en un tablero de ajedrez de manera que ninguna de ellas ataque a otra (una reina ataca horizontal, vertical y diagonalmente).[7]

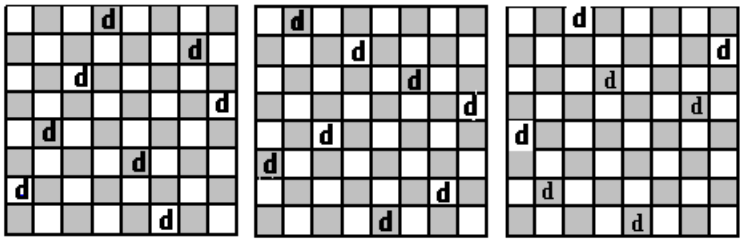
Figura 3.4. *Unas soluciones de colocación de damas.*

---

una vía de solución, en cuyo caso se juzga directamente la eficacia relativa de esa técnica de investigación.

[6] *Grafos (o más especialmente, árboles) son estructuras que permiten visualizar los caminos de solución y, por tanto, facilitan guardar marcas para control de las secuencias de reglas.*

[7] *Un problema que tiene 92 soluciones.*



El conjunto de estados es el tablero con 0 a 8 reinas. El operador es poner una reina en cualquiera de los cuadros sin que ataque a otra reina. Es posible que no exista función de costos (algunas veces se solicita mínimo tiempo o mínimo número de movimientos). Por ejemplo, posibles soluciones se muestran en la figura 3.4

{PRIVADO }3.1.3 Criptoaritmética{TC \l 33 ".1.3 Criptoaritmética"}

En los problemas de criptoaritmética, las letras representan dígitos y el objetivo es determinar el dígito correspondiente a cada letra para cumplir que la operación propuesta sea correcta aritméticamente.

El conjunto de estados es el reemplazo de cada letra por un único dígito. Los operadores (funciones o acciones) es remplazar una letra por un dígito que no repita alguno de los ya hallados. La prueba de meta está cuando la operación sea correcta.

{PRIVADO }			D	O	S					D	O	S
+		T	R	E	S	*				D	O	S
=	C	I	N	C	O	=	C	U	A	T	R	O

{PRIVADO }3.1.4 Recipientes de agua{TC \l 33 ".1.4 Recipientes de agua"}

Se tienen dos recipientes, uno de 3 galones y otro de 4 galones. Ninguno tiene marcas de nivel. Existe una fuente para llenar los recipientes con agua si se requiere. ¿Cómo puede tenerse exactamente 2 galones de agua en el recipiente de 4 galones?

Considerando que X representa al recipiente de 4 galones y Y al de 3 galones podemos tener:

X = 0,1,2,3,4; **Y** = 0,1,2,3

Las operaciones válidas son llenar recipiente, vaciar recipiente y pasar agua de un recipiente a otro. Las reglas a utilizar son:

1. Llenar el recipiente de 4 galones:
SI (X,Y) , X < 4 => (4, Y)
2. Llenar el recipiente de 3 galones:
SI (X,Y), Y < 3 => (X, 3)
3. Vaciar el recipiente de 4 galones:
SI (X,Y), Y > 0 => (0, Y)
4. Vaciar el recipiente de 3 galones:
SI (X,Y), Y > 0=> (X, 0)
5. Pasar agua del recipiente de 3 galones al de 4, hasta que se llene:
SI (X,Y), (X + Y) >= 4, Y > 0 => (4, Y - (4 - X))
6. Pasar agua del recipiente de 4 galones al de 3, hasta que se llene:
SI (X,Y), (X + Y) >= 3, X > 0 => (X - (3 - Y), 3)
7. Pasar toda el agua del recipiente de 3 galones al de 4:
SI (X,Y), (X + Y) <= 4, Y > 0 => (X + Y, 0)
8. Pasar toda el agua del recipiente de 4 galones al de 3:
SI (X,Y), (X + Y) <= 3, X > 0 => (0, X + Y).

El programa debería encontrar un conjunto de estados para ir del estado (4,3) al estado (2,0). Existe más de un camino hacia la solución, por ejemplo:

(4,3) -> (0,3) -> (3,0) -> (3,3) -> (4,2) -> (0,2) -> (2,0)

Otro camino es:

(4,3) -> (4,0) -> (1,3) -> (1,0) -> (0,1) -> (4,1) -> (2,3) -> (2,0)

Con respecto a las reglas puede decirse: las condiciones establecidas en la parte izquierda a veces no son necesarias pero restringen la aplicación de la regla a estados más adecuados. Esto incrementa la eficiencia del programa que las utiliza.

Por ejemplo, la regla 1 llenar el recipiente de 4 galones: SI (X,Y), X<4 => (4,Y). La condición X<4, especifica que recipiente no se halla lleno. Si esta condición no se incluye, se aplicaría la regla aun cuando el recipiente este lleno. Como el estado del problema no cambiaría la aplicación de la regla se considera inútil.

Las reglas no solo deben describir el problema sino también algún tipo de conocimiento sobre su solución.

A veces, cuando se alcanzan algunos estados resulta obvio preguntar cómo continuar hacia la solución. Es posible agregar reglas de propósito especial. En este ejemplo, pueden agregarse las siguientes reglas:

SI (2,Y), Y>0 => (2,0)
SI (X,2) => (2,0)

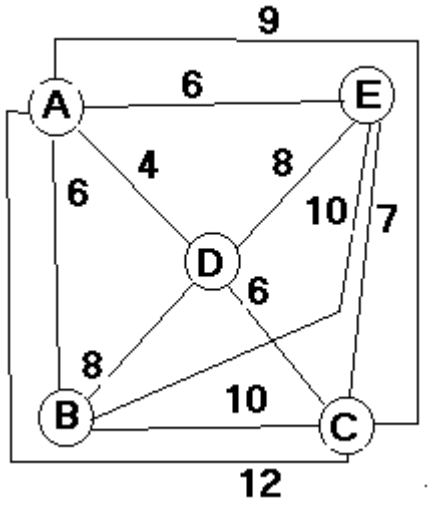
Dependiendo de la estrategia de control que se utilice para seleccionar reglas durante la resolución del problema puede mejorarse el rendimiento al darle preferencia a las reglas de casos especiales.

{PRIVADO }3.1.5 El vendedor viajero{TC \l 33 ".1.5 El vendedor viajero"}

Un viajero debe visitar cinco ciudades, no debe pasar por una ciudad más de una vez y retornar a la ciudad de donde partió (ver figura 3.5); existe un tren entre cada par de ciudades y la distancia que se recorre se indica en el grafo. El problema es encontrar la distancia mínima de la ruta para visitar todas las ciudades sin pasar por una más de una vez.

Para exponer este tipo de problema se especifica lo siguiente:

La BC será una lista de ciudades visitadas. Así la BC inicial es descrita por (A); como no se permite más de una vez una ciudad excepto al recorrer todas, A puede volverse a



llamar.

Figura 3.5. *Conexión de ciudades.*

Las reglas corresponden a las decisiones:

a) ir a la ciudad próxima A;
b) ir a la ciudad próxima B;
c) ir a la ciudad próxima C;
d) ir a la ciudad próxima D;

e) ir a la ciudad próxima E.

Una regla de la BC no es aplicable si no hay transformación a un estado nuevo del sistema. Así, la regla *ir a la ciudad próxima A* sólo se aplica hasta que se haya llamado todas las ciudades.

Cualquier BC empezando y finalizando con A y nombrando todas las ciudades satisface la condición de terminación.

La figura 3.6 muestra parte del árbol que puede generarse para visualizar las soluciones al problema del vendedor viajero.

El árbol permite explorar las posibles rutas y así hallar la que tenga menor longitud o costo. Esta estrategia funciona en la práctica para listas con pocas ciudades, pero se colapsa rápidamente conforme aumente el número de ciudades. Si existen N ciudades, el número de rutas diferentes entre ellas es (N-1)!; el tiempo total empleado para completar la búsqueda es proporcional a N! Si se asume que hay 10 ciudades, el tiempo o pasos es de 3´628.800, un número bastante grande. Si el vendedor tiene que visitar 20 ciudades, solucionar el problema puede gastar más tiempo que el que emplea realmente el vendedor al visitar las ciudades.

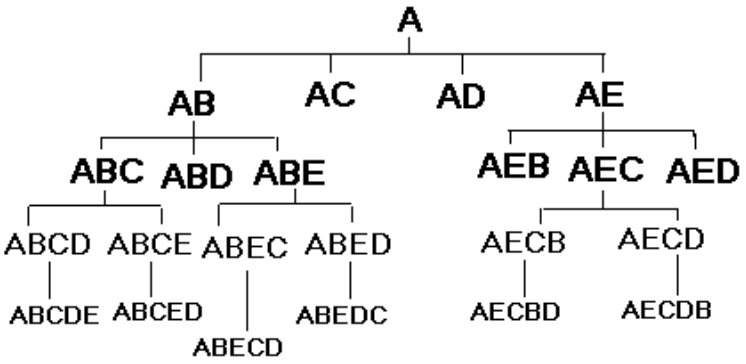


Figura 3.6. *Árbol de solución de rutas.*

{PRIVADO }3.1.6 El análisis sintáctico{TC \l 33 ".1.6 El análisis sintáctico"}

Otro problema que puede desearse solucionar por un SP aproximado es determinar si una secuencia de símbolos forman una sentencia de un lenguaje, esto es, saber si cumplen las normas de una gramática.

Supongamos que se da una gramática que define un lenguaje, dada cualquier oración o sentencia, determinar si ella pertenece al lenguaje o no.

Problemas

Sea la gramática G[<s>] dada por:

<s> --> [<sus>] <ver> <pre>
<sus> --> [<art>] <nom>
<art> --> el | la | un | una
<nom> --> presidente | Inés | partido
<ver> --> vende | corre | juega | ordena
<pre> --> [<art>] <nom>

Ésta es una gramática muy simple para usar en análisis de sentencias, que para el ejemplo está restringida a pocos elementos, pero nos ayuda para ver la realidad. Para exponer un problema en el análisis sintáctico, se especifica lo siguiente:

La BC consiste de una cadena de símbolos; la BC inicial es una cadena de símbolos a investigar si pertenece o no al lenguaje.

Las reglas de producción forman la gramática del lenguaje. Cada símbolo de la base de conocimiento inicial se remplaza por elementos determinados en la gramática en la parte derecha de cada regla de producción. Una regla no es aplicable si en la BC no hay en la regla de la gramática un lado derecho correspondiente. Una regla, también, puede ser aplicable de diferentes maneras (no para este ejemplo); esto hace que el proceso se desplace por diferentes estados intermedios hasta llegar a la condición de terminación.

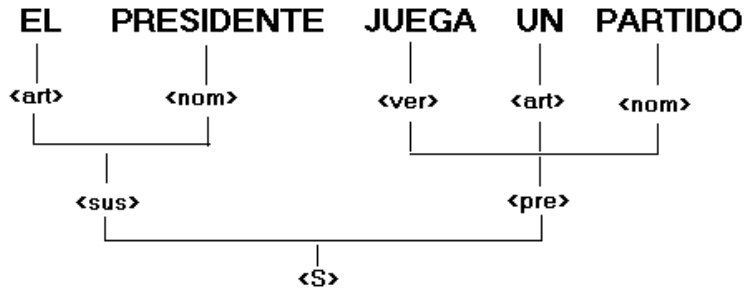


Figura 3.7. *Árbol de análisis sintáctico.*

Si la BC consiste del simple símbolo <s>, satisface la condición de terminación.

Supongamos que deseamos determinar si la cadena de símbolos; **El presidente juega un partido** es o no una sentencia del lenguaje generado por la gramática G[<s>]. Una parte del árbol de este problema se halla en la figura 3.7.

Cada elemento de la cadena tiene un sentido semántico que determina la aplicabilidad de la regla. Esta gramática puede construir un conjunto finito de cadenas; algunas para nuestro lenguaje (español) no tiene sentido pero para G[<S>] sí.

{PRIVADO }3.1.7 El granjero{TC \l 33 ".1.7 El granjero"}

Un granjero quiere cruzar un río llevando consigo una zorra silvestre, un pato gordo y un saco de apetitoso trigo. Por desgracia, su bote es tan pequeño que sólo puede transportar una de sus pertenencias en cada viaje. Peor aun, la zorra, si no se le vigila, se come al pato, y el pato, si no se le cuida, se come el trigo; de modo que el granjero no debe dejar a la zorra sola con el pato o al pato sólo con el trigo. ¿Qué puede hacer?[8]

Una descripción puede ser una matriz de dos filas y tres columnas, la primera fila es la orilla uno y la segunda fila la orilla dos; la columna uno el *zorro*, la columna dos el trigo y la columna tres el pato. El grafo, donde puede hacerse una búsqueda, está dado en la figura 3.8.

El árbol de la figura 3.8 se ha realizado con base en una evolución racional de una descripción de un mecanismo en que se usan los operadores SI, AND, OR y ENTONCES para hacer los cambios pertinentes en la matriz.

En la orilla uno:

SI m(1,1)=1 AND m(1,3)=1, ENTONCES llevar(m(1,3)).
SI m(1,2)=1 AND m(1,3)=1, ENTONCES llevar(m(1,3)).

En la orilla dos:

SI m(2,1)=1 AND m(2,3)=1, ENTONCES regresar(m(2,3)).
SI m(2,2)=1 AND m(2,3)=1, ENTONCES regresar(m(2,3)).

Las reglas anteriores establecen las medidas de seguridad planteadas en el problema.

Llevar y regresar significan hacer los cambios en la columna respectiva.

Por tanto, la representación está dada por:

a) *Léxica*: los símbolos 0 y 1.
b) *Estructural*: restricción de no tener dos 0 o 1 en la misma columna.
c) *Procedimental*: es el mecanismo para responder a reglas de máxima seguridad.
d) *Semántica*: la asociación de correspondencia de la columna 1 con el zorro, la columna 2 con el trigo y columna 3 con el pato, de los unos y ceros en las filas con la pertenencia de la orilla y el granjero que los mueve.

Otra representación puede estar en una tabla que explicite todo el espacio de estados. Basta precisar la situación antes y después de cruzar. El granjero y sus pertenencias tienen que estar en alguna de las dos orillas. La representación del estado debe

[8] *Cambiando el enunciado, el problema puede ser: un arriero se encuentra al borde de un río llevando un puma, una cabra y un bulto de lechuga. Debe cruzar a la otra orilla utilizando un bote en que a lo más cabe una pertenencia. La dificultad es que si no cuida al puma se devoraría la cabra y, si no cuida la cabra ésta se comerá las lechugas ¿Cómo cruzar sin perder ninguna pertenencia?*

entonces indicar en qué lado se encuentra cada uno de ellos (ver tabla 3.2).

Para esto se utiliza el término simbólico con la siguiente sintaxis: e(G,Z,P,T), en que G,Z,P y T son variables que representan al granjero, la zorra, el pato y el trigo, respectivamente, en su posición. Las variables pueden tomar dos valores (i,d), que simbolizan respectivamente el lado izquierdo y el lado derecho del río. Por convención se elige a partir del lado izquierdo. El estado inicial es entonces e(i,i,i,i) y el estado meta es e(d,d,d,d).

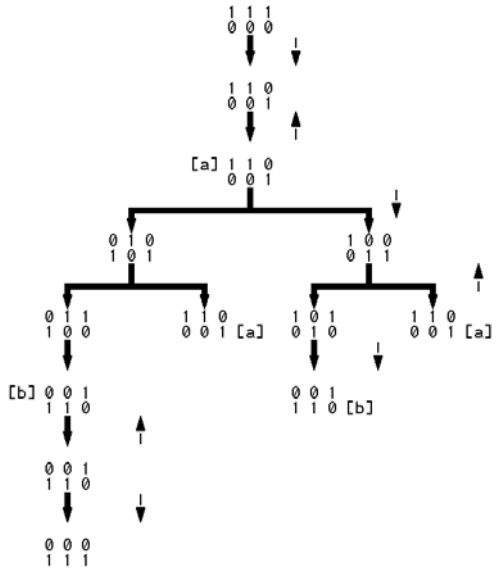


Figura 3.8. *Árbol del granjero.*

{PRIVADO }Tabla 3.2. *Posibles movimientos.*

Estado	**Movimientos**			
	Solo	*Con la zorra*	*con el pato*	*con el trigo*
e(i,i,i,i)	X	x	e(d,i,d,i)	x
E(d,i,d,i)	e(i,i,d,i)	xx	e(i,i,i,i)	xx
e(i,i,d,i)	e(d,i,d,i)	e(d,d,d,i)	xx	e(d,i,d,d)
E(d,d,d,i)	X	e(i,i,d,i)	e(i,d,i,i)	xx
E(d,i,d,d)	X	xx	e(i,i,i,d)	e(i,i,d,i)
e(i,d,i,i)	X	xx	e(d,d,d,i)	e(d,d,i,d)

e(i,i,i,d)	X	e(d,d,i,d)	e(d,i,d,d)	xx
E(d,d,i,d)	e(i,d,i,d)	e(i,i,i,d)	xx	e(i,d,i,i)
E(i,d,i,e)	e(d,d,i,d)	xx	e(d,d,d,d)	xx
E(d,d,d,d)	X	x	e(i,d,i,d)	x

El granjero tiene cuatro acciones posibles: cruzar solo, cruzar con la zorra, cruzar con el pato y cruzar con el trigo. Estas acciones están condicionadas a que ambos objetos del bote estén en la misma orilla y a que no queden solos la zorra con el pato o el pato con el trigo. El estado que resulta de una acción se determina intercambiando los valores i y d para los objetos del bote. El camino que pasa por la siguiente secuencia de estados es una solución del problema:

estado(i,i,i,i)	
cruza con pato	estado(d,i,d,i)
cruza solo	estado(i,i,d,i)
cruza con zorra	estado(d,d,d,i)
cruza con pato	estado(i,d,i,i)
cruza con trigo	estado(d,d,i,d)
cruza solo	estado(i,d,i,d)
cruza con pato	estado(d,d,d,d)

{PRIVADO }3.1.8 Problema de los esposos{TC \l 33 ".1.8 Problema de los esposos"}

Tres señores y sus respectivas esposas llegan a un río cuentan con un bote para cruzar el río, únicamente pueden ir como máximo dos personas. ¿Cómo deben cruzar el río de tal forma que en ningún momento se halle una señora con otro señor sin estar su esposo al lado?

Se emplea un vector de ocho elementos (también puede presentarse por medio de un árbol), que indican cada uno de los hombres (h_i) o mujeres (m_i), un \ que indica el río, lo cual sirve para indicar en qué lado de la orilla del río se hallan, y un 1 (orilla A) o 0 (orilla B) para indicar en qué orilla se halla la barca.

Los diferentes vectores que muestran una solución son:

(h_1m_1 ,h_2m_2 ,h_3m_3\ ;1)	(h_2m_2 ,h_3m_3 \h_1m_1 ;0)
(h_1 ,h_2m_2 ,h_3m_3\m_1 ;1)	(h_1 ,h_2 ,h_3\m_1 ,m_2 ,m_3 ;0)
(h_1 ,h_2m_2 ,h_3\m_1 ,m_3 ;1)	(h_2m_2 \h_1m_1 ,h_3m_3 ;0)
(h_2m_2 ,h_3m_3 \h_1m_1 ;1)	(m_2,m_3 \h_1m_1 ,h_2,h_3 ;0)
(m_1,m_2 ,m_3\h_1 ,h_2,h_3 ;1)	(m_3 \h_1m_1 ,m_2h_2 ,h_3 ;0)
(m_2 ,m_3 \h_1m_1 ,h_2 ,h_3 ;1)	(\h_1m_1 ,h_2m_2 ,h_3m_3 ;0)

¿Cómo sería la solución y la representación si existe además la condición que de las mujeres sólo una sabe manejar la barca?

"Estamos llegando al punto donde los problemas
que debemos resolver se volverán insolubles sin los computadores.
No le temo a los computadores; temo que no haya computadores"
Isaac Asimov.

{PRIVADO }3.2 SOLUCIÓN DE PROBLEMAS{TC \l 23 ".2 SOLUCIÓN DE PROBLEMAS"}

Cuando se desea solucionar un problema dentro del contexto de IA, el primer paso es representar el problema formalmente.[9] ¿Qué implica esto? Implica definir claramente una serie de características de la configuración, una función de costos para determinar cuál es la mejor solución, establecer condiciones o restricciones límite, definir la meta que se desea hallar. Se considera que las metas son estados del mundo.

Las operaciones o acciones son las causantes de la transición entre un estado y otro. La formulación de un problema consiste en decidir qué acciones y estados son válidos. El espacio de estados es un conjunto de estados que pueden alcanzarse mediante cualquier secuencia de acciones. La solución del problema consiste en encontrar una secuencia de acciones posibles que conduzcan a la meta. Al proceso de hallar tal secuencia se conoce como *búsqueda*.

Para ver en realidad útil la IA en la solución de problemas, es necesario determinar una comunicación directa entre el usuario y la máquina, lograr inferencias relevantes y los resultados se obtengan en el menor tiempo posible. Para esto se debe recurrir a los modelos que describan lo más ampliamente posible el sistema. La representación del conocimiento debe ser apropiada y la solución lo más clara posible.

Un espacio de estados se representa con una gráfica (por ejemplo, árbol) donde los nodos son estados de un problema o soluciones parciales de éste y los arcos son acciones u operaciones que se aplican a los nodos para pasar de un estado a otro. En una gráfica con estado inicial y estados finales (o metas), el proceso de búsqueda consiste en encontrar un camino en la gráfica que vaya del estado inicial a un estado final.

La representación de un problema tiene una gran influencia en el esfuerzo necesario para solucionarlo. Obviamente, se prefiere un espacio de estados pequeño. Una representación consiste de las siguientes cuatro partes fundamentales [Winston, 1994]:

[9] *Un problema en realidad es un conjunto de información que la entidad o sistema utiliza para decidir lo que va a hacer. Algunas veces se dice representación de conocimiento para problemas*

- Léxica: que determina qué símbolos son permitidos en el vocabulario de representación.
- Estructural: que describe las restricciones de cómo los símbolos pueden organizarse.
- Procedimental: que especifica los procedimientos de acceso que permiten crear descripciones, modificarlos y responder a preguntas usándolos.
- Semántica: que establece una manera de asociar significados con las descripciones.

Todo problema debe buscar el principio de representación:

Una vez un problema se describe usando una representación apropiada, el problema será siempre solucionado.

Hay muchos ejemplos de rompecabezas, al parecer difíciles, que al representarse apropiadamente tienen espacios triviales[10]. Algunas veces, un espacio de estados puede ser un fracaso, pero por reconocimiento de ciertas reglas que pueden ser descartadas o que pueden ser combinadas para hacer más potentes otras, el espacio cambia y se vuelve muy óptimo. A menudo cuando transformaciones simples no pueden ser realizadas, es posible que una reformulación completa del problema haga un espacio trabajable.

Las situaciones catalogadas como *problemas* no son todas de la misma índole. Hay variaciones significativas respecto al nivel de definición de los estados inicial y final, así como respecto a los tipos de soluciones que se le pueden dar y a la forma como las personas abordan el proceso.

Es de gran importancia aprender a resolver problemas[11], como es muy necesario superar la dimensión memorística y repetitiva en los procesos de instrucción y aprendizaje. Se hace indispensable utilizar métodos adecuados con el fin de pensar al tiempo que se adquiere el conocimiento con el objeto de favorecer el pensamiento creativo y analítico.

Pueden considerarse cuatro categorías de problemas según Reitman [1965], definidas

[10] *¿Deben representarse todos los 362.880 estados por medio de un árbol para un rompecabezas-8 o debe buscarse una representación más óptima? ¿Por medio de reglas?*

[11] *Son muchos los factores que influyen en la solución de un problema. Las indicaciones y pistas tienen poca importancia si no pensamos en ellas de manera correcta. Más que relacionar unas con otras es indispensable buscar maneras nuevas e inusitadas de razonar. El aprendizaje de la solución de problemas requiere ser activo. Sólo se aprende a resolver problemas resolviéndolos, sólo se aprende a pensar, pensando; las pistas, las orientaciones, demostraciones, sugerencias que puede dar quien enseña esta habilidad, se pierden si el aprendiz no se ve envuelto en el proceso de analizar y tratar de resolver problemas. Se trata de una habilidad mental que sólo se adquiere practicándola.*

según el nivel de especificación de los estados inicial y final:

a. Estados inicial y final bien definidos.
b. Estado inicial bien definido y meta pobremente establecida.
c. Estado inicial pobremente definido y meta bien definida.
d. Estados inicial y final pobremente definidos.

Un problema bien definido es más fácil de resolver que uno que no lo esta y que por algo dice el refrán que *al especificar el problema se obtiene la mitad de la solución*.

Los métodos de solución de problemas se basan en tres poderosos paradigmas: generar y probar, análisis de medios y fines, y reducción del problema.

{PRIVADO }3.2.1 Método de generar - probar{TC \l 33 ".2.1 Método de generar - probar"}

El método de generar-probar consiste de dos módulos básicos: uno que genera o enumera todas las soluciones posibles y otro que evalúa las soluciones dadas, las acepta o rechaza. El proceso puede ser:

```
Alg Gen_pro()
 HQ una solución sea aceptada o no hay más soluciones
        generar una solución posible
        evaluar la solución, verificar si es "buena" solución
 FHQ
 SI una solución es aceptada V: ÉXITO
        F: FALLO
 FSI
Fin Gen_pro()
```

Este método se usa frecuentemente para solucionar problemas de identificación; por ejemplo: se le da al sistema una huella y él investiga en la información que tiene con reglas establecidas. No necesariamente, contra conocimiento exacto, se crean hipótesis para indicar posibles soluciones.

Un buen generador debe tener tres propiedades:

a) Ser completo: producir eventualmente todas las soluciones posibles.
b) No ser redundante: no proponer la misma solución dos veces
c) Ser informador: usar la información límite posible restringiendo las soluciones propuestas.

Ejemplo. Cuatro amigos ejercen oficios distintos. Se le preguntó a un compañero por los respectivos oficios y en vez de darnos una respuesta directa nos propuso el

siguiente enigma:

a. Sánchez desea un préstamo del banquero.
b. Castro conoció al chofer cuando le pagó por hacer una carrera.
c. El pintor y Sánchez son amigos pero no han hecho negocios entre sí.
d. Ni el pintor ni Ortíz conocen a Ojeda.
e. El escultor es un gran amigo.

¿Cuál es la profesión de cada uno de ellos?

Solución. Una representación formal facilita un mejor entendimiento. La profesión se asocia en un vector de cuatro posiciones donde se han de colocar los apellidos de las personas, de modo que se cumplan todas las condiciones o pistas.

{PRIVADO }1	2	3	4
Chofer	Banquero	Pintor	Escultor

Se enumeran las pistas, puede ser de la siguiente manera:

* Sánchez no es el banquero	profesión[2] <> Sánchez
* Castro no es el chofer	profesión[1] <> Castro
* El pintor no es Sánchez	profesión[3] <> Sánchez
* Ortíz no es el pintor	profesión[3] <> Ortíz
* Ojeda no es el pintor	profesión[3] <> Ojeda

Se formalizan ahora condiciones que no están explícitas en el texto, así:

1. **SI** Castro es el pintor **ENTONCES** el chofer no puede ser Ojeda, esto por la pista 2, Castro pagó al chofer, y la pista 4, que el pintor no conoce a Ojeda.
e.d. **SI** Castro = profesión[3] **ENTONCES** profesión[1] <> Ojeda.

2. **SI** Castro es el pintor **ENTONCES** el chofer no puede ser Sánchez, esto por la pista 3, Castro y Sánchez no han hecho negocios.
e.d. **SI** Castro = profesión[3] **ENTONCES** profesión[1] <> Sánchez.

La generación de posibles soluciones llevaría a tener mayor número de condiciones y en cada una de ellas aplicar los formalismos anteriores para aceptar o rechazar cada asignación.

Puede limitarse el número de asignaciones al aplicar formalismos más directos para eliminar algunas posibilidades.

Con este propósito se forma la tabla 3.3:

Tabla 3.3 *Resultados básicos de formalización.*

{PRIVADO}	**1= chofer**	**2 = banquero**	**3 = pintor**	**4 = escultor**
Sánchez		**NO**	**NO**	
Castro	**NO**			
Ortíz			**NO**	
Ojeda			**NO**	

Ahora, Castro debe ser el pintor, por tanto, no puede tener ninguna de las otras profesiones; las marcamos con X.

{PRIVADO}	**1= chofer**	**2 = banquero**	**3 = pintor**	**4 = escultor**
Sánchez		**NO**	**NO**	
Castro	**NO**	**X**	**SI**	**X**
Ortíz			**NO**	
Ojeda			**NO**	

Se enumera un conjunto de posibilidades por combinatoria de los apellidos en las profesiones,

{PRIVADO}	**1= chofer**	**2 = banquero**	**3 = pintor**	**4 = escultor**
1	**Sánchez**	**Ortíz**	**Castro**	**Ojeda**
2	**Sánchez**	**Ojeda**	**Castro**	**Ortíz**
3	**Ojeda**	**Ortíz**	**Castro**	**Sánchez**
4	**Ortíz**	**Ojeda**	**Castro**	**Sánchez**

1. Establece **SI** Castro = profesión[3] **ENTONCES** profesión[1] <> Ojeda
   por tanto se elimina 3.

2. Establece **SI** Castro = profesión[3] **ENTONCES** profesión[1] <> Sánchez;

así, se elimina 1 y 2 y queda como solución

{PRIVADO}	**1= chofer**	**2 = banquero**	**3 = pintor**	**4 = escultor**
4	**Ortíz**	**Ojeda**	**Castro**	**Sánchez**

{PRIVADO }3.2.2 Método de análisis de medios y fines{TC \l 33 ".2.2 Método de análisis de medios y fines"}

Consiste en distinguir cuáles son los *medios* disponibles (los operadores) y los *fines* (los estados meta) a partir de un estado actual. El estado actual es donde se halla el problema en un momento dado. El estado meta corresponde a donde desea llegar el sujeto y el interés es hallar una secuencia de transiciones que nos lleva del estado actual al estado meta o al menos a otro estado desde donde se lograra llegar al estado meta. El procedimiento es analizar las diferencias que hay entre el estado actual y el estado meta. Una vez detectadas. Se accede a la representación en la que se estipula las acciones a seguir. Puede ocurrir que un operador no se pueda aplicar directamente al estado; en cuyo caso, se establecen submetas en que dicho operador es aplicable. La aplicación de un operador garantiza la reducción de diferencias.

El proceso puede ser:

Alg Ana_fin()
Estado actual = RAÍZ
HQ META es alcanzada o ninguna transición es posible.
describir el estado actual, el estado meta y las diferencias entre ellos.
usar esas diferencias y seleccionar un paso BUENO
usar el paso BUENO y actualizar estado actual
FHQ
SI META es hallada V: ÉXITO
F: FALLO
FSI
Fin Ana_fin()

Debe considerarse:

- Establecer claramente cuáles son las condiciones que permiten aplicar cada uno de los operadores.
- Ser fácilmente accesibles los resultados causados por la utilización de cada uno

de los operadores.

- Las diferencias dependen fuertemente del dominio.
- La búsqueda de la solución es una combinación de *búsqueda de retroceso* (determinar y resolver las diferencias para encontrar estados en los que los operadores puedan ser aplicados) y de *encadenamiento hacia adelante*

{PRIVADO }3.2.3 Método de reducción del problema{TC \l 33 ".2.3 Método de reducción del problema"}

Algunas veces, es posible convertir metas difíciles en una o más submetas más fáciles. Cada submeta puede dividirse en otras de más bajo nivel. Este método busca reconocer las submetas como metas, sin olvidar la meta por hallar.

El proceso puede ser:

```
Alg Red_pro()
 estado_actual = RAÍZ
 SI estado_actual = META V: "la meta se halló", ÉXITO
          F: Generar submetas
           MQ existan submetas Red_pro(submeta)
           FMQ
 FSI
Fin Red_pro()
```

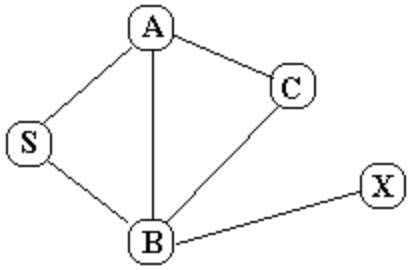


Figura 3.9. *Ejemplo de red.*

Podría decirse que existen dos tipos de problemas: uno, al que es posible asociar un procedimiento -que en principio no sabemos cuál es- para hallar una solución; y dos, los que se presentan en la vida diaria del mundo real, que es realizar tentativas para hallar la solución, búsqueda de metas.

{PRIVADO }3.3 BÚSQUEDA DE METAS{TC \l 23 ".3 BÚSQUEDA DE METAS"}

La búsqueda de metas tiene como objetivo primordial hallar una trayectoria en una red. Encontrar una trayectoria que nos lleve del nodo (inicial) S, al nodo meta X (ver figura

3.9).

Para hallar una solución se consideran todas las trayectorias posibles, eliminando, naturalmente, las trayectoria que formen ciclos (ver figura 3.10).

Los procesos de búsqueda tienen sentido en problemas que reúnen una serie de características:

* Cabe la posibilidad de asociar un conjunto de estados a las diferentes situaciones.
* Existen estados iniciales desde donde se empezará la búsqueda.
* Existen operadores que al aplicar alguno a un estado produce otro estado.
* Existe al menos un estado meta o estado solución.

La más obvia manera de hallar una solución cuando se tiene una representación gráfica es mirar los posibles caminos, analizarlos y evaluarlos para escoger el mejor. Pueden organizarse todos los posibles caminos desde un nodo inicial a otro nodo meta o no; deben ser evaluados (recorridos) o ignorados.

Existen algoritmos para encontrar un camino solución en un espacio de estados. Un esquema general puede ser:

```
Alg Buscar()
 ABIERTO <- [estado inicial]
 CERRADO <- []
 MQ ABIERTO <> vacío
          remover un estado X de ABIERTO
          SI (X = META) V: ÉXITO
           F: generar sucesores del estado X
           agregar el estado X a CERRADO
           eliminar sucesores que ya estén en ABIERTO o CERRADO
           agregar resto de sucesores a ABIERTO
          FSI
 FMQ
 retorne FALLO
Fin Buscar()
```

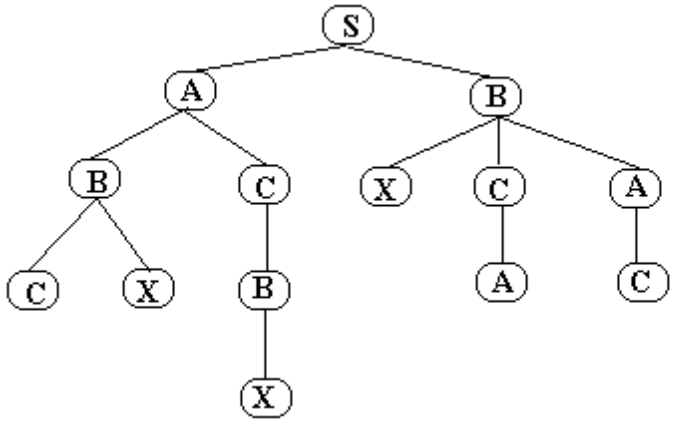


Figura 3.10. *Árbol de la red de la figura 3.9.*

El conjunto ***ABIERTO*** contiene a los estados que todavía no han sido visitados, no se ha verificado si son estados meta. El conjunto ***CERRADO*** contiene a los estados visitados[12].

Básicamente existen dos tipos de búsqueda de metas: óptimas y básicas. Dentro de las búsquedas básicas se tienen las no informadas y las informadas (ver figura 3.11).

Sin información (ciega):
- *Depth-first* (profundidad principalmente),
- *Breath-first* (de amplitud principalmente),
- No determinística.

Con información (heurística):
- *Hill Climbing*,
- *Beam Search*,
- *Best first Search*.

Búsquedas óptimas (El mejor camino):
- *British Museum*,
- *Branch and Bound*,
- *Programación dinámica*,
- *A**.

[12] *Al considerar solo a los sucesores que no han sido previamente generados se evita entrar en ciclos. Dependiendo del orden en que se visiten los estados del conjunto* ***abierto*** *se obtienen distintos tipos de recorrido.*

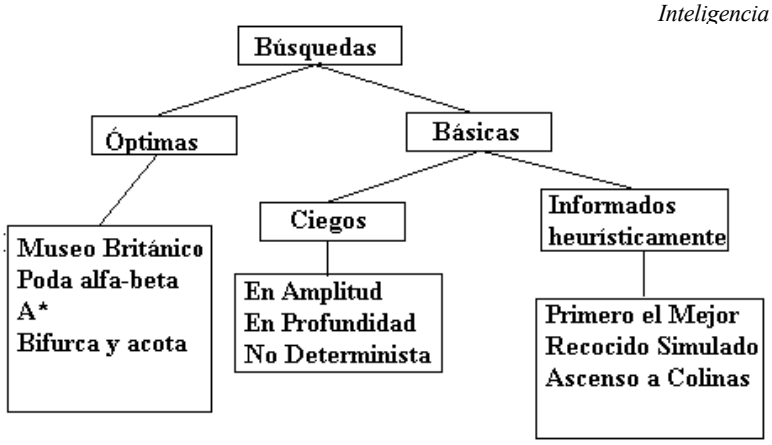


Figura 3.11. *Posibles búsquedas.*

{PRIVADO }3.3.1 Búsqueda en profundidad principalmente{TC \l 33 ".3.1 Búsqueda en profundidad principalmente"}

La búsqueda en profundidad principalmente (en Inglés, *depth-first search*) es una simple manera de encontrar un camino por los descendientes que tiene un nodo. Entendiendo que las alternativas válidas son en orden izquierda a derecha, lo primero que se hace es recorrer rápidamente el árbol a través de bifurcaciones izquierdas.

Un algoritmo simple es:

Lista = RAÍZ.
HQ Lista = vacío o se alcance META.
Tomar elemento de Lista
SI elemento = META V: ÉXITO.
F: eliminar elemento de Lista
agregar sus hijos al principio de Lista.
FSI
FHQ

Para ilustrarla, supongamos que somos soldados que nos introducen a un laberinto y tenemos la posibilidad de marcar los caminos recorridos. En el momento de ver una bifurcación elegimos siempre la alternativa más hacia la izquierda, realizando este procedimiento hasta encontrar la salida o un camino ciego. Si hallamos la salida, qué alegría. Si es un punto ciego nos devolvemos a la última bifurcación y exploramos otra alternativa; si ha sido la última alternativa de esta bifurcación, nos devolvemos a una bifurcación anterior, repitiendo este procedimiento hasta encontrar la salida (meta).

Un algoritmo más completo podría ser:

Alg Buscar_prof()

```
Problemas
 ABIERTO <- [RAÍZ]
 CERRADO <- []
 MQ ABIERTO <> vacío
    tomar primer estado X de ABIERTO
    SI X = META
           V : indicar camino
            ÉXITO
           F : generar sucesores de X
              SI no existen sucesores V: retornar a nodo anterior
                F:  eliminar sucesores que ya estén en ABIERTO o CERRADO
                       agregar resto de sucesores al principio de ABIERTO
             FSI
            agregar X a CERRADO
    FSI
 FMQ
 SI META no hallada V: "no existe camino", FALLO
 FSI
Fin Buscar_prof()
```

Si el conjunto ABIERTO se maneja como una lista LIFO, es decir, como un *stack*, siempre se estará visitando primero los últimos estados en ser generados.[13]

La búsqueda en profundidad es una buena idea cuando se conoce que hay caminos que llegan al nodo META; es decir, existen metas a buscar. En contraste, la búsqueda en profundidad es una mala idea si hay muchos niveles y se desconoce si hay metas. El hallar una solución no indica nada de la optimalidad.

Considerando que la cantidad promedio de sucesores de los nodos visitados es B (en inglés, *branch factor* y en español factor de ramificación), y suponiendo que la cantidad máxima alcanzada es n, el algoritmo tiene una complejidad en tiempo de $O(B^n)$. Si no se considera el conjunto cerrado, una complejidad en estados de O(B*n)

```
Lista <- [RAÍZ]
HQ halle Meta o devuelva fallo
          SI Lista = vacío V: FALLO
           F: tomar primer nodo de Lista, sea XX
            SI (profundidad(XX) =lp) V: retornar valor
              F: generar sucesores de XX
               SI (algún sucesor=META) V: ÉXITO
                     F: Incluir sucesores al inicio de Lista
               FSI
             FSI
          FSI
```

[13] *El algoritmo tiene la tendencia de profundizar la búsqueda en una rama antes de explorar ramas alternativas*

```
FHQ
```

3.3.2 Búsqueda en amplitud principalmente (nivel)

La búsqueda en amplitud (en inglés, *breadth-first search*) chequea los nodos existentes por las alternativas que se presentan en el mismo nivel, es decir: primero toma en cuenta todas las rutas de longitud 1, luego las de longitud 2, etcétera; en general, todos los nodos que están en la profundidad m del árbol de búsqueda se expanden antes de los nodos que estén en la profundidad m+1.

Un algoritmo simple sería:

```
Lista = RAÍZ.
HQ Lista = vacio o se alcance META.
 SI primer elemento = META V: ÉXITO.
        F: eliminar el primero de Lista
        agregar sus hijos al final de Lista.
 FSI
FHQ
```

Como ejemplo, supongamos que debemos buscar un objeto que se halla en un baúl en el altillo. Al llegar allí vemos varios baúles grandes, procedemos a buscar uno por uno. Si hallamos el objeto, nos vamos a entregarlo. Pero si no, de seguro nos hemos percatado que entre cada baúl hay otros baúles (más pequeños). Realizamos el procedimiento de buscar en cada uno de ellos olvidándonos de los anteriores, si hallamos el objeto, qué felicidad. De lo contrario, debemos seguir buscando en otros baúles que van apareciendo dentro de los anteriores: debemos suponer que no existe limitante alguna del número de baúles que podría contener alguno de ellos y que siempre pueden haber otros más y más pequeños.

Un algoritmo más completo estaría dado por:

```
Alg Buscar_ampl()
ABIERTO <- [RAÍZ]
CERRADO <- []
MQ ABIERTO <> vacío
        remover el primer estado X de ABIERTO
        SI (X = META) V: ÉXITO
                F: generar el conjunto de sucesores de X
                agregar X a CERRADO
                eliminar sucesores que ya estén en ABIERTO o CERRADO
                agregar el resto de sucesores al final de ABIERTO
        FSI
 FMQ
```

Problemas
FALLO
Fin Buscar_ampl()

Si el conjunto **ABIERTO** se maneja como una lista LIFO, es decir, como una cola, siempre se estará visitando primero los estados en ser generados. Si existe una solución, la búsqueda en amplitud garantiza encontrarla. La búsqueda en amplitud es una buena idea cuando se conoce que existen nodos META en diferentes niveles y un número razonable de ellos. En contraste, la búsqueda en amplitud es una mala idea si el árbol es muy ancho o son pocos los nodos META.

Tómese en cuenta un espacio en el que la expansión de cada uno de los estados produce m nuevos estados. Se dice que el factor de ramificación de estos estados es B. La raíz del árbol de búsqueda genera B nodos en el primer nivel; cada uno de éstos genera B nodos más, con un total de B^2 en el segundo nivel. Cada uno de éstos genera B nodos más, produciendo B^3 nodos en el tercer nivel, etcétera. Supóngase ahora que en la solución de este problema hay una ruta de longitud j. Así pues, la cantidad máxima de nodos expandidos antes de hallar la solución es: $1+B+B^2+B^3+...+B^j$. El algoritmo tiene una complejidad en tiempo t espacio de $O(B^n)$.

Lista <- [RAÍZ]
MQ Lista <> vacío o se halle META
tomar primer nodo de Lista, sea XX.
Generar sucesores de XX.
Aplicar operador a XX, obtener nuevo estado, crear puntero a predecesor XX.
SI nuevo_estado=META V: ÉXITO.
F: Incluir nuevo estado al final de Lista.
FSI
FMQ

{PRIVADO }3.3.3 Búsqueda iterativa en profundidad{TC \l 33 ".3.3 Búsqueda iterativa en profundidad"}

La búsqueda iterativa en profundidad (en inglés, *iterative deepening*) se basa en combinar las buenas características de la búsqueda en profundidad, en cuanto a la complejidad en espacio, con las ventajas de la búsqueda en amplitud. Esto se logra haciendo una búsqueda en profundidad acotada por una profundidad máxima i que se hace variar iterativamente de 1 en adelante con incrementos de 1. Si el estado objetivo más cercano se encuentra a una profundidad n, se hacen n búsquedas en profundidad, a nivel 1, a nivel 2, hasta el nivel n.

La complejidad en espacio de esta búsqueda es de O(B*n). En tiempo, la complejidad es la suma de las complejidades en tiempo de cada una de las búsquedas en profundidad acotadas que fueron necesarias, es decir, $O(B^n)$.

{PRIVADO }3.3.4 Búsqueda no determinística{TC \l 33 ".3.4 Búsqueda no determinística"}

Se ha buscado una mezcla de las búsquedas en profundidad y en amplitud cuando el factor de bifurcación es grande, los caminos son muy largos y puede saberse de antemano que existe buen número de caminos.

```
Alg Buscar_nodeter()
 ABIERTO <- [RAÍZ]
 CERRADO <- []
 HQ ABIERTO = vacío o se halle META.
        Anexar nodo a ABIERTO aleatoriamente (hijo o hermano).
        Aplicar profundidad o anchura.
 FHQ
 SI META es encontrada V: ÉXITO.
        F: FALLO.
 FSI
Fin Buscar_nodeter()
```

La búsqueda no determinística es una buena idea cuando se conoce que existen muchos nodos META. En contraste, la búsqueda de anchura es una mala idea si existen pocos nodos META.

{PRIVADO }3.4 BÚSQUEDAS INFORMADAS{TC \l 23 ".4 BÚSQUEDAS INFORMADAS"}

Existen problemas especialmente complejos en los que el siguiente paso correcto por tomar no puede determinarse a priori. Para ello se emplean heurísticas de búsqueda que permiten encontrar una posible solución, sin que llegue a ser la mejor necesariamente.

Existen problemas en los cuales debe optimizarse unos parámetros; por ejemplo:

1. Al entrar a una pieza, se halla que la temperatura no es confortable, se decide ajustar el termostato.
2. Las imágenes en el televisor están distorsionadas por el tiempo. Se decide ajustar con los controles de color, tinte y brillo para una mejor imagen.
3. Se está subiendo una montaña cuando una densa nube nos cubre dejándonos con poca visibilidad, no se cuenta con mapa o guía para seguir, pero se tiene una brújula, un altímetro y una determinación de ir hasta el final.
4. Establecer la estrategia óptima para ganar (o perder poco) en un juego.
5. Hallar el camino de menor costo y la mejor ruta para ir de un lugar a otro.

Samuel Johnson.

{PRIVADO }3.4.1 Búsqueda heurística{TC \l 33 ".4.1 Búsqueda heurística"}

Para resolver muchos problemas difíciles (explosión combinatoria), es necesario muchas veces llegar a un compromiso de los requerimientos de movilidad y sistematicidad y construir una estructura de control que no necesariamente garantiza encontrar la mejor respuesta, sino que casi siempre encuentra una buena respuesta. El origen de ésta búsqueda está en el análisis del comportamiento humano.

En casi todos los espacios de estados, existe información que permite guiar los procesos de búsqueda, aunque esa información no es perfecta y con certidumbre ayuda a resolver los problemas a travéz de un conjunto de estados sin que tenga definido un camino, esto se denomina heurística[14].

Una técnica heurística mejora la eficiencia del proceso de búsqueda sacrificando, usualmente, exhaustividad. Las consideraciones que sirven de soporte a un proceso de búsqueda heurística son:

- Rara vez se requiere, en realidad, una solución óptima. Una buena aproximación, normalmente, sirve muy bien.
- A pesar que una aproximación heurística no puede resultar muy buena, en el peor de los casos, raras veces aparecen los peores casos en la práctica.
- El tratar de comprender por qué un heurístico funciona o por qué no funciona,

[14] *La palabra heurística viene de la palabra griega HEURISKIN que significa descubrir, que también es origen de EUREKA. El termino técnico* ***heurística*** *ha adoptado diversas connotaciones. George Poyla cuando publicó el libro* ***How to solve it*** *(Cómo resolverlo) aplicaba el termino al estudio de métodos para descubrir e inventar técnicas para la resolución de problemas. Algunas personas utilizan el término* ***heurístico*** *como opuesto al algorítmico. Newell, Shaw y Simon declaran: "Cuando un proceso afirma poder resolver un problema determinado, pero no ofrece ninguna garantía de ello, se dice que es la heurística de dicho problema". Otras veces, se considera a la* ***heurística*** *como una* ***regla practica*** *utilizada por los expertos para generar buenas soluciones sin tener que desarrollar un exhaustivo proceso. En general, cuando se encuentra como sustantivo, se refiere al arte o la ciencia del descubrimiento. Cuando aparece como adjetivo, se refiere a estrategias, reglas o incluso a conclusiones. En la IA, las heurísticas han tenido un papel fundamental ya que gracias a ellas se realizan procesos que utilizando ejemplos y estrategias la máquina aprende de sus errores. La búsqueda heurística es "un conjunto de reglas prácticas" que exploran alternativas en la solución de problemas, donde la solución se descubre por la evaluación del progreso logrado. Las heurísticas tratan de resolver problemas difíciles eficientemente. No garantizan encontrar la solución óptima, pero sí buenas aproximaciones con menos esfuerzo. Los problemas que son objeto de esta técnica pueden ser de dos tipos: (a) problemas de solución parcialmente conocida, como por ejemplo los casos de diagnóstico en medicina; (b) problemas intratables de solución conocida. Son los que no pueden resolverse completamente por la complejidad implicada, pero que sí se conoce el método para resolverlos, por ejemplo, el ajedrez. Las heurísticas son adecuadas para problemas en los que para alcanzar un nivel de principiante no es necesario realizar un proceso excesivamente complejo. El fin del modelo no tiene porque ser exactamente obtener una meta óptima, también puede ser reducir el costo o el tamaño de la solución.*

a menudo conduce a una mejor comprensión del problema.

Las búsquedas anteriores hacen una búsqueda ciega y exhaustiva, lo que presenta ciertos problemas cuando el espacio de estado es grande. Esto se mejora expandiendo primero los estados con mayores expectativas de hallar la solución, lo cual implica emplear una función de evaluación heurística (**feh**) que lleve a darle prioridad a estos estados.

El conjunto **ABIERTO** se maneja como una cola de prioridad ordenada según una función heurística que aporta un conocimiento adicional sobre el problema abordado.

```
Alg Buscar_heu()
ValorHeurístico(estado_inicial)=0
ABIERTO <- [estado_inicial]
CERRADO <- []
MQ (ABIERTO <> vacío)
        remover el primer estado X de ABIERTO
        SI (X = META) V: indicar camino hasta X, ÉXITO
                F: generar los sucesores de X
                PC Y en sucesores
                        asignar a Y valor heurístico
                        SI (Y no esta en ABIERTO ni en CERRADO)
                                V: agregar (Y,VH(Y)) a ABIERTO
                                F: SI Y está en ABIERTO
                                V: SI nuevo camino a Y es más corto
                            V: actualizar camino en ABIERTO
                           FSI
                         F: SI Y está en CERRADO
                            V: SI nuevo camino a Y es más corto
                                V: remover el estado (Y,VH(Y)) del conjunto CERRADO
                                 agregar el estado (Y,VH(Y)) a ABIERTO
                             FSI
                           FSI
                        FSI
                FSI
               FPC
        agregar el estado (X,VH(X)) a CERRADO
        reordenar la lista ABIERTO según valores heurísticos
        FSI
 FMQ
 FALLO
Fin Buscar_heu()
```

Además de utilizar una cola de prioridad, el algoritmo actualiza los caminos almacenados en *ABIERTO*. Cuando se llega a un estado en *CERRADO* por un camino más corto, habría que trasmitir esta información a todos sus sucesores. Sin embargo,

algunos de estos sucesores pueden haber sido generados o posteriormente actualizados por otro camino que hasta el momento parecía más corto, por lo cual estarían desvinculados del ancestro que esta considerándose.

Existen por lo menos tres formas para medir la eficiencia de una búsqueda:

Primera: ¿Permite encontrar una solución?
Segunda: ¿Es una buena solución?
Tercera: ¿Cuál es el costo de búsqueda correspondiente al tiempo y memoria para hallar la solución?

Cada uno de los problemas conforman una abstracción en la cual hay parámetros ajustables y cantidad medible que dice acerca de la calidad y del desarrollo asociado a un escenario particular. Una heurística bien diseñada desempeña un papel importante de guía eficiente en la búsqueda de una solución, que en ocasiones proporciona buenas estimaciones sobre si una ruta es adecuada o no.

La clave de las heurísticas es ahorrar tiempo y/o espacio de almacenamiento evitando recorrer muchos caminos inútiles, no consiste en eliminar una parte del espacio de búsqueda, sino en introducir información adicional que guíe el recorrido realizado (introduciendo reglas de control en el proceso de búsqueda. Las reglas de control heurísticas en función del conocimiento reflejado, pueden ser de dos tipos:

- Dependiente del dominio: para seleccionar algunas de las situaciones alcanzadas y también para alcanzar la siguiente acción aplicable sobre aquella.
- Conocimiento general disponible sobre el funcionamiento del solucionador.

Desde tiempos remotos, las facultades intelectuales del ser humano se han ocupado en creación y desarrollo de juegos, en ocasiones difíciles. Juegos como el ajedrez[15], las damas, etcétera, requieren una gran abstracción; esta abstracción es lo que hace interesante los juegos para la IA. La clase de juegos por considerar son aquellos en los que intervienen dos jugadores con información completa. Hay dos jugadores adversarios quienes alternan sus movimientos, cada uno ve las jugadas del oponente y las propias, sean éxito o fallas.

Desde la perspectiva de un problema de búsqueda, se tiene un árbol de juego que es una representación explícita de todos los posibles caminos de una partida. El nodo raíz es la posición inicial del juego, sus sucesores son las posiciones que el primer jugador pueda alcanzar, los sucesores de estos nodos son las posiciones resultado de la replica del segundo jugador, y así sucesivamente. Los nodos terminales representan posiciones

[15] *Los primeros investigadores en IA eligieron para su trabajo el ajedrez por varias razones. Un computador capaz de jugar ajedrez sería la prueba que las máquinas pueden realizar algo que requiere inteligencia. Además, es un juego de reglas sencillas y el juego se representa con una búsqueda a través de un espacio de posibles posiciones del juego.*

ganadoras, perdedoras o tablas. El procedimiento de propagación hacia la raíz de los valores estimados para las hojas del árbol más conocido es el MINIMAX y el truco para simplificar el árbol es el ALFA-BETA.

{PRIVADO }3.4.2 Minimax{TC \l 33 ".4.2 Minimax"}

Muchos juegos interesantes involucran dos componentes, ninguno de los cuales tiene control completo sobre el desarrollo de las movidas. Entonces así no tiene sentido buscar un camino óptimo desde el estado inicial hasta un estado objetivo, en particular porque los oponentes no comparten el mismo objetivo (se supone que ambos desean ganar, pero si uno gana el otro pierde).

Consideremos el caso general de un juego con dos participantes. Los dos jugadores participan por turnos en el juego, el resultado de cada movimiento sitúa a cada jugador en un nuevo *estado*; esta forma de interpretar el árbol que representa el juego, permite considerar a la vez todas las respuestas posibles ante una jugada seleccionada. Los *nodos hojas* representan las tres situaciones posibles: ganar, perder o empatar el juego en la rama que conduce de la raíz a dicho nodo. El número de movimientos posibles que deben considerarse para garantizar una estrategia ganadora es generalmente intratable. Surge por tanto el problema de la explosión combinatoria. Por ser problemas intratables se simplifica el proceso determinando el estado de los nodos hasta un cierto *límite de profundidad* en el que el valor del nodo es el valor devuelto por la aplicación de la **feh**. El método consiste en prever el mayor número de jugadas posibles y actuar en consecuencia. Se considera que el adversario va a realizar la mejor de las opciones posibles cuando a él le toque jugar. Los nodos son de dos tipos: MAX y MIN. Max es considerado el que sabe más del juego. Max busca una secuencia de movimientos que lo conducen a un estado meta que le haga ganador. Sin embargo, Max no está sólo y Min por su parte tratará de ganar. Por eso, Max debe hallar una estrategia que lo conduzca al estado meta independientemente de lo que haga Min.

El objetivo del jugador Max es maximizar el valor de la función f de evaluación que mida la proximidad estimada a la meta de una situación dada y, así mismo, Min refleja el modo de actuar del adversario con respecto al objetivo del nodo Max; es decir, realiza aquel movimiento que haga más pequeño el valor de f.

Los valores de los estados del juego son una medida heurística del mérito de la configuración evaluada que no necesariamente corresponde a una estimación de la distancia al objetivo. Los valores deben ser simétricos, ya que lo que es bueno para un jugador es malo para su contrincante.

- El estado inicial (BC) contempla la indicación de quien juega y una posición en el tablero.
- Los operadores determinan qué jugadas están permitidas.

- La meta que define el termino del juego.
- La función de utilidad que asigna un valor a la jugada.

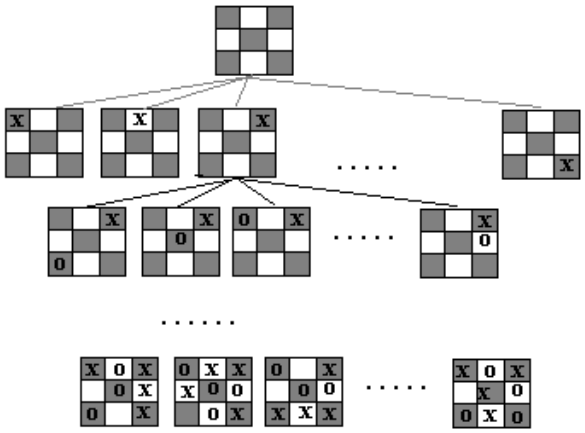


Figura 3.12. *Juego de tres en línea.*

En la figura 3.12 se muestra parte del árbol de búsqueda correspondiente al juego de *tres en línea*[16].

El juego de *tres en línea* consiste de una matriz 3*3 en donde cada uno de dos jugadores puede en cada jugada colocar su marca. Se considera ganador al jugador que coloca primero una fila, columna o diagonal con su marca.

Para optimizar la estrategia, cada jugador ha de tener en cuenta todas las posibles respuestas de su adversario antes de realizar una jugada.

El problema no es hallar un camino completo, sino encontrar la mejor movida dado el estado actual del juego. La solución para determinar cada movida es hacer una búsqueda hasta una profundidad P dada, y evaluar las posiciones encontradas en el último nivel. Cuanto mas profundo es P, mayor es el horizonte y más seguros los elementos de decisión para elegir la mejor jugada (se prevén las consecuencias de la jugada a más largo plazo).

En cada nivel del árbol de juego, cuando un nodo corresponda a una jugada de MAX, el valor de este nodo será el máximo de los valores de sus descendientes, y cuando el nodo corresponda a una jugada de MIN, el valor de este nodo será el mínimo de los valores descendientes. Por cada una de las jugadas de MAX puede obtenerse un valor que corresponde a la posición (función de evaluación) que puede esperar MAX contra toda estrategia de MIN[17].

---

[16] *A pesar de la utilización de la heurística, es imposible estudiar el árbol del juego completo, ni siquiera una parte significativa.*

[17] *La mejor opción de MAX se le conoce como decisión minimax, se obtiene el máximo de la utilidad con*

Una forma de construir una función de evaluación es usando la expresión SUM $w_i * f_i$, donde w_i son los pesos de cada rama en el árbol, y los f_i son los rasgos de una posición particular.

Si en el juego de tres en línea se toma la siguiente función de evaluación

f(s) = (#filas o #columnas o #diagonales abiertas por MAX)
- (#filas o #columnas o #diagonales abiertas por MIN)

Entonces:

Si m es posición ganadora para MAX, f(m)= inf.
Si c es posición perdedora para MAX, f(c)= -inf.

Una componente necesaria en un procedimiento para propagar los valores calculados es la función de evaluación de las ramas del árbol hasta un nivel dado. El procedimiento clásico es la regla mínima:

1. El valor g(j) de un nodo terminal es igual a su función de evaluación heurística E(j).
2. Si k es un nodo Max, g(j) es igual al valor máximo de todos sus sucesores.
3. Si k es un nodo Min, g(j) es igual al valor mínimo de todos sus sucesores.

Para cada nodo:

a. Generar los sucesores de k; $k_1, k_2, ..., k_n$.
b. Evaluar $g(k_j)$ de izquierda a derecha.

```
Alg Minimax()
 Generar todo el árbol del juego
 Búsqueda fundamentalmente en profundidad
  HQ nivel sea nodo raíz
   Aplicar función de utilidad a cada estado terminal
   utilizar la utilidad para calcularla en nodos del siguiente nivel superior
  FHQ
Fin Minimax()
```

Una versión más completa del algoritmo de Minimax es:

```
Alg()
// Entrada: nodo,lado
// Salida: Valor_posición
// lado=1 (Max). lado=-1 (Min)
Iniciar: prof=1; E(prof)=listajugadas(nodo,lado); lado=1; ev(1)=-(lado)*inf
MQ prof>=1
```

---

el supuesto que el oponente jugara para reducirla al mínimo.

```
        Problemas
        MQ E(prof)<>vacío
         jugada=primera(E(prof));
         nodo=jugar(nodo,jugada);
         SI prof<>(profmaxima-1)
                  V: prof=prof+1
                    lado=-lado
                    E(prof)=listajugadas(nodo,lado)
                    ev(prof)=-(lado)*inf
                  F: SI lado=1
                           V: ev(prof)=MAX[ev(prof),valor(nodo)]
                           F: ev(prof)=MIN[ev(prof),valor(nodo)]
                    FSI
                    jugada=sacarprimera(E(prof));
                    nodo=restaurar(nodo,jugada)
          FSI
        FMQ
        SI prof=1
          V: minimax=ev(1)
           EXITO
        FSI
        lado=-lado;
        prof=prof-1;
        jugada=sacarprimera(E(prof));
        nodo=restaurar(nodo,jugada);
        SI lado=1
          V: ev(prof)=MAX[ev(prof),ev(prof+1)]
          F: ev(prof)=MIN[ev(prof),ev(prof+1)]
        FSI
FMQ
FAlg()
```

Una versión simplificada del proceso de Minimax puede ser estipulada por:

El objetivo del jugador MAX es maximizar el valor de la función *f* de evaluación que mide la proximidad estimada a la meta de una situación dada y, así mismo, los nodos MIN reflejan el modo de actuar del adversario con respecto al objetivo del nodo MÁXIMO; es decir, realizar aquel movimiento que haga más pequeño el valor de *f*.

El estado de un nodo MAX será:

- Ganador si alguno de sus sucesores conduce a un terminal ganador (carácter disyuntivo).
- Perdedor si ningún sucesor conduce a uno ganador (carácter conjuntivo).
- Empate si ningún sucesor conduce a la meta y al menos uno conduce al empate.

El estado de un nodo MIN será:

- Ganador si todos sus sucesores llegan a un final ganador.

- Perdedor si alguno conduce a un final perdedor.
- Empate si ningún sucesor es perdedor y alguno conduce a un empate.

Por tanto una estrategia ganadora para MAX sería un subárbol en el que todos sus nodos terminales son ganadores y una estrategia ganadora para MIN será el subárbol en el que todos sus nodos terminales son perdedores.

El método de MINIMAX se comporta como un procedimiento de *búsqueda con retroceso* con una frontera de exploración calculada.

```
Minimax(m, profundidad, jugador)
SI m no tiene sucesores o ha alcanzado el límite de profundidad
        V: devolver mmv(m) = f(m)
 FSI
 Generar sucesores de m
   Inicializar la variable mejor con el valor mínimo
   mejor=minj {f(j) para toda j}
   PC sucesor w de m
        mmv(w) = Minimax(w, profundidad+1, C(jugador)
        (C(jugador) una función que cambia de jugador)
        mejor= max[-MMv(w), mejor]
   FPC
 Una vez se analizan recursivamente todos los sucesores de un nivel,
 devolver nodo con mejor valor
 devolver camino
 mmv=mejor
FMinimax()
```

Un árbol de coste uniforme de una profundidad p y con factor de ramificación n contiene n^p nodos terminales que serán analizados por el procedimiento MINIMAX. Pero hay que encontrar al menos dos estrategias, una para MAX y otra para MIN, que sean compatibles. Dado que una estrategia se expande en niveles alternativos del árbol su número de nodos es alrededor de $n^{p/2}$.

Ventajas: Es un método general y sencillo que permite representar y buscar estrategias ganadoras en los problemas de juegos en los que hay dos adversarios. El etiquetado de los nodos permite identificar las distintas estrategias posibles.

Desventajas: Peca por ser demasiado exhaustivo (poco eficiente debido a su complejidad).

{PRIVADO }3.4.3 Poda ALFA-BETA{TC \l 33 ".4.3 Poda ALFA-BETA"}

Es una modificación del método de Minimax que permite ahorrarse el recorrido de

caminos inútiles del árbol de búsqueda. Dada su efectividad es el método más común en los problemas de juegos y no hay riesgos con la poda.

Mientras se esta explorando el espacio de estados y propagando las evaluaciones del último nivel, las posiciones no terminales reciben valores provisionales que permiten descartar ramas enteras del árbol de búsqueda. En efecto, si el valor provisional de un nivel Min llega a ser menor que el valor provisional del nivel Max inmediatamente superior, el valor final del nivel Min ya no podrá alterar el valor provisional del nivel Max. Esta estructura abandona el estudio de subárboles que no puedan conducir a soluciones mejores que las ya encontradas en un cierto nodo.

Es decir, el algoritmo facilita no explorar aquellas ramas de un árbol que el análisis indique no es de interés para el jugador que va a realizar una jugada. El ahorro en exploración del árbol que puede lograrse es extremadamente grande como consecuencia de aplicar el algoritmo de poda alfa-beta[18].

Si a > b, entonces para cualquier c y d se cumple max{a,min{b,c},d} = max{a,d}.
No es necesario conocer c para evaluar max{a,min{b,c},d}.

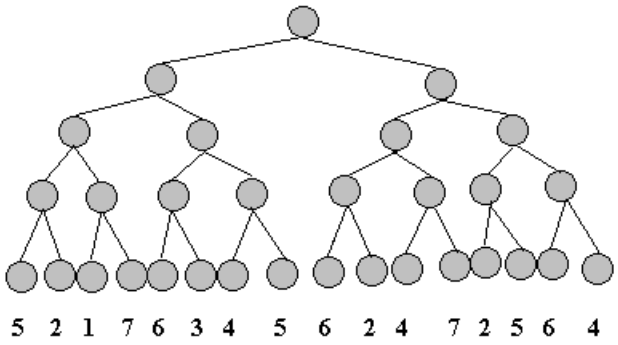


Figura 3.13. *Árbol de búsqueda para un juego.*

Generalizando se llega a las siguientes reglas de corte:

1. La búsqueda puede ser abandonada debajo de cualquier nodo MIN con un valor provisorio, llamado cota beta, inferior o igual al valor provisorio de cualquiera de sus ancestros MAX.
2. La búsqueda puede ser abandonada debajo de cualquier nodo MAX con un valor provisorio, llamado cota alfa, superior o igual al valor provisorio de cualquiera de sus ancestros MIN.

Un procedimiento de búsqueda en profundidad consiste en comenzar en la posición actual y generar un conjunto de posiciones posibles, buscando modificar la estrategia

[18] *De John McCarthy (1961), dado como respuesta a la explosión combinatoria por búsqueda en árboles.*

de ramificación y acotación.

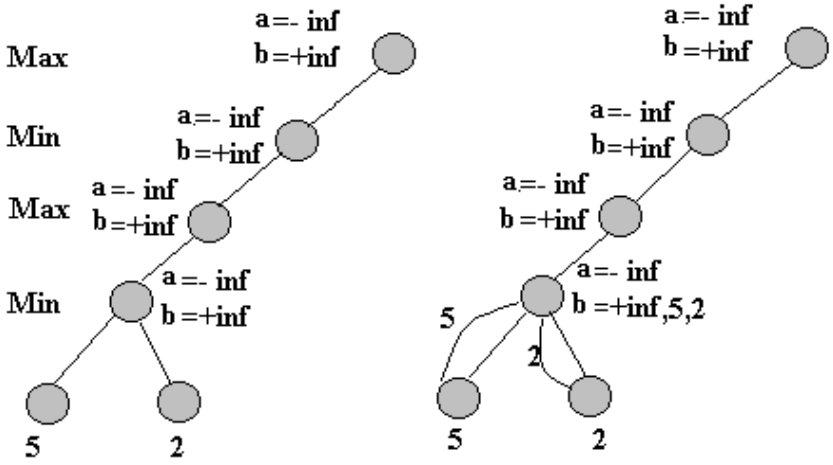


Figura 3.14. *Proceso de poda alfa-beta en la primer parte del árbol.*

```
Alg Alfa_beta()
SI nivel = 0 V: a = - inf
          b = + inf
FSI
HQ todo hijo visitado
          SI nivel es minimización
            V: HQ todo hijo visitado o a >= b (poda)
              uno ()
              SI valor < b V: b <- valor
               FSI
             FHQ
             reportar b
            F: HQ todo hijo visitado o a >= b (poda)
              uno ()
              SI: valor > a V: a <- valor   FSI
             FHQ
             reportar a
          FSI
FHQ
SALIR
uno()
SI nivel alcanzado <> último
          V: a = a-act
            b = b-act
          F: reportar valor
FSI
retornar
```

Problemas
Fin Alfa_beta()

La eficiencia del algoritmo alfa-beta depende del número de cortes que se realizan. La figura 3.14 muestra parte del proceso alfa-beta para el árbol de la figura 3.13. En la parte B indica una primera poda, y como puede verse no se analiza un nodo (valor 5) debido a la misma poda. Continuando el proceso, encontramos las figuras 3.15 y 3.16, en donde se ve el recorrido total realizado.

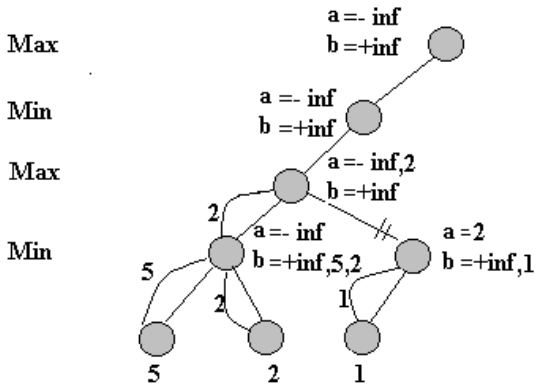


Figura 3.15. *Proceso de Poda Alfa-Beta.*

Una vez realizado el proceso, aplicarle la poda establecida, el árbol queda reducido como lo indica la figura 3.17.

Tabla 3.4 *Características de los nodos Max y Min.*

{PRIVADO }Características nodos MAX	Características nodos MIN
* Tienen valores **a**. * Tienen garantizado conseguir, a partir de ese nodo, un estado con un valor >=**a**. * Los valores **a** de un nodo Max son límites inferiores del nodo y nunca se decrementan. * El valor **a** de un nodo no terminal es el mayor valor entre valores de sus nodos sucesores.	* Tienen valores **b.** * Tienen garantizado conseguir, a partir de ese nodo, un estado con un valor <=**b**. * Los valores **b** de un nodo Min son límites superiores del nodo y nunca se incrementan. * El valor **b** de un nodo no terminal es el menor valor entre valores de sus nodos sucesores.

Algoritmo de Minimax con corte alfa-beta (primera forma)[19]:

// Entrada: nodo,lado
// Salida: Valor_posición
// lado=1 (Max). lado=-1 (Min)
Alg()

[19] *Este algoritmo y el de Minimax fueron mejorados por los profesores Luis Roberto Ojeda Chaparro y Nelson Becerra Correa.*

```
Iniciar: prof=1; E(1)=listajugadas(nodo,lado); ev(1)=-(lado)*inf
MQ prof>=1
  MQ E(prof)<>vacío
    jugada=primera(E(prof));
    nodo=jugar(nodo,jugada);
    SI prof<>(profmaxima-1)
          V: prof=prof+1;
           lado=-lado;
```

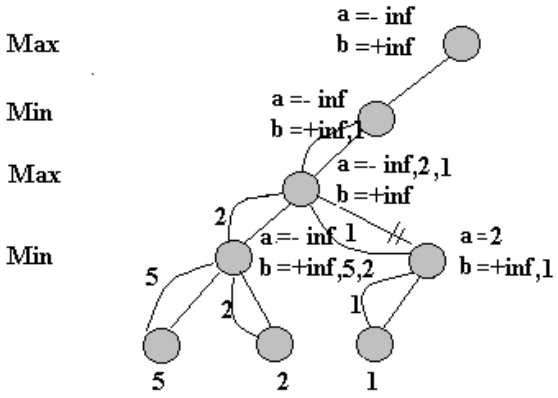


Figura 3.16. *Proceso de poda Alfa-Beta.*

```
          E(prof)=listajugadas(nodo,lado);
          ev(prof)=-(lado)*inf
        F: sw=1
          SI lado=1
                V: ev(prof)=MAX[ev(prof),valor(nodo)]
                  SI prof>1 V: SI ev(prof)>=ev(prof-1)
                            V: eliminar(E(prof));
                              sw=0
                           FSI
                  FSI
                F: ev(prof)=MIN[ev(prof),valor(nodo)]
                  SI prof>1 V: SI ev(prof)>=ev(prof-1)
                            V: eliminar(E(prof));
                              sw=0
                           FSI
                  FSI
                  SI sw=1 V: jugada=sacarprimera(E(prof));
                      nodo=restaurar(nodo,jugada)
                  FSI
          FSI
        FSI
  FMQ
  SI prof=1 V: retornar(ev(1))
```

```
        Problemas
        terminar
  FSI
  lado=-lado;
  prof=prof-1;
  jugada=sacarprimera(E(prof));
  nodo=restaurar(nodo,jugada);
  SI lado=1
        V: ev(prof)=MAX[ev(prof),ev(prof+1)]
          SI prof>1 & ev(prof) >= ev(prof-1)
                V: eliminar(E(prof))
          FSI
        F: ev(prof)=MIN[ev(prof),ev(prof+1)]
          SI prof>1 & ev(prof) <= ev(prof-1)
                V: eliminar(E(prof))
          FSI
  FSI
FMQ
FAlg()
```

{PRIVADO }3.4.4 Escalada simple{TC \l 33 ".4.4 Escalada simple"}

El algoritmo de escalada es una variante del de generación y prueba; en él existe realimentación para ayudar al *generador* a decidirse por cuál dirección debe moverse en el conjunto de estados.

```
        Alg Esc-sim()
          estado actual = estado_inicial
          SI estado_actual = META V: ÉXITO
          FSI
           HQ hallar META o no existan reglas para aplicar
                seleccionar regla a aplicar (no aplicada)
                aplicar regla a estado_actual
                SI estado_nuevo = META V: ÉXITO
                  F: SI estado_nuevo es mejor a estado_actual
                        V: estado_actual = estado_nuevo
                    FSI
                FSI
            FHQ
        Fin Esc_sim()
```

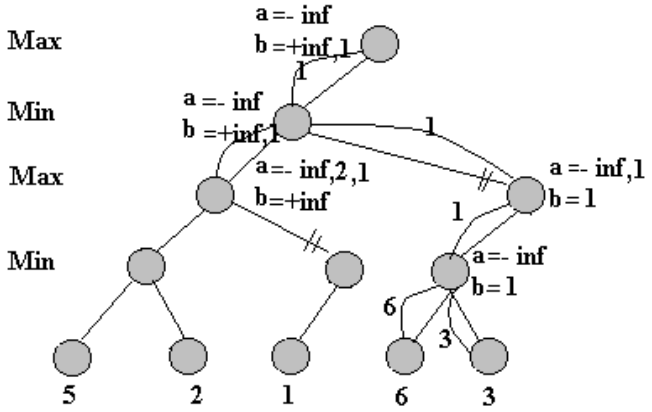


Figura 3.17. *Estado final del proceso Alfa-beta.*

Nótese que la función heurística se aplica en "Es mejor a", un término muy vago, y por tanto debe proporcionarse una definición precisa.

{PRIVADO }3.4.5 Subir montañas{TC \l 33 ".4.5 Subir montañas"} (Hill Climbing)

Es una mejora del de *Escalada Simple*. Una versión simple del algoritmo de subir montañas es:

```
Lista <- RAÍZ
HQ Lista = vacía o se alcance META
  tomar nodo de Lista, X
  SI X = META V: ÉXITO
    F: agregar hijos de X al final de Lista
          Ordenar lista
  FSI
FHQ
```

Desventajas: no es completo, no funciona cuando existen máximos locales, no llega así a la meta.

Considerar todos los posibles movimientos a partir de un estado actual y elige el mejor de ellos, es una variación del algoritmo de escalada simple.

```
Alg Sub-mon()
 estado actual = estado_inicial
  SI estado_actual = META V: ÉXITO
  FSI
   HQ halle META o no existan cambios al estado actual
          Generar sucesores de estado_actual
```

```
Problemas
        Lista = NULL
        HQ se evalúe todo sucesor
          SI sucesor = META V: ÉXITO
            F: SI sucesor es mejor a estado_actual
                  V: guardar sucesor en Lista
              FSI
          FSI
        FHQ
        SI Lista >< NULL V: seleccionar mejor nodo de Lista
                  F: ÉXITO
  FHQ
Fin Sub_mon()
```

Tanto el algoritmo de escalada simple como el de subir montañas pueden no encontrar una solución. Cualquiera de los dos puede acabar sin encontrar un estado objetivo, y en cambio sí hallar un estado del que no puede generarse estados mejores que él.

{PRIVADO }3.4.6 Método del gradiente o búsqueda en escalada{TC \l 33 ".4.6 Método del gradiente o búsqueda en escalada"}

Se sigue el recorrido a través de los nodos en los que el valor de la pendiente sea máximo. El parámetro que define el sentido de la máxima pendiente es el valor de la feh aplicada, que llamaremos g.

```
Denominar m al estado inicial, elegido <- m
HQ hallar META o se devuelva fallo
  Expandir m en SUCC
  PC operador aplicable y cada forma de aplicación
        aplicar el operador a m generando estado nuevo
        SI nuevo = META V: ÉXITO
          F: SI g(nuevo) es mejor de g(elegido) V: elegido <- nuevo
        FSI
  FPC
  SI g(elegido) <> g(m) V: m= elegido
        F: FALLO
  FSI
FHQ
```

Se usa en todos los casos en que se pueda identificar una función que tenga la propiedad de ir creciendo (o decreciendo) hasta el valor que tenga la función en el nodo meta:

- La teoría de juegos
- Los problemas de búsqueda de máximos y mínimos en el campo del análisis numérico
- Las situaciones en las que en cualquier estado es posible aplicar cualquiera de

los operadores disponibles, lo que se denomina *conmutatividad*.

Ventajas: su sencillez y que la única memoria necesaria consiste en guardar el estado alcanzado (elegido).
Desventajas: la dependencia con respecto de la definición correcta de la función g (que sea lo más informativa posible).

3.4.7. Técnica de escalada

La técnica de escalada es la evolución de la técnica de profundidad en la que cada nodo se dispone en una forma de evaluar cómo está de cerca o de lejos la solución. La forma más común de evaluar es la función de evaluación.

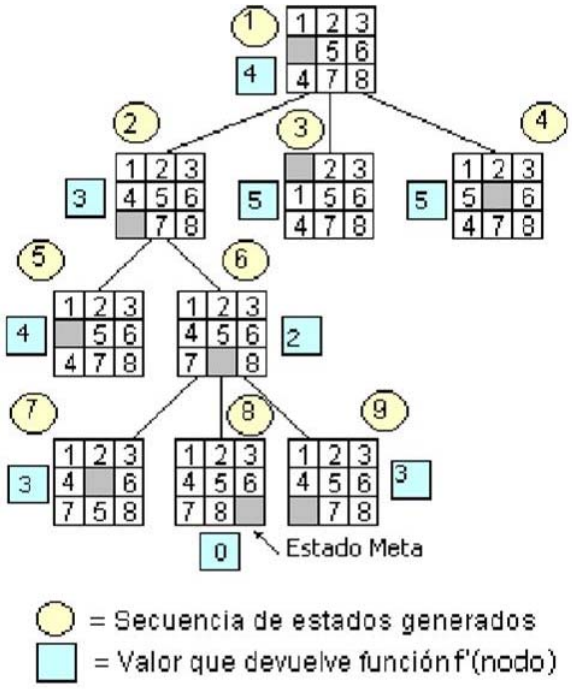


Figura 3.18. *Grafo de aplicación de funciones de evaluación.*

f(nodo)= # de casillas bien colocadas (maximizo)
f'(nodo)= # de casillas mal colocadas (minimizo)

Como ejemplo del juego 8-puzzle se puede definir una función de evaluación que fuera igual: f(nodo)= # de casillas bien colocadas (máximo)

Es decir, devuelve un número que representa como está de cerca un determinado estado de la solución, cuanto mayor sea el número se estará cerca de la solución.

Por tanto al tener que elegir entre varios estados se debería escoger aquel, que tendría un valor mayor de esta función; es decir, es una función que se debe maximizar.

Procedimiento **escalada** (*estado inicial, estado final*)
    X = estado inicial; Exito = Falso.
    HQ Éxito.
        Generar los sucesores de X
        SI algún sucesor es estado final **V**: Éxito = Verdadero.
            F: Evaluar cada nodo con la función de evaluación
            X = mejor sucesor
        FSI
    SI Éxito V: Solución = camino desde nodo del estado-inicial al nodo X.
        F: Solución = Fracaso.
    FHQ
Fin_Procedimiento()

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

1. Puede haber máximo o mínimo local. 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.
2. *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 pueden adoptar son:
    - Retroceder
    - Dar mas de un paso

    a. *Retroceder:* A la lista de razones a la que se debe retroceder que provienen a la técnica de profundidad se le añade cuando ocurra cualquier caso de los anteriores.
    b. *Dar más de un paso:* En lugar de retroceder se generan todos los sucesores de los sucesores del nodo en cuestión y aún así no hay ningún sucesor de los sucesores que sea mejor que el nodo, se puede expandir un nivel mas, hasta que se obtenga un valor mayor / menor que el nodo.

{PRIVADO }3.4.8. Satisfacción de restricciones{TC \l 33 ".4.7 Satisfacción de restricciones"}

En problemas que satisfacen restricciones (PSR) los estados se definen mediante valores de un conjunto de variables y la comprobación de meta. Esto especifica un conjunto de restricciones que los estados deben satisfacer. La solución de PSR da valores a todas las variables satisfaciendo las restricciones establecidas.

Las restricciones son de diversos tipos. Las unarias se refieren a los valores de una sola variable. Las binarias se refieren a pares de variables. Las terciarias a una tripleta de variables, etcétera.

Cada variable, d_i, de un PSR tiene un dominio D_i, que es el conjunto de posibles valores que puede adoptar la variable. El dominio puede ser discreto o continuo[20].

El estado inicial contiene las restricciones que da la descripción del problema. Un estado meta es aquel que satisface las restricciones "suficientemente" ("suficientemente" debe definirse para cada problema).

El diseño de tareas puede contemplarse como PSR, dado que el diseño debe realizarse contemplando unos límites fijos de tiempo, costo y materiales. Un procedimiento de búsqueda para resolver un problema criptoaritmética podría trabajar en un espacio de soluciones en el que a las letras se les asignan números, que serían sus valores. Entonces, un esquema de control primero en profundidad podría seguir un camino de asignaciones hasta descubrir la solución o una inconsistencia.

```
Alg PSR()
  HQ halle solución o inconsistencia
    Crear conjunto ABIERTO con nodos propagando restricciones
    HQ exista inconsistencia o ABIERTO = vacío
            Seleccionar nodo X de ABIERTO
            fortalecer conjunto S de restricciones que afectan a X
            SI S <> al asignado a X la última vez o es primera vez que se examina
              añadir a ABIERTO nodos que comparten restricción con X
              Borrar X de ABIERTO
            FSI
    FHQ
    SI conjunto restricciones definen una solución
          V: presentar solución, SALIR
          F: SI conjunto restricciones definen una inconsistencia
              V: presentar ERROR, SALIR
              F: hacer una suposición (restricción).
                 Repetir hasta hallar una solución o una inconsistencia
            FSI
    FSI
  FHQ
Fin PSR()
```

{PRIVADO }3.4.9. Museo Británico{TC \l 33 ".4.8 Museo británico"}

[20] *Todos los problemas tratados en este texto son de dominio discreto.*

Este algoritmo busca todos los caminos posibles que llegan al nodo META y selecciona el mejor de ellos de una forma sistemática; es simplemente una búsqueda exhaustiva por el espacio del problema.[21]

```
Alg Museobri();
 Lista <- RAÍZ
 #cam = 0, Costo_opt = 0, Costo_act = 0, Camopt = ?
 MQ Lista <> null
  SI hay nodos_adelante
        V: Costo_act + Costo
         Lista <-- nodo
        F: SI nodo = META
           V: SI #cam = 0 V: Camopt <- Lista
                     Costo_opt = Costo_act
                  F: SI Costo_opt > Costo_act
                     V: Camopt <- Lista
                        Costo_opt = Costo_act
                   FSI
             FSI
             #cam + 1
         FSI
         Lista --> nodo
         Costo_act - Costo
         marcar nodo
  FSI
 FMQ
 SI #cam = 0 V: "no existen caminos"
     F: "camino óptimo", Camopt
       "costo óptimo", costopt
 FSI
Fin Museobri()
```

El algoritmo de museo británico emplea la búsqueda en profundidad con retroceso. La figura 3.19 nos muestra el árbol de búsqueda formado por los diferentes caminos a recorrer. La meta está bien determinada debido a que se marca ésta m. La tabla 3.5 muestra el proceso con el algoritmo del Museo Británico[22].

Existen diferentes formas del algoritmo dada las diferentes tipos de búsqueda que se desarrollan y el método para solucionar el problema que se emplee.

Después de estudiar la tabla 3.5, nos podemos dar cuenta que el algoritmo podría

[21] *Este método no presenta problema si el tamaño del árbol es pequeño.*
[22] *Debe recorrerse todo el árbol, aunque se halle el camino de menor costo de primero (no se sabe si verdaderamente es el de menor costo al hallar el primero)*

mejorarse implementando el almacenamiento de todos los caminos encontrados, sus costos y el nivel en la cual se hallan las metas.

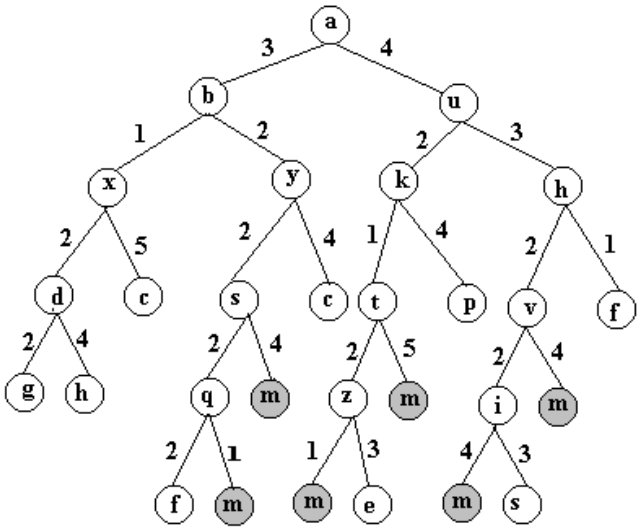


Figura 3.19. *Árbol que indica costos y metas.*

Tabla 3.5 *Proceso del Museo Británico.*

{PRIVADO } Cola	Costo actual	Costo Óptimo	Camino óptimo	#cam
a	0	0	null	0
ab	3			
abx	4			
abxd	6			
abxdg	8			
abxd	6			
abxdh	10			
abxd	6			
abx	4			
abxc	9			
abx	4			
ab	3			
aby	5			
abys	7			
abysq	9			
abysqf	11			
abysq	9			
abysqm	10	10	abysqm	1
abysq	9			
abys	7			

{PRIVADO } Cola	Costo actual	Costo Óptimo	Camino óptimo	#cam
abysm	11	10	abysqm	2
abys	7			
aby	5			
abyc	9			
aby	5			
ab	3			
a	0			
au	4			
auk	6			
aukt	7			
auktz	9			
auktm	10	10	abysqm	3
auktz	9			
aukt	7			
auktm	12	10	abysqm	4
aukt	7			
auk	6			
aukp	10			
auk	6			
au	4			
auh	7			
auhv	9			
auhvi	11			
auhvim	15	10	abysqm	5
auhvi	11			
auhvis	14			
auhvi	11			
auhv	9			
auhvm	13	10	abysqm	6
auhc	9			
auh	7			
auhf	8			
auh	7			
au	4			
a	0			

La solución es camopt = abysqm, costopt = 10, #cam = 6.

Un criterio importante para definir una buena función de evaluación heurística es que las soluciones encontradas con su ayuda esten cercanas al óptimo. Es decir, se llega rápidamente al estado objetivo con una evaluación local que elija el mejor sucesor, pero esto es insuficiente si se quiere además minimizar el coste del camino.

Suponiendo que para cada estado o nodo n existe una evaluación heurística h´(n) que permite estimar la distancia que falta para llegar de n al nodo meta, para encontrar lo más rápidamente posible una solución, conviene priorizar los estados con la menor evaluación h´(n). Pero si lo que está buscándose es el camino óptimo, la evaluación de los nodos debe considerar el coste total del camino y no sólo lo que queda por recorrer. La evaluación g(n) corresponde al largo del camino por el cual se llegó al nodo n; es un estimativo porque siempre queda la posibilidad de encontrar un camino más corto.

La evaluación f(n) expresa, por tanto, un estimativo del largo total del camino desde el nodo inicial a un nodo objetivo pasando por el nodo n. El algoritmo de búsqueda heurística que utiliza una evaluación de este tipo se denomina algoritmo A. Cuando dos estados tienen la misma evaluación heurística, se le da prioridad al primero en generarse ya que tiene más probabilidades de corresponder a un camino de coste mínimo.

Admisibilidad. Un algoritmo de búsqueda es admisible si garantiza encontrar el camino óptimo entre el estado inicial y el estado objetivo. Una búsqueda basada en una función de evaluación heurística g(n)= f(n)+h´(n) es admisible si h´(n) subestima el largo del camino mínimo efectivo.

Para demostrar esta propiedad, se define una función de evaluación $g^*(n)$ que entrega el largo del camino óptimo del nodo inicial a un nodo objetivo, pasando por el nodo n (una función de esta naturaleza se denomina un oráculo, porque implica adivinar cuál es el mejor camino sin recorrer el espacio de estados). Esta función se calcula como una suma $g^*(n)=f^*(n)+h^*(n)$ en que $f^*(n)$ es el largo efectivo del camino óptimo del nodo inicial al nodo n, y $h^*(n)$ el largo efectivo desde n hasta el objetivo.

Siempre se cumple que $f(n)>=f^*(n)$. Se quiere demostrar que si $h´(n)<=h^*(n)$, la búsqueda heurística es admisible.

Demostración por el absurdo:
Supongamos que el nodo objetivo **o** fue generado por un camino de largo l. Se sabe que g(o)=l.

El nodo **o** fue el último en ser expandido; por tanto, para todo nodo **x** de la lista *ABIERTO* se debería cumplir que **g(x)>=g(o)**. Así que g(x)>=l.

Sin embargo, si el camino que lleva a **o** no es el óptimo, debería existir un nodo n en la lista *ABIERTO*, generado antes de expandir **o**, que se encuentra en el camino óptimo.

Si n está en el camino óptimo $f(n)=f^*(n)$.
Por tanto, $g(n)=f^*(n) + h´(n)$.
Por hipótesis $h´(n)<=h^*(n)$, entonces $g(n)<=g^*(n)$.

Si suponemos que el camino de largo l que lleva a o no es óptimo, entonces $g^*(n)<l$. Por tanto, $g(n)< l$.

El algoritmo A con una evaluación heurística $h´(n)<=h^*(n)$ se denomina A^* [23].

"Es una mente muy pobre aquella que sólo puede pensar en una forma de escribir una palabra"
Andrew Jackson.

{PRIVADO }3.4.10. Primero el mejor{TC \l 33 ".4.9 Primero el mejor"}

La figura 3.19 muestra un árbol que describe la situación de los nodos. Se necesita una función que haga un estimativo a los nodos que van generándose, sea esta función g' (indica que es una aproximación de g). Para muchas aplicaciones, es adecuado definir esta función como la suma de dos funciones f y h'. La función f es la medida del costo para ir del nodo estado inicial al nodo actual. La función h' es una estimación del costo adicional necesario para alcanzar un nodo objetivo a partir del nodo actual.

Un algoritmo simple nos dice:

```
Lista = RAÍZ
HQ lista = vacía o se alcance la meta:
  tomar primer nodo de Lista, X
  SI X = Meta V: ÉXITO
    F: eliminar X.
          agregar sucesores de X al final de Lista
          Ordenar Lista, pero no hace eliminaciones (heurística)
  FSI
FHQ
```

Ventaja: es completo, encuentra siempre la solución.

Sin embargo, para implementar un procedimiento de búsqueda *Primero el mejor* con mayor eficiencia se requieren dos listas de nodos:

ABIERTO: nodos que se han generado, a los que se les ha aplicado la función heurística[24], pero aun no han sido examinados (es decir, no se han generado sus sucesores)

CERRADO: nodos que ya se han examinado.

---

[23] *Nótese que un caso particular del algoritmo A^* se da cuando h(n)=0. Este caso corresponde a una búsqueda en amplitud (que también es admisible).*

[24] *Una función heurística también se denomina con frecuencia función objetivo (**objective function**).*

El procedimiento puede resumirse como sigue:

```
Alg Primeromejor()
  ABIERTO <- [RAÍZ]
  CERRADO = []
  Mejornodo = RAÍZ
  HQ Mejornodo = META o ABIERTO = []
        tomar Mejornodo de ABIERTO
        generar sucesores(Mejornodo)
        PC sucesor
          evaluar función costo
          añadirlos a ABIERTO
          llevar padre a CERRADO
        FPC
  FHQ
  SI Mejornodo = META V: indicar camino, costo
      F: indicar no hay soluciones
  FSI
Fin Primeromejor()
```

Usualmente, el costo de ir de un nodo a su sucesor, f, se fija en una constante 1. Algunos problemas explicitan f con un valor. Si h´ vale cero, la búsqueda será controlada por f.

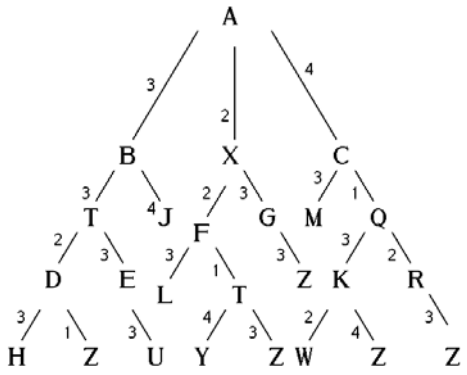


Figura 3.20. *Árbol para hallar soluciones.*

La meta se denomina Z. Inicialmente sólo existe un nodo (la raíz), de forma que éste se expande. Al hacerlo, se generan tres nuevos nodos (la función heurística es una estimación de costos).

Como el nodo X es el más prometedor de todos, es el siguiente que se expande, lo cual produce nuevos nodos, F y G. La función heurística se aplica a los nodos. En este

punto, el camino que nace del nodo B parece más prometedor, por lo que se selecciona para generar nuevos nodos, T y J. Se aplica la función heurística, se observa que el mejor nodo es el F, el cual se expande y aparecen los nodos L y T; en el siguiente paso el nodo que se expande es D, dado que es el mejor de todos. El proceso continúa así hasta hallar una solución al problema.

La figura 3.21, muestra el proceso por búsqueda *Primero el mejor* para hallar una solución.

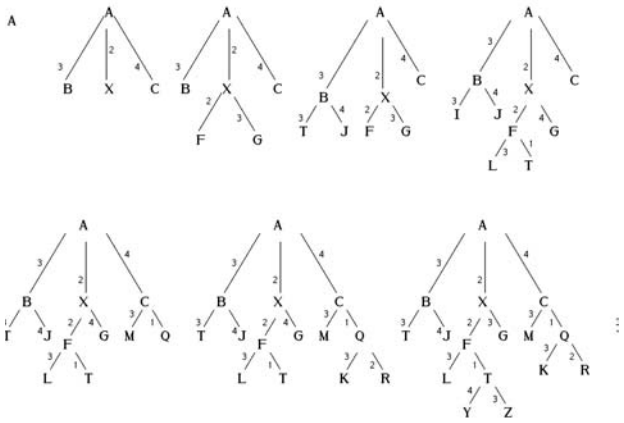


Figura 3.21. *Proceso de búsqueda Primero el Mejor.*

{PRIVADO }3.4.11. A*{TC \l 33 ".4.10 A*"}

El algoritmo de búsqueda *primero el mejor*, es una simplificación del algoritmo A*; el proceso para éste puede ser:

```
Alg A*()
  ABIERTO <- [RAÍZ], g(RAÍZ)=0, CERRADO=[]
  HQ nodo = META o ABIERTO = []
    Tomar mejornodo de ABIERTO que tenga el menor valor de g, llamarlo MN
    eliminar este nodo de ABIERTO, llevarlo a CERRADO
    SI MN = META
          V: recorrer puntero a los antepasados de MN, devolver(camino), ÉXITO
          F: generar sucesores en SUCC
            PC w de SUCC
                  generar enlace de w y MN
                  calcular g(w)=g(MN)+costo(MN, w)
                  SI w esta en ABIERTO
                    V: marcarlo como V
                      SI g(w) < g(V) V: redireccionar el puntero de V a MN
                         cambiar camino de menor costo de w a RAÍZ;
```

```
                    g(V)=g(w), f(V)=g(w)+h(V)
                  FSI
                 F: SI W está en CERRADO
                        V: actualizar enlaces
                  FSI
                FSI
                SI w no está en ABIERTO o CERRADO,
                        V: colocarlo en ABIERTO
                         calcular g(w)=f(w)+h(w)
                FSI
          FPC
    FSI
  FHQ
Fin A*()
```

A* no dice qué nodo elegir en el caso de varios nodos con el mismo valor de g´; por tanto, puede mejorarse al elegir el nodo más (o menos) profundo. Aun así puede haber empates; podríamos elegir entonces el nodo generado más (o menos) recientemente.

La crítica a las búsquedas heurísticas son amplias porque no son precisas. Sin embargo, A* posee una serie de propiedades:

a. Es una heurística completa: incluso para grafos infinitos encuentra la solución si esta existe.
b. Es admisible; la solución que encuentra es óptima.
c. Si la feh es monótona y consistente se halla el camino óptimo para todos los nodos expandidos.
d. A* expande el menor número de nodos.

Una posible mejora al problema de espacio requerido en que cada iteración es una búsqueda en profundidad completa en la que se calcule si cualquier nodo supera un umbral de máximo costo (costo(nodo)>μ, para todo nodo), en cuyo caso se elimina dicha rama y se vuelve al nodo generado más reciente. Este método se llama A* en profundidad progresiva (A*-P).

{PRIVADO }3.4.12. AO*{TC \l 33 ".4.11 AO*"}

Este proceso es generalización del algoritmo A* para el caso de grafos Y/O. Encuentra una solución óptima siempre y cuando la función heurística *h* de estimación de la distancia al objetivo siga un criterio optimista de valoración (devuelva un valor menor o igual que la distancia real existente). Aquí, la solución ya no es un único nodo meta; puede ser una colección de éstos, que se corresponden con cada uno de los subproblemas en los que puede haberse dividido el problema inicial.[25]

[25] *La clave del algoritmo AO* está en la búsqueda (hacia delante) de formas de expandir el mejor grafo*

Problemas

Crear un grafo G con nodo raíz X.
Estimar la distancia heurística a la meta h(X).
HQ X se considere resuelto o su costo supere cierto valor costo-máximo
- Seguir enlaces que señalan el mejor grafo parcial de la solución obtenido hasta el momento hasta alcanzar los extremos de dicho grafo.
- Escoger alguno de dichos extremos, Y.
- Generar sucesores de Y, S.
SI no hay sucesores V: asignarle el valor costo-máximo.
FSI
PC S
SI S= nodo terminal V: marcarlo como resuelto asignarle h(S) = 0;
F: asignarle h(S).
FSI
- Crear un conjunto SUCC solo con los nodos S.
- HQ SUCC = vacio:
Sacar de SUCC algún nodo W que no tenga descendientes en SUCC.
Actualizar costo de W:
Calcular el costo de todos los enlaces que parten de W
asignar h(W), el menor de éstos, marcar dicho enlace
(borrarlo si existe marca de otro enlace previamente elegido de W).
SI todos los nodos del enlace marcado son terminales
V: marcar también W como resuelto.
FSI
SI paso anterior ha hecho que W sea resuelto o si su h(W) ha cambiado
V: añadir todos sus antecesores a SUCC
continuar proceso recursivo para actualizar valores de los nodos (que pueden llegar hasta la raíz del grafo G).
FSI
FHQ
FPC
FHQ

{PRIVADO }3.4.13. Bifurcación y acotamiento{TC \l 33 ".4.12 Bifurcación y acotamiento"}

Como su nombre lo dice, bifurcar y acotar es un algoritmo muy simple.

Lista = RAÍZ
HQ Lista = vacío o se alcance META:
tomar primer nodo de Lista, X
SI X = Meta V: SALIR
F: eliminar X, agrega sucesor(X) sumando el costo acumulado del padre
Ordenar elementos de acuerdo con el costo (heurística)

parcial a la solución encontrada hasta el momento

Expandir todos los nodos con costo menor que nodo meta.

FSI

FHQ

Ventaja: la primer trayectoria encontrada es la solución óptima. Una mejora a *branch and bound* es utilizar programación dinámica. Esta técnica considera que puede llegarse a un mismo nodo por diferentes rutas, su heurística es ordenar los elementos, seleccionar el de menos costo y eliminar los elementos repetidos (con mayor costo).

3.4.14. Recocido Simulado (Simulated Annealing)

Es una variación del ascenso a colina. Al inicio, este algoritmo, permite explorar una buena parte del espacio de estado, de tal forma que la solución final puede resultar insensible al estado inicial. En consecuencia, la probabilidad de quedar atrapado en un máximo local, en una meseta o en un risco, se hace mínima.

El recocido simulado es un proceso computacional que refleja los pasos establecidos en el proceso físico de tratamiento térmico de materiales. En el recocido, por ejemplo, un metal es llevado a elevados niveles energéticos, hasta que alcanza su punto de fusión. Luego, gradualmente es enfriado hasta alcanzar un estado sólido, de mínima energía, previamente definido. Por su naturaleza, este algoritmo debe ser formulado como un **descenso a valle** en el que la función objetivo es el nivel energético.

Las sustancias físicas usualmente se mueven desde configuraciones de alta energía a las de menor energía, así que el descenso al valle, ocurre en forma natural. Pero, eventualmente pueden haber transiciones a niveles energéticos más altos, con una probabilidad dada por:

$$p = e^{-\frac{\Delta E}{k.T}}$$

$$\Delta E = E_{nuevo} - E_{actual}$$

T es la Temperatura absoluta y **k** es la constante de Boltzmann.

El procedimiento que se va a seguir para enfriar el sistema, se llama *programa de recocido*. Su forma óptima depende de cada tipo de problema y usualmente se lo descubre empíricamente. El *programa de recocido*, debe incluir los siguientes ingredientes:

1. El valor inicial de la temperatura.
2. El criterio que será utilizado para decidir cuando reducir la temperatura.
3. La cantidad en que será reducida la temperatura.
4. Cuando finalizar el proceso.

El algoritmo para el recocido simulado, es ligeramente diferente del procedimiento

simple de ascenso a colina. Las diferencias son:

- Se debe respetar el programa de recocido.
- Movimientos a estados peores que el actual, pueden ser aceptados.

Se debe mantener, a más del estado actual, el mejor estado encontrado hasta el momento. Así, si por alguna razón el estado final resulta peor que el mejor encontrado anteriormente, siempre será posible regresar a él.

{PRIVADO }3.5 EJERCICIOS{TC \l 23 ".5 EJERCICIOS"}

1. Tres misioneros y tres caníbales llegan a un río. Hay un bote para cruzar el río que puede ser usado únicamente por máximo dos personas. ¿Cómo debe cruzarse el río de tal forma que los caníbales no deben pasar en número a los misioneros a ningún momento en ningún lado del río?

2. Describir los estados del siguiente problema:
Se tiene un conjunto de cuadros n*n (n impar, n>=3). Colocar en ellos los primeros n números enteros de tal manera que las sumas horizontal, vertical y diagonal sean iguales (suma = $((n^2 + 1)\ n) / 2$).

3. Describir los anteriores problemas como PSR (si es posible).

4. Indicar los procesos que se siguen en la solución de cada problema.

Se cuenta con tres jarras de agua, cada una con la capacidad que se enuncia. Se trata de medir con ellas la cantidad de agua que se propone en cada caso.

Nro.	Jarra-1	Jarra-2	Jarra-3	¿Agua?
1	18	43	10	5
2	9	42	6	21
3	15	39	3	18
4	18	4	48	22
5	14	36	8	6
6	7	34	8	4
7	40	13	11	4
8	31	17	4	9
9	9	11	17	8

5. Escribir el proceso empleado para solucionar los siguientes problemas y plantearlo en forma declarativa.

a. Existen 12 naranjas que pesan 85 gr., pero 3 naranjas pesan 17 gr. y 8 pesan 53 gr. ¿Cuánto pesa cada naranja?

b. 3 pulgas mas 5 pulgas son 7 pulgas ¿por qué? (no es un problema de pulgas).
c. Sea la serie 1, 3, 2, 9, 3, 17, 4, 29, 5, 49, ... ¿Cuáles son los próximos 7 términos?
d. José es mayor que Miguel en 13 años y de Carlina en 2. Carlina tiene el doble de edad de Vicente. Miguel le lleva 3 años a Pedro y es menor que Tulia en 5. Si en 5 años las edades suman 100, ¿cuál es la edad de cada uno?

El peso en una fama difiere en 2 onzas menos por libra con respecto a lo legal. Si la libra cuesta $5200.oo, ¿cuánto gana el carnicero en una arroba?

6. Asignar a cada letra distinta un digito diferente de tal forma que la operación sea lógica y correcta.

```
      LUIS
     LUISA
+    LUCAS
     LUCIA
      CAMA
    _______
    ORGIAS
```

7. El ingeniero Roberto al final de la semana descubrio que debía pagar:

$11000 a la cocinera y al pintor.
$17000 al pintor y al plomero.
$11000 al plomero y al electricista.
$33000 al electricista y al carpintero.
$53000 al carpintero y al albañil.
$25000 al albañil y a la cocinera.

¿Si tiene $100000 cuánto le sobra o le falta al pagarles?

8. Plantee un problema:
   a. De pulgas.
   b. De estrellas.
   c. Biológico.
   d. Complejo.
   e. Sistémico.

Solamente el agua que bebe el caballo
es la que le llega al estómago.
Si el caballo se para frente al agua
no bebe nada.