METODOS DE SOLUCION DE PROBLEMAS.

1.1 Introducción.

La Ciencia de la Computación trata cada vez de resolver nuevos problemas que sean difíciles de resolver mediante las técnicas computacionales existentes. Estos problemas generalmente no tienen solución algorítmica conocida o esta es tan compleja que no tiene una implementación práctica computacional.
Los problemas de decisión que normalmente se presentan en la vida empresarial pueden caer en esta clase de problemas. Por lo general existe una serie de recursos escasos (obreros, presupuesto, tiempo, etc.), o bien requisitos mínimos que hay que cumplir (producción necesaria, horas de descanso, mantenimientos, etc.) que condicionan la elección de la estrategia más adecuada. Por lo general el objetivo al tomar la decisión es llevar a cabo el plan propuesto de manera óptima (bien incurriendo en mínimos costes o bien buscando el máximo beneficio).

No es de extrañar que la resolución de este tipo de problemas haya atraído la atención de los investigadores, dando lugar a la llamada Investigación Operativa. Un ejemplo de esta es la llamada Programación Lineal, siendo el algoritmo Simplex el punto de referencia de la misma. Sin embargo, como ocurre en muchas ocasiones la cantidad de variables que intervienen en el modelo hacen que la aplicación del Simplex no resulte eficiente dado que el tiempo de cálculo necesario es excesivamente largo, incluso para los grandes supercomputadores. *No es que el simplex no puede llegar al óptimo buscado, sino que su tiempo de respuesta no es operativo.*

La respuesta fue desarrollar nuevas técnicas de solución de problemas, similares a las humanas, una de las más importantes fue la búsqueda.

La búsqueda de la I.A. difiere de la búsqueda convencional sobre estructuras de datos esencialmente en que se busca en un espacio problema, no en una pieza de dato particular. Se busca un camino que conecte la descripción inicial del problema con una descripción del estado deseado para el problema, es decir, el problema resuelto. Este camino representa los pasos de solución del problema.

El proceso de buscar una solución a un problema produce un espacio solución, o sea, la parte del espacio problema que se examina realmente. A diferencia de las estructuras de datos que están predefinidas y ya existen cuando comienza la búsqueda, los espacios problema son generalmente definidos proceduralmente, es decir, el espacio problema es creado a medida que es explorado. Se usan procedimientos para definir los siguientes estados posibles en el espacio a través de los cuales la búsqueda puede continuar desde el estado actual. Solamente los caminos explorados tienen que estar definidos explícitamente.

Hay diferentes alternativas para realizar la búsqueda. Desde un punto de vista podemos apreciar tres alternativas: aleatoria, a ciegas y dirigida. Con el siguiente ejemplo se ilustran estos. Supóngase que está en París sin un mapa y no habla francés ¿Cómo llegar hasta la torre Eiffel?.

Un método podía ser tomar aleatoriamente una calle esperando que más pronto que tarde se llegará a la torre. Esta búsqueda aleatoria puede llevar a encontrar la torre pero puede requerir una cantidad infinita de tiempo por la forma arbitraria en la cual seleccionamos un camino (el mismo puede tomarse múltiple veces).

Otra alternativa es seguir exhaustivamente cada calle de inicio a fin. Cuando se alcanza un final se busca una calle paralela y se sigue esta en dirección opuesta independientemente de si nos acercamos o alejamos del objetivo. Eventualmente esta variante consideraría todas las posiciones de nuestro espacio problema. Este tipo de búsqueda se llama a ciegas, ya que no usa conocimiento de cuan cerca estamos de la solución para tomar un determinado camino.

Alternativamente, podemos usar nuestro conocimiento sobre la torre para mejorar la eficiencia de la búsqueda. Suponiendo que el extremo superior de la torre puede ser visto desde cualquier lugar del espacio problema, podemos tomar la calle que nos parezca nos lleve en esa dirección. Esta es llamada una búsqueda dirigida. La búsqueda dirigida es la base de la I.A.

Entonces:
Dado un sistema de I.A., donde el conocimiento está representado en una de las formas de representación del conocimiento (F.R.C.), se desea resolver un determinado problema de un dominio D bajo un conjunto de ciertas condiciones C.
Solucionar este problema significa encontrar una solución $x \in X$ que satisfaga el conjunto de condiciones C, donde X es el conjunto de todas las soluciones posibles a dicho problema.
Este proceso de solucionar un problema se realiza mediante una búsqueda en el espacio X. Existen tres esquemas básicos de solución de problemas:

- Esquema de producción.
- Esquema de reducción.
- Esquema de reducción débil.

Analicemos cada uno de estos esquemas.

Esquema de Producción: En este esquema el problema a solucionar se representa en un espacio de estados y su solución se reduce a la búsqueda en este espacio. Los conceptos principales en este esquema son:

- Estados: situaciones en que se pueden encontrar los objetos que caracterizan al problema durante su solución.
- Movimientos: acciones legales que permiten pasar de un estado del problema a otro.

- Función de evaluación de estados: asigna a un estado un estimado heurístico del esfuerzo necesario para alcanzar una solución desde este estado.
- Función de selección de movimientos: selecciona uno o más de los movimientos aplicables.
- Función de selección de estados: selecciona el estado a partir del cual debe continuar la búsqueda.

Encontrar la solución de un problema significa, entonces, descubrir algún camino entre el estado inicial y el final. El procedimiento general de un esquema de producción es:

1) Determinar, a partir del problema, el estado inicial S_0 y colocarlo como raíz del árbol de búsqueda.
2) Aplicar la función de selección de movimientos a S_0 y generar sus estados descendientes.
3) Si el árbol de búsqueda contiene alguna solución, o sea, si uno de los estados generados es el estado final, entonces TERMINAR y EXITO.
4) Si no,
   a) Aplicar la función de evaluación a cada uno de los estados genera-dos.
   b) Si no existen en el árbol estados sin procesar, entonces TERMINAR y FALLO.
   c) Aplicar la función de selección de estados a los estados del árbol de búsqueda para seleccionar uno de ellos.
   d) Aplicar la función de selección de movimientos al estado seleccionado y generar los nuevos estados aplicando dichos movimientos.
   e) Ir al paso 3).

La construcción de la función de evaluación de estados para un problema concreto requiere del uso de conocimiento sobre ese problema. No siempre se dispone de dicho conocimiento, por lo que no siempre es posible definir esta función. Esto trae por consecuencia que los métodos de búsqueda se dividan en dos tipos:

- Búsqueda a ciegas: no se dispone de conocimiento para definir la función de evaluación de estado.
- Búsqueda heurística: se puede definir la función de evaluación de estado.

Dentro de estos dos tipos existen diferentes métodos de búsqueda, los cuales están determinados por la forma en que se construyen las funciones de selección de estados y de movimientos. Entre estos métodos podemos señalar:

Métodos de búsqueda a ciegas (Brute-Force searches):

- Algoritmo del museo británico.
- Primero en profundidad (depth-first).
- Primero a lo ancho (breadth-first).
- Búsqueda en árboles y/o.
- Búsqueda en sistemas de producción.
- Búsqueda bidireccional.
- Búsqueda por diferencias.
- Búsqueda de soluciones múltiples.

Métodos de búsqueda heurística:
- Búsqueda por el incremento mayor (hill-climbing).
- Búsqueda por el mejor nodo o algoritmo A^* (best-first).
- Búsqueda heurística en árboles y/o o algoritmo AO^*.

Existen otros métodos de solución de problemas a los que no haremos alusión en este documento.

Esquema de reducción: En este esquema el problema se descompone en varios subproblemas, los cuales se resuelven de manera independiente y luego se combinan sus soluciones para obtener la solución del problema original. Existen dos tipos de movimientos: uno de ellos se encarga de la descomposición de un estado en otros y el otro constituye un movimiento terminal que resuelve el problema completamente. Los conceptos principales de este esquema, así como su procedimiento general son similares a los del esquema de producción, pero las funciones de evaluación de estado, selección de estados y selección de movimientos se basan en consideraciones diferentes. Es usual representar los problemas que se resuelven con este esquema mediante árboles y/o.

Esquema de reducción débil: Este esquema está orientado a tareas de planificación.

1.2 Pasos para resolver un problema.

Para resolver un problema debemos seguir los siguientes pasos:
- Definir el problema con precisión: especificar el espacio del problema, los operadores para moverse en dicho espacio y los estados inicial y final o meta.
- Analizar el problema: determinar las características del problema para seleccionar las técnicas que pueden resolverlo.
- Escoger la mejor técnica y aplicarla al problema particular.

1.2.1 Primer paso: definición precisa del problema.

El primer paso hacia el diseño de un sistema para resolver un problema es la creación de una descripción formal y manejable de dicho problema, o sea, la definición del espacio de búsqueda de las soluciones.

1.2.1.1. Tipos de espacios de búsqueda.

Un *espacio de búsqueda* no es más que el ambiente o espacio de todas las soluciones posibles donde se realiza la búsqueda de una solución. Está formado por un conjunto de nodos que constituyen soluciones parciales o posibles del problema y un conjunto de operadores que permiten movernos de un nodo a otro. El proceso de búsqueda de una solución consiste, entonces, en encontrar

una secuencia de operadores que transformen el nodo inicial en el final. Existen tres tipos de espacios de búsqueda:

- Espacio de estado.
- Espacio de reducción de problemas.
- Arboles de juego.

Analicemos cada uno de estos tipos.

Espacio de estado.

Veamos un ejemplo:

Analicemos el problema de los jarros de agua, el cual consiste en que se tienen dos jarros de agua de 3 y 4 litros respectivamente. Ninguno de los jarros tiene marca de medición y se puede usar una bomba para llenar de agua los mismos. ¿Cómo poner exactamente 2 litros de agua en el jarro de 4 litros?.

El espacio de estados puede describirse como el conjunto de pares de enteros (x,y) tales que x=0,1,2,3,4 y y=0,1,2,3; *x* representa el número de litros de agua del jarro de 4 litros y *y* representa el número de litros de agua del de 3 litros. El estado inicial es (0,0), pues inicialmente los jarros están vacíos y el estado final es (2,n) para cualquier valor de n, pues el problema no especifica cuántos litros deben quedar en el jarro de 3 litros. Los operadores para moverse de un estado a otro pueden describirse de la siguiente manera:

1. $(X,Y / X<4) \rightarrow (4,Y)$ Llenar el jarro de 4 litros.
2. $(X,Y / Y<3) \rightarrow (X,3)$ Llenar el jarro de 3 litros.
3. $(X,Y / X>0) \rightarrow (X-D,Y)$ Arrojar en el suelo D litros de agua del jarro de 4 litros.
4. $(X,Y / Y>0) \rightarrow (X,Y-D)$ Arrojar en el suelo D litros de agua del jarro de 3 litros.
5. $(X,Y / X>0) \rightarrow (0,Y)$ Vaciar el jarro de 4 litros en el suelo.
6. $(X,Y / Y>0) \rightarrow (X,0)$ Vaciar el jarro de 3 litros en el suelo.
7. $(X,Y / X+Y\geq 4 \wedge Y>0) \rightarrow (4,Y-(4-X))$ Echar agua del jarro de 3 litros en el jarro de 4 litros hasta que el jarro de 4 litros esté lleno.
8. $(X,Y / X+Y\geq 3 \wedge X>0) \rightarrow (X-(3-Y),3)$ Echar agua del jarro de 4 litros en el jarro de 3 litros hasta que el jarro de 3 litros esté lleno.
9. $(X,Y / X+Y\leq 4 \wedge Y>0) \rightarrow (X+Y,0)$ Verter todo el contenido del jarro de 3 litros en el jarro de 4 litros.
10. $(X,Y / X+Y\leq 3 \wedge X>0) \rightarrow (0,X+Y)$ Verter todo el contenido del jarro de 4 litros en el jarro de 3 litros.

En la práctica, las reglas 3 y 4 deben omitirse, pues aunque representan acciones que están permitidas en el dominio del problema, no tiene sentido aplicarlas, ya que no nos acercan a la solución del mismo porque D no se puede medir.

Por supuesto, para resolver este problema de los jarros de agua, es necesario implementar un mecanismo que seleccione la regla cuya parte izquierda concuerde con el estado actual, genere los nuevos estados y así sucesivamente, hasta alcanzar el estado final. Estos métodos los estudiaremos en las próximas secciones.

En [Bra86] pueden verse las representaciones gráficas de los espacios de estado para los problemas del mundo de bloques y del rompecabezas de 8 piezas.

Un espacio de estado es, justamente, otro formalismo para representar el conocimiento. Se representa, usualmente, mediante un árbol donde cada nodo corresponde a un estado particular del problema y cada arco corresponde a un operador de transición de estados. La solución del problema puede ser definida, entonces, como una búsqueda de un camino entre el nodo correspondiente al estado inicial y el nodo correspondiente al estado final o meta. La forma en que se represente cada estado varía considerablemente de problema en problema. Por ejemplo, en el ajedrez puede ser una matriz de 8x8 donde cada elemento tiene el carácter que representa la pieza situada en esa posición; en el problema del camino entre las ciudades podría ser una cadena con el nombre de la ciudad y en el problema de los jarros de agua podemos usar dos enteros. Si el problema que queremos resolver es más complicado, podemos usar una de las F.R.C. estudiadas en el capítulo anterior para representar cada estado individual.

Espacio de reducción del problema.

A diferencia del espacio de estado, en este espacio el nodo inicial representa el problema original que se desea resolver, los nodos finales o metas constituyen problemas que pueden resolverse mediante una primitiva simple y los restantes nodos representan subproblemas en que puede descomponerse un determinado problema. Los arcos serían, entonces, operadores que permiten descomponer un problema en un conjunto de subproblemas. Existen dos tipos de nodos. Si sólo basta con resolver uno de los subproblemas en que se descompuso un nodo dado para resolver éste, el nodo se denomina *nodo O* (or). Si por el contrario, deben ser resueltos todos los subproblemas en que se descompuso el nodo, éste se denomina *nodo Y* (and).

Este tipo de espacio de búsqueda se representa, usualmente, mediante un árbol y/o, el cual no es más que un árbol que contiene nodos del tipo Y y nodos del tipo O.

Una solución de un problema sería, entonces, un subárbol que comienza en el nodo raíz y que contiene siempre una de las ramas de los nodos O y todas las ramas de los nodos Y hasta llegar a uno o varios nodos terminales.

En este tipo de espacio es conveniente representar aquellos problemas que pueden ser descompuestos fácilmente en subproblemas, como por ejemplo, la generación de estructuras químicas y la integración simbólica.

En la figura 3.1 se muestra un ejemplo sencillo de un problema representado mediante un árbol y/o. El problema de adquirir un televisor puede descomponerse en dos subproblemas: robarlo o tratar de adquirirlo mediante una vía legal. Esta última puede descomponerse a su vez en dos subproblemas, los cuales deben ser resueltos para poder resolver el problema original.

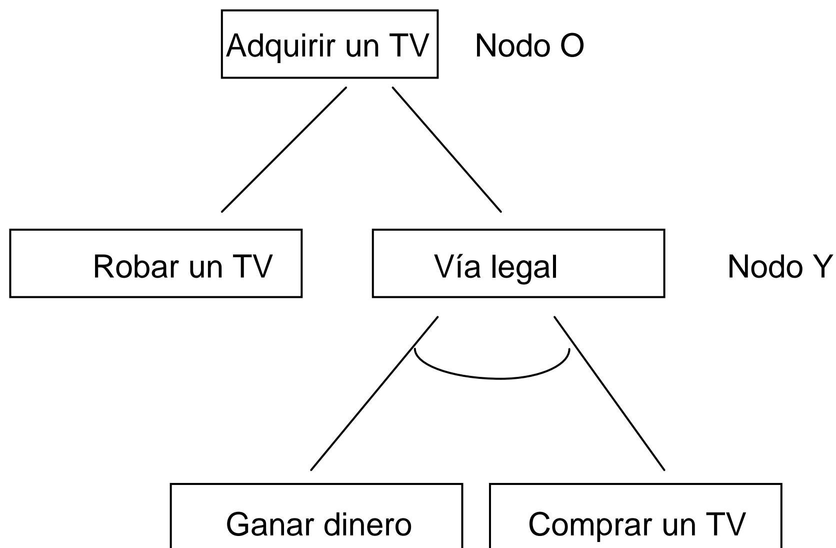


Fig. 1.1. Árbol y/o para el problema de adquirir un televisor.

Analicemos otro ejemplo. Supongamos que tenemos el mapa representado en la figura 1.2, donde se muestra un conjunto de ciudades y los caminos existentes entre ellas. Se desea encontrar un camino desde la ciudad 1 a la 11. Existen dos vías para lograr este camino: una pasando por la ciudad 6 y la otra pasando por la ciudad 7. Pero el subproblema de encontrar un camino por la primera vía significa encontrar un camino entre las ciudades 1 y 6 y un camino entre las ciudades 6 y 11. Lo mismo ocurre para el subproblema de encontrar el camino por la segunda vía. Una porción del árbol y/o correspondiente a esta descomposición del problema original puede verse en la figura 1.2.

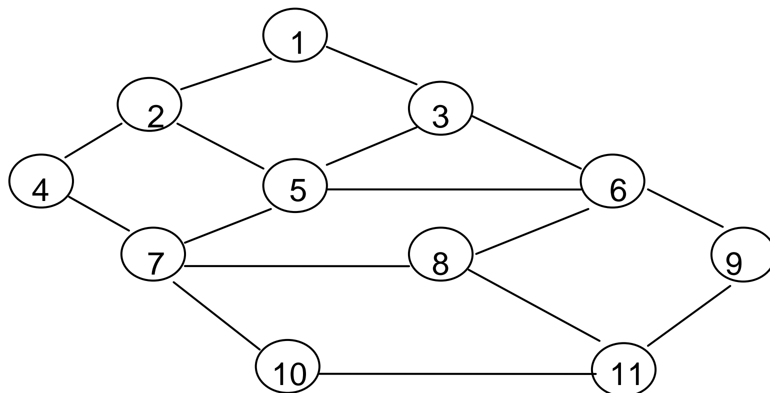
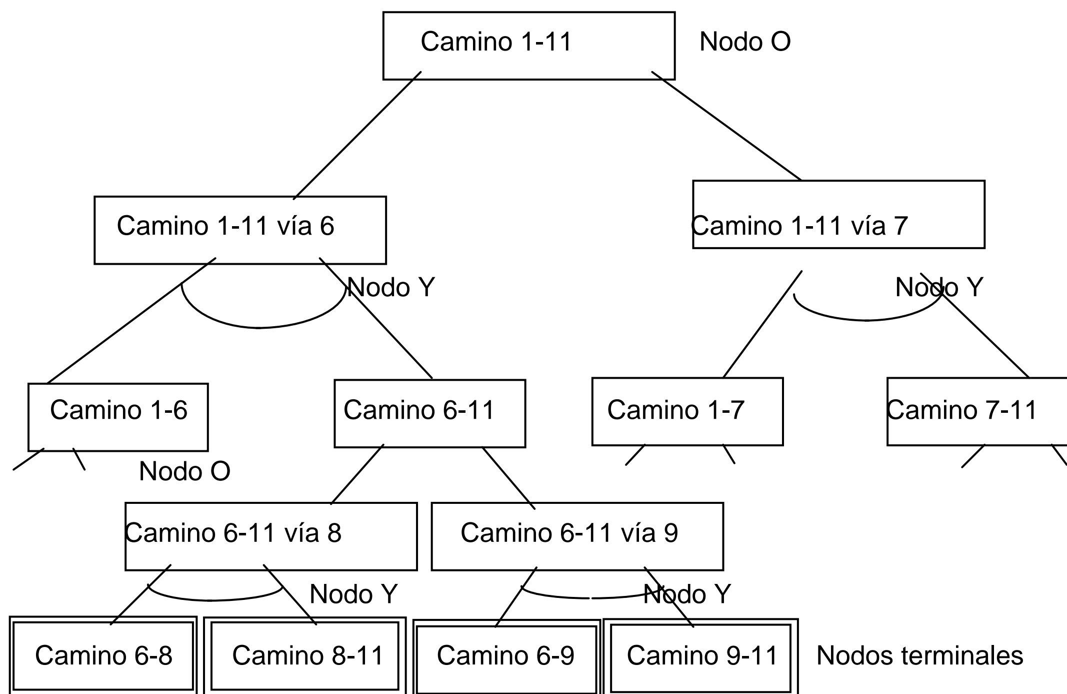


Fig. 1.2. Un mapa compuesto de 11 ciudades.

Por otro lado, ¿cómo podríamos representar el problema de la integración simbólica en un árbol y/o? Esto se realizaría, fácilmente, situando como nodo inicial la integral que se desea resolver. Los operadores de reducción del problema serían, por ejemplo, la regla de la integral de la suma y la integración por partes, las cuales generarían los nodos intermedios del árbol y las reglas que permiten calcular la integral de funciones primitivas, las cuales generarían los nodos terminales. Estos nodos, a su vez, serían funciones que no contienen el símbolo integral.



1.2.2. Segundo paso: análisis del problema.

Para escoger la técnica de I.A. más apropiada para resolver un problema es necesario analizar el mismo teniendo en cuenta los siguientes aspectos:

- ¿Se puede descomponer el problema en un conjunto de subproblemas independientes más pequeños o más fáciles?
- ¿Es posible en la solución del problema ignorar o deshacer pasos mal hechos?
- ¿Es predecible el universo del problema?
- ¿Se desea una solución cualquiera o la mejor solución al problema?
- ¿Es consistente el conocimiento disponible para resolver el problema?
- ¿Cuál es el papel del conocimiento?
- ¿Se requiere la interacción con una persona?

En las próximas secciones explicaremos cada uno de estos aspectos.

1.2.2.1. ¿Se puede descomponer el problema?.

Los problemas que se pueden descomponer en un conjunto de subproblemas se pueden solucionar usando la técnica de *divide y vencerás* (divide and

conquer). Un ejemplo de este tipo de problema es la integración simbólica. Sin embargo, estas técnicas generalmente no pueden usarse en aquellos problemas no descomponibles, aunque a veces es posible usarlas para generar una solución aproximada y, entonces, arreglarla para reparar los errores causados por las interacciones existentes entre los subproblemas. Un ejemplo de problema no descomponible es el mundo de bloques.

1.2.2.2. ¿Es posible ignorar o deshacer pasos para la solución?

En relación con este aspecto existen tres clases de problemas:

1. Ignorables: En ellos los pasos dados incorrectamente para la solución del problema pueden ignorarse. Un ejemplo es la demostración de un teorema matemático. Supongamos que comenzamos demostrando un lema que pensamos es útil, pero después nos damos cuenta que no era de ninguna ayuda. Cualquier regla que pudo haberse aplicado, todavía puede aplicarse. Podemos simplemente ignorar lo hecho y lo único que se ha perdido es el esfuerzo.
2. Recuperables: En ellos los pasos dados incorrectamente para la solución del problema pueden deshacerse. Un ejemplo es el rompecabezas de 8 piezas, el cual consiste en que se tiene una bandeja en la que se colocan ocho baldosas cuadradas. El noveno cuadrante sobrante queda sin cubrir. Cada baldosa tiene un número sobre ella. Una baldosa que esté adyacente al espacio en blanco puede deslizarse a dicho espacio. El juego consiste en, dadas una posición inicial y una posición final (usualmente las baldosas en orden consecutivo con el espacio en blanco en el centro), transformar la posición inicial en la final, desplazando las baldosas. Al intentar resolver este problema, podemos realizar un movimiento tonto. Los errores cometidos pueden enmendarse, retrocediendo para deshacer cada paso incorrecto. Lógicamente hace falta memorizar el orden de los pasos realizados para poder corregirlos. Note que esto no fue necesario en el caso anterior.
3. Irrecuperables: En ellos los pasos dados incorrectamente para la solución del problema no pueden deshacerse. Un ejemplo es el juego del ajedrez. Si se realiza un movimiento estúpido, no se puede ignorar ni retroceder al principio de la partida. Lo único que puede hacerse es tratar de realizar la mejor jugada a partir de la situación actual.

1.2.2.3. ¿Es predecible el universo del problema?

En los problemas donde es posible predecir qué ocurrirá, pueden usarse las técnicas de planificación, las cuales permiten generar una secuencia de operadores que conducen con certeza a una solución. Ejemplo: en el problema del rompecabezas de 8 piezas, cada vez que se hace un movimiento sabemos exactamente qué pasará, por lo que se puede planificar una secuencia completa de movimientos y saber de antemano el resultado. Sin embargo, existen problemas donde esto no es posible. En ellos las técnicas de planificación generan como máximo una secuencia de operadores que conducen a una solución con una buena probabilidad. Por ejemplo, en el juego de la brisca, para

decidir qué carta jugar no podemos planificar la partida completa, pues no sabemos qué cartas tienen los restantes jugadores ni qué jugadas ellos harán.

1.2.2.4. ¿Se desea una solución cualquiera o la mejor solución al problema?

Los problemas en los que se desea encontrar el mejor camino a la solución son más difíciles de resolver que aquellos donde basta encontrar una solución cualquiera, pues mientras los primeros requieren búsqueda exhaustiva, los segundos pueden resolverse eficientemente usando técnicas heurísticas. Un ejemplo de problema donde basta encontrar una solución se expone a continuación. Supongamos que se tiene el siguiente conjunto de hechos:

1. Marcos era un hombre.
2. Marcos era pompeyano.
3. Marcos nació en el año 40 d.C.
4. Todos los hombres son mortales.
5. Todos los pompeyanos murieron en la erupción del volcán en el año 79 d.C.
6. Ningún mortal tiene más de 150 años.
7. Hoy estamos en el año 1995 d.C.

y se desea responder a la pregunta ¿está vivo Marcos?. Representándolos en lógica de predicados y realizando inferencias se llega fácilmente a una respuesta. Note que en este caso existen dos caminos para llegar a la respuesta de que Marcos no está vivo. Uno de ellos se obtiene utilizando los hechos 1,3,4,6 y 7 y el otro, utilizando 2,5 y 7. No importa cuál de ellos se tome, lo que importa es la respuesta a la pregunta.

Un ejemplo donde se desea encontrar la mejor solución es el problema del vendedor ambulante, también conocido como el agente viajero, el cual consiste en que un vendedor tiene una lista de ciudades, cada una de las cuales debe ser visitada solamente una vez. Existen carreteras directas entre cada par de ciudades de la lista. Se debe encontrar el camino más corto que debería seguir el vendedor para visitar todas las ciudades, comenzando por una cualquiera y retornando a ella misma. En este caso, es necesario analizar todos los caminos posibles para tomar el mejor de ellos.

1.2.2.5. ¿Es consistente el conocimiento disponible para resolver el problema?

Hay problemas donde el conocimiento que se usa es completamente consistente. Un ejemplo de ellos es: dado los siguientes axiomas de un grupo multiplicativo,

1. x.y está definido $\forall$x,y elementos del grupo.
2. $(x=y \wedge y=z) \Rightarrow x=z$
3. $x=x$
4. $(x.y).z = x.(y.z)$
5. $I.x=x$ $\forall$x, donde I es el elemento identidad del grupo.
6. $x^{-1}.x = I$, donde x^{-1} es el inverso de x.
7. $x=y \Rightarrow z.x=z.y$
8. $x=y \Rightarrow x.z=y.z$

se desea demostrar que $\forall x \; x.I=x$.

Sin embargo, existen problemas que tienen inconsistencias. Un ejemplo es el problema de la diana, el cual consiste en que un hombre está de pie a 50m de una diana y quiere dar en el blanco con una pistola que dispara a una velocidad de 500 m/s. ¿A qué distancia debe apuntar por encima del blanco?. Razonemos: la bala tarda 0.1s en alcanzar el blanco, suponiendo que viaja en línea recta. La bala cae a una distancia igual a $1/2gt^2 = 1/2(9.8)(0.1)^2 = 0.049$ m = 4.9 cm. Si el hombre apunta 4.9 cm por encima del blanco daría en la diana. Pero se ha supuesto que la bala viaja en línea recta, lo que entra en conflicto conque viaja en una parábola.

1.2.2.6. ¿Cuál es el papel del conocimiento?

Existen problemas donde se necesita conocimiento sólo para restringir la búsqueda. Por ejemplo, en el problema del ajedrez, suponiendo potencia de computación ilimitada, se necesita de poca cantidad de conocimiento: sólo reglas para describir los movimientos legales del juego y un proceso de búsqueda adecuado. Usar conocimiento adicional sobre táctica y buena estrategia, ayudaría a restringir la búsqueda de la solución y acelerar la ejecución del programa. Sin embargo, hay problemas que necesitan gran cantidad de conocimiento tan sólo para reconocer una solución. Ejemplo: explorar los periódicos de Estados Unidos para decidir quién está apoyando a los demócratas o a los republicanos en una elección próxima. El programa tendría que saber cosas como los nombres de los candidatos de cada partido, el hecho de que si quiere bajar los impuestos apoya a los republicanos, el hecho de que si quiere educación para las minorías apoya a los demócratas, etc.

1.2.2.7. ¿Requiere la interacción con una persona?

Teniendo en cuenta este aspecto podemos distinguir dos tipos de problemas:

- Solitario, en el cual se le da a la computadora una descripción del problema y ella produce una respuesta sin comunicación intermedia y sin petición de una explicación de su razonamiento. Ejemplo: para demostrar un teorema matemático usando resolución, lo único que se desea es saber si existe una.
- Conversacional. Ejemplo: en problemas como el diagnóstico médico, el programa debe ser capaz de explicar su razonamiento, pues si no, no será aceptado por los médicos.

1.2.3. Tercer paso: Aplicar la mejor técnica de I.A. para el problema particular.

Para poder escoger la mejor técnica de I.A. a aplicar en el proceso de resolución de un problema particular es necesario estudiar, primeramente, las distintas técnicas de búsqueda que existen. A esto, precisamente, nos dedicaremos en las siguientes secciones de este capítulo.

1.3. Métodos de búsqueda a ciegas.

La búsqueda a ciegas es una colección de procedimientos que investigan el espacio de estados de manera exhaustiva pero ciega. Estos procedimientos se consideran métodos débiles, pues imponen restricciones mínimas a la búsqueda, en general son técnicas de solución de problemas de propósito general y pueden describirse independientemente de cualquiera sea el dominio del problema. Estos métodos usan solamente la información estructural y no hacen ninguna distinción cualitativa entre los nodos, respecto a su posibilidad de encontrarse sobre el camino deseado. En consecuencia, para los problemas con un extenso espacio de estados, la cantidad de alternativas que deben explorarse es tan grande que hace que su uso sea computacionalmente imposible. El número de nodos a explorar crece, en general, exponencialmente con la longitud del camino que representa la solución del problema. Esto genera una explosión combinatoria que estos métodos son incapaces de superar. No obstante, continúan formando el núcleo de la mayoría de los sistemas de I.A. Estudiaremos algunas de las técnicas de búsqueda a ciegas más usadas.

1.3.1. Algoritmo del museo británico.

Este procedimiento demuestra cuán ineficiente puede resultar un algoritmo de búsqueda. Consiste en colocar a un mono delante de una máquina de escribir y que presionando aleatoriamente las teclas genere todos los trabajos de Shakespeare existentes en el museo. Para generar una frase de 18 caracteres tendría una probabilidad de 1 en 27^{18}. De esta forma, el procedimiento consiste en generar todas las soluciones posibles y comprobar cuál es la correcta. Con un tiempo suficiente logra encontrar la solución optimal, sólo que es intratable computacionalmente.

1.3.2. Búsqueda primero a lo ancho.

Una búsqueda primero a lo ancho (breadth-first) explora primero todos los sucesores del nodo raíz. Si no se encuentra la meta, pasa a los sucesores del segundo nivel y así sucesivamente por niveles. Suponiendo que el objetivo a alcanzar es el nodo 7, el recorrido primero a lo ancho del espacio de búsqueda que se muestra en la figura 3.7, es 1-2-3-4-5-6-7. Este método simboliza a un explorador bastante conservador.

Si el número de máximo de hijos (o ramas) de un nodo es b y la profundidad de la solución es d, entonces el número de nodos en el nivel d es b^d y la cantidad de tiempo usada en la búsqueda es en el caso peor:

$1 + b + b^2 + ... + b^d$, la cual para grandes valores de d, puede ser aproximada por b^d. Es por esto que la complejidad temporal de este método de búsqueda es $O(b^d)$.

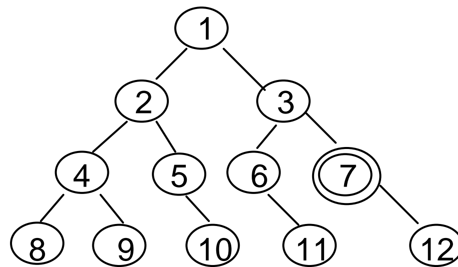


Fig. 1.3. Un ejemplo de espacio de búsqueda.

Este método tiene como ventaja que siempre encuentra el camino más corto a la solución, si ésta existe, aún en el caso de que el espacio de búsqueda sea infinito. Es efectivo cuando el factor de ramificación, o sea, el número promedio de hijos de un nodo, es pequeño, pues entonces la cantidad de nodos por niveles será pequeña y es mejor explorar un nivel antes de pasar al siguiente. Sin embargo, tiene las siguientes desventajas:

- Necesita mucha memoria. Como cada nivel del árbol tiene que ser almacenado completamente para poder generar el próximo nivel y la cantidad de memoria es proporcional al número de nodos almacenados, su complejidad espacial es también $O(b^d)$.
- Requiere mucho trabajo, especialmente si el camino más corto a la solución es muy largo, puesto que el número de nodos que necesita examinar se incrementa exponencialmente con la longitud del camino.
- Los operadores irrelevantes o redundantes incrementarán grandemente el número de nodos que deben explorarse.

Una forma de implementación de este método es usar un conjunto de caminos candidatos:

- Si el primer camino está encabezado por un nodo objetivo, ésta es la solución del problema.
- Si no,
  a) Remover el primer camino del conjunto de caminos candidatos, generar el conjunto de todas las posibles extensiones un nivel más de este camino y añadir las mismas al final del conjunto.
  b) Repetir el proceso.

En el ejemplo mostrado en la figura 1.3 este proceso se desarrollaría como sigue:

a) Conjunto de caminos candidatos inicial
[[1]]
b) Generando las extensiones de [1]
[[2,1] , [3,1]]
c) Al remover el primer camino candidato y generar sus extensiones [[4,2,1] , [5,2,1]] se obtiene el nuevo conjunto de caminos candidatos:
[[3,1] , [4,2,1] , [5,2,1]]
d) Removiendo y generando nuevamente las extensiones [[6,3,1] , [7,3,1]] se obtiene [[4,2,1] , [5,2,1] , [6,3,1] , [7,3,1]]
e) Así sucesivamente son obtenidos los siguientes conjuntos hasta llegar a:
[[7,3,1] , [8,4,2,1] , [9,4,2,1] , [10,5,2,1], [11,6,3,1]]

En este caso el camino inicial tiene como cabeza el nodo objetivo y el proceso termina en esta solución.

Algoritmo de Búsqueda Primero-a-lo-Ancho (BPA) (Breadth-First Search)
listas: Nodos=[], Succs=[] / variables inicializadas con la lista vacia */*
nodos: N / N es el nombre del nodo */*
BPA1: *Colocar nodo-inicial en Nodos*
BPA2: **if** *Nodos == []* **then** Salida= ***failure***
BPA3: *Eliminar de Nodos el primer nodo N*
BPA4: **if** *N == nodo-objetivo* **then** *Salida*= **success**
BPA5: *Expandir N colocando todos sus hijos en la lista Succs /* la expansión se realiza*
*aplicando algún operador a N */*
BPA6: *Nodos = Nodos + Succs, Succs = []*
BPA7: goto BPA2

El algoritmo anterior puede escribirse en el lenguaje PROLOG como sigue:

```
busq_ancho([[Nodo|Resto]|_],[Nodo|Resto]):- meta(Nodo).
busq_ancho([[Nodo|Resto]|Otros],Solucion):-
      bagof([M,Nodo|Resto],(sucesor(Nodo,M), not miembro(M,[Nodo|Resto])),
                                          NuevoCamino),
      append(Otros,NuevoCamino,Camino1),!,
      busq_ancho(Camino1,Solucion).
   %En caso de que Nodo no tenga sucesor, entonces
busq_ancho([[Nodo|Resto]|Otros],Solucion):- busq_ancho(Otros,Solucion).
```

Aquí meta/1 es un hecho que indica el nodo objetivo, miembro/2 es un predicado que determina si un elemento es miembro de una lista, sucesor/2 indica el sucesor de un nodo dado y append/3 concatena dos listas.

1.3.3. Búsqueda primero en profundidad.

La búsqueda primero en profundidad (depth-first) explora, primeramente, el nodo raíz y luego el sucesor de éste ubicado en la rama más a la izquierda. Si este nodo es el objetivo, entonces hemos encontrado el camino. Si no, se continúa extendiendo este camino tomando siempre el primer sucesor. Si el nodo no tiene más sucesores, se pasa al siguiente sucesor de su predecesor, o sea, se retrocede al nivel anterior para tomar el otro sucesor y así sucesivamente, hasta alcanzar el objetivo o hasta una profundidad determinada. El recorrido primero en profundidad del espacio de búsqueda mostrado en la figura 3.7 es 1-2-4-8-9-5-10-3-6-11-7. Este método simboliza a un explorador que toma riesgos.

Este método tiene como ventaja la eficiencia que se alcanza en el uso de la memoria. Si la longitud máxima de una rama del árbol es *d*, como sólo se necesita almacenar el camino actual, entonces la complejidad espacial del algoritmo es O(*d*). En la práctica la búsqueda por este método se limita por el tiempo y no por el espacio. Sin embargo, tiene como desventaja que si existen

ramas infinitas, puede no encontrar la solución al problema, aún teniéndola. Es por esto que, en ocasiones, se requiere que se defina un corte a una profundidad arbitraria para evitar, lo más posible, caer en caminos o lazos infinitos. Si la profundidad de corte seleccionada c es menor que la profundidad de la solución d, el algoritmo terminará sin encontrar una solución, mientras que si $c > d$, se paga un precio alto, $O(b^c)$, en términos del tiempo de ejecución, donde b es el número máximo de hijos (o ramas).

Algoritmo de Búsqueda Primero-en-Profundidad (BPP) (Depth-First Search)
listas: Nodos=[], Succs=[] / variables inicializadas con la lista vacía */*
nodos: N / N es el nombre del nodo */*
BPP1: *Colocar nodo-inicial en Nodos*
BPP2: **if** *Nodos == []* **then** *Salida=* ***failure***
BPP3: *Eliminar de Nodos el primer nodo N*
BPP4: **if** *N == nodo-objetivo* **then** *Salida=* **success**
BPP5: *Expandir N colocando todos sus hijos en la lista Succs /* la expansión se realiza*
*aplicando algún operador a N */*
BPP6: *Nodos = Succs + Nodos, Succs = []*
BPP7: goto BPA2

El algoritmo anterior puede escribirse en el lenguaje PROLOG como sigue:

```
busq_prof(Nodo,[Nodo]):- meta(Nodo).
busq_prof(Nodo,[Nodo|Solucion]):- sucesor(Nodo,Nodo1),
                                  busq_prof(Nodo1,Solucion).
```

Aquí meta/1, nuevamente, es un hecho que indica el nodo objetivo y sucesor/2 indica el sucesor de un nodo dado.

La búsqueda primero en profundidad es mejor cuando el nodo objetivo está situado en la porción inferior izquierda del árbol de búsqueda, mientras que la búsqueda primero a lo ancho es mejor cuando el nodo objetivo está situado en la porción superior derecha de dicho árbol, según se muestra en la figura 3.8.

Una variante del método de búsqueda primero en profundidad es la búsqueda iterativa primero en profundidad. Esta consiste en realizar una búsqueda primero en profundidad con corte 1, luego otra para el corte 2 y así sucesivamente, incrementando en uno la profundidad de corte, hasta encontrar la solución. Como este método nunca explora un nodo hasta que todos los nodos de los niveles anteriores han sido examinados, se garantiza encontrar la solución óptima. Además, como en cada momento se está ejecutando una búsqueda primero en profundidad, la complejidad espacial es $O(d)$. Aunque parezca lo contrario, se puede demostrar que la complejidad temporal es $O(b^d)$, al igual que en la búsqueda primero a lo ancho. La razón de esto es que como el número de nodos, en un nivel dado del árbol, crece exponencialmente con la longitud del camino o profundidad, casi todo el tiempo se gasta en el nivel más profundo. El recorrido de este método para la figura 1.3 es 1-2-3-1-2-4-5-3-6-7.

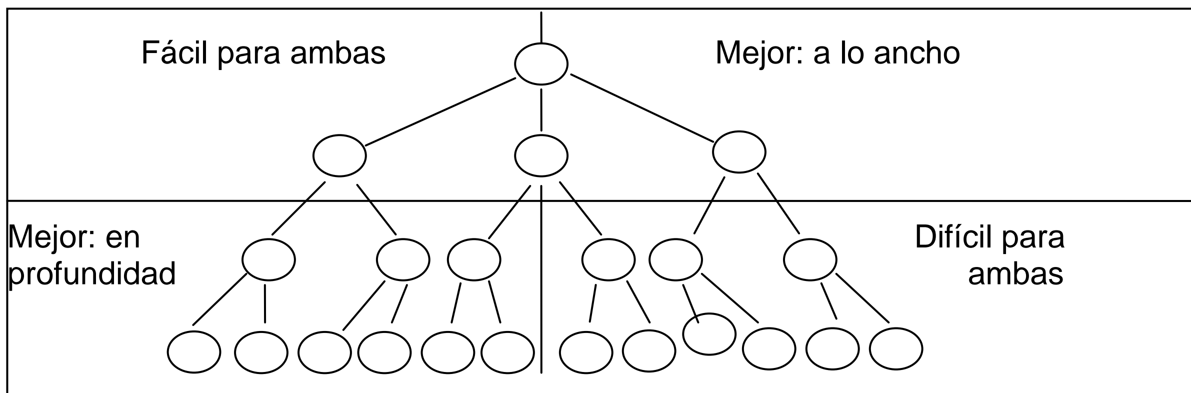


Fig. 1.4. Comparación de los dos métodos de búsqueda.

1.3.4. Búsqueda en árboles y/o.

Los algoritmos para búsqueda primero a lo ancho y primero en profundidad en árboles y/o son similares a los anteriormente estudiados, sólo se diferencian en la verificación de las condiciones de terminación. Cada vez que se genera un nodo o un grupo de nodos, el algoritmo tiene que chequear si se satisfacen las condiciones de terminación de acuerdo al tipo de nodo, ya sea Y u O.

1.3.5. Búsqueda en sistemas de producción.

En los sistemas de producción se utilizan generalmente dos tipos de métodos de búsqueda. Un método consiste en realizar la búsqueda desde el estado inicial a un estado final, o sea, desde las evidencias a las conclusiones, y recibe el nombre de *encadenamiento hacia delante* (forward chaining) o *enfoque guiado por datos* (datadriven). El otro método realiza la búsqueda en dirección contraria, comenzando desde el estado objetivo y terminando en un estado inicial, o sea, el razonamiento se realiza desde una hipótesis (objetivo) hasta las evidencias primarias necesarias para refutar o confirmar dicha hipótesis. En este caso el método recibe el nombre de *encadenamiento hacia atrás* (backward chaining) o *enfoque guiado por objetivos* (goaldriven). Estudiemos con profundidad estos métodos, así como una combinación de ambos.

1.3.5.1. Búsqueda con encadenamiento hacia adelante (enfoque guiado por datos).

En la búsqueda con encadenamiento hacia delante se comienza a construir el árbol situando como raíz al estado inicial. El siguiente nivel del árbol se genera encontrando todas las reglas cuyas partes izquierdas concuerden con el nodo raíz y usando sus partes derechas para crear los nuevos estados. De esta

manera el proceso continúa hasta generar un estado que concuerde con el estado meta.

En este método las reglas son sólo aplicables si su parte condición es satisfecha por la B.D. El siguiente procedimiento da la idea algorítmica de un módulo de aplicación de reglas sencillo que está basado en este enfoque.

Procedimiento *generar*:

1) Identificar el conjunto S de reglas aplicables.
2) Mientras S no sea vacío,
   a) Seleccionar una regla R de S.
   b) Aplicar R, generando los nuevos estados y añadiéndolos a la B.D.
   c) Si se generó el estado objetivo entonces TERMINAR y EXITO.
   d) Si no, llamar nuevamente al procedimiento *generar*.
   e) Eliminar R de S y anular el efecto de aplicar R.

Este proceso de encadenamiento hacia delante tiene un carácter no determinístico, pues el ordenamiento de las reglas aplicadas no está explícitamente definido en el caso de que el conjunto de reglas aplicables esté formado por más de una regla. Ellas forman, precisamente, el llamado *conjunto conflicto*. En el capítulo anterior mencionamos algunas técnicas de resolución de conflictos en los sistemas de producción.

<u>Base de reglas</u>

R_1: Si X es divisible por 12, entonces X es divisible por 6.
R_2: Si X es divisible por 20, entonces X es divisible por 10.
R_3: Si X es divisible por 6, entonces X es divisible por 2.
R_4: Si X es divisible por 10, entonces X es divisible por 5.

<u>Base de Datos inicial</u> N es divisible por 12.
N es divisible por 20.

Fig. 1.5. Una base de reglas y una B.D. inicial.

Analicemos un ejemplo. Supongamos que tenemos la base de reglas y la B.D. inicial mostradas en la figura 1.5. El problema es determinar si N es divisible por 5. Comencemos a aplicar el procedimiento *generar*, suponiendo que el sistema siempre escoge la primera regla que aparece:

$S=\{R_1,R_2\}$. Se selecciona R_1.

Ahora la B.D. es: N es divisible por 12.
N es divisible por 20.
N es divisible por 6.

Ningún hecho concuerda con el estado objetivo. Se llama de nuevo a *generar*.

$S_1=\{R_2,R_3\}$. R_1 no está incluido en S_1 porque su aplicación no cambiaría el estado en curso de la B.D. Se aplica ahora R_2 y la B.D. es:

N es divisible por 12.
N es divisible por 20.
N es divisible por 6.
N es divisible por 10.

Nuevamente llamamos a *generar*.

$S_2=\{R_3,R_4\}$. R_1 y R_2 no están en S_2 porque no son aplicables. Se selecciona ahora a R_3 y la B.D. es:

N es divisible por 12.
N es divisible por 20.
N es divisible por 6.
N es divisible por 10.
N es divisible por 2.

Al llamar nuevamente a *generar*,

$S_3=\{R_4\}$. Se aplica R_4 y se llega a la solución del problema.

Un listado de cómo se consiguió esta solución da el orden en que se utilizaron las reglas: R_1, R_2, R_3, R_4. El procedimiento finaliza realizando lo siguiente:

- Anula el efecto de R_4, es decir, elimina N es divisible por 5 de la B.D.
- Elimina R_3 de S_2.
- Anula el efecto de R_3, eliminando N es divisible por 2 de la B.D.
- Se continúa en el ciclo del paso 2).
- Se selecciona R_4 de S_2 y se aplica.

Se obtiene otra solución: R_1, R_2, R_4. El proceso continúa hasta que se hayan establecido todas las maneras de poder conseguir la solución. El espacio de búsqueda para este problema está representado en la figura 1.6. Note cómo en este espacio cada operador de transición de estados es precisamente una regla de producción.

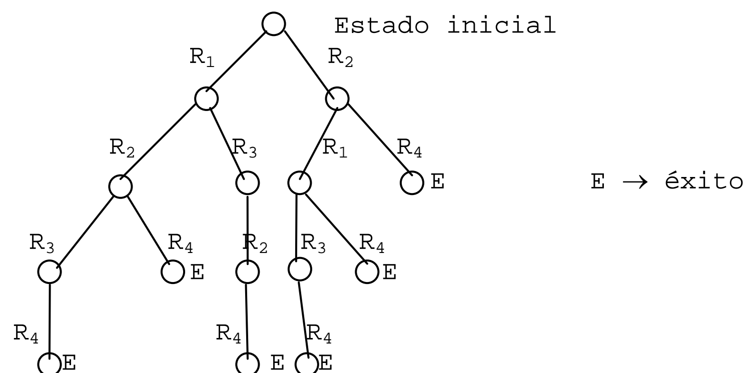


Fig. 1.6. Espacio de búsqueda del ejemplo del encadenamiento hacia delante.

La búsqueda con encadenamiento hacia delante tiene las siguientes ventajas:

- Simplicidad
- Puede utilizarse para proporcionar todas las soluciones a un problema.

Sin embargo, tiene la desventaja de que el comportamiento del sistema, al intentar solucionar un problema, puede ser ineficaz y parecer también desatinado, ya que algunas de las reglas ejecutadas podrían no estar relacionadas con el problema en cuestión. En el ejemplo estudiado las reglas 1 y 3 no nos acercan a la solución del problema.

1.3.5.2. Búsqueda con encadenamiento hacia atrás (enfoque guiado por objetivos).

En el encadenamiento hacia atrás se comienza a construir el árbol situando como raíz al estado objetivo. El siguiente nivel del árbol se genera encontrando todas las reglas cuyas partes derechas concuerden con el nodo raíz y usando las partes izquierdas para crear los nuevos estados. Todas estas son las reglas

que generarían el estado que queremos si pudiésemos aplicarlas. Este proceso continúa hasta generar un estado que concuerde con el estado inicial.

En este enfoque el sistema centra su atención, únicamente, en las reglas que son relevantes para el problema en cuestión. En él el usuario comienza especificando un objetivo mediante la declaración de una expresión E cuyo valor de verdad hay que determinar. El siguiente procedimiento da la idea algorítmica de un módulo simple de aplicación de reglas que utiliza este enfoque.

Procedimiento *validar*: Tiene como entrada a una expresión X.

1) Resultado:= falso
2) Identificar el conjunto S de reglas aplicables que tengan a X en su parte derecha.
3) Si S es vacío, entonces pedir al usuario que añada reglas a la base.
4) Mientras Resultado sea falso y S no sea vacío,
   a) Seleccionar y eliminar una regla R de S.
   b) C:= parte izquierda de R.
   c) Si C es verdadero en la B.D., entonces Resultado:= verdadero.
   d) Si C no es ni verdadero ni falso en la B.D., entonces
      i) Llamar a *validar* pasándole como argumento a C.
      ii) Si *validar* retornó verdadero, entonces Resultado:= verdadero.
5) RETORNAR VERDADERO.

Para ejecutar este procedimiento, el usuario pregunta por el valor de verdad de *validar* tomando como argumento la expresión E, donde E puede ser, por ejemplo, “N es divisible por 5”. Lo primero que hace este procedimiento es identificar todas las reglas que tienen a E en el miembro derecho suponiendo que se hacen sustituciones apropiadas. Si no existen tales reglas, se pide al usuario que proporcione alguna. Si hay más de una regla, el sistema selecciona una, usando alguna estrategia de resolución de conflictos. Cuando una regla es seleccionada, su parte condición C es verificada con respecto a la B.D. Si C es verdadera en la B.D., entonces se establece la verdad de E y el proceso puede terminar con éxito. Si C es falsa en la B.D., entonces no se puede utilizar R para establecer la verdad de E y se selecciona otra regla de S. Si C es desconocida (es decir no es verdadera ni falsa en la B.D.), entonces se le considera como nuevo subobjetivo y se intenta establecer su valor de verdad llamando recursivamente a *validar* con C como argumento. Si *validar*(C) es verdadero, entonces la regla R es aplicable, se establece la verdad de E y el proceso puede terminar con éxito. Si *validar*(C) es falso, entonces se selecciona otra regla de S. Así, el proceso opera hacia atrás a partir del objetivo, intentando alcanzar subobjetivos que puedan establecer dicho objetivo por sí mismos.

Analicemos el ejemplo estudiado en la sección anterior, considerando las reglas y la B.D. inicial de la figura 1.5. Supongamos nuevamente que se quiere determinar si N es divisible por 5.

S={R_4}, pues R_4 es la única regla con X es divisible por 5 en el miembro derecho. Se genera el subobjetivo N es divisible por 10 mediante la sustitución de N por X en la parte condición de R_4. Como N es divisible por 10 no es ni verdadera ni falsa en la B.D. se llama a *validar* tomando como argumento la expresión “N es divisible por 10”.

S_1={R_2}. Se genera el subobjetivo N es divisible por 20 que es verdadero en la B.D. Así la llamada inicial a *validar* termina con verdadero.

Los sistemas guiados por objetivos preguntan tanto al usuario como a la B.D. al determinar la verdad de un subobjetivo. Así, el usuario no necesita entrar todos los datos disponibles inicialmente, sino que puede esperar hasta que el sistema pida datos.
Si se requieren todas las maneras de establecer E, se elimina de la condición del ciclo en el paso 4) Resultado=falso y se deja sólo S no es vacío.
El procedimiento *validar* es simple, en el sentido de que supone que las partes izquierdas de las reglas constan de declaraciones únicas, o sea, que no contienen conectivas lógicas. Es necesario, entonces, extender este procedimiento si se desea trabajar con este método de búsqueda.
El encadenamiento hacia atrás tiene como ventaja que no solicita datos ni aplica reglas que no estén relacionadas con el problema en cuestión.

Nótese que en ambos métodos de búsqueda pueden usarse las mismas reglas, tanto para razonar hacia delante desde el estado inicial como para razonar hacia atrás desde el estado objetivo. Dependiendo de la topología del espacio del problema, puede ser significativamente más eficiente la búsqueda en un sentido que en otro. Tres factores influyen en esto:

- ¿Existen más estados iniciales posibles o más estados finales?. Preferiríamos movernos desde un conjunto de estados lo más pequeño posible hacia el conjunto de estados mayor, y por tanto, más fácil de encontrar. Un ejemplo que ilustra esto es el siguiente: es más fácil conducir desde un lugar desconocido a nuestra casa que desde ella a ese lugar desconocido. Existen muchos más lugares que nos ayudan a llegar a nuestra casa de los que nos pueden ayudar a llegar al lugar desconocido. Si podemos llegar a cualquiera de ellos podemos llegar a casa fácilmente. Por tanto si nuestra posición de partida es nuestra casa y nuestra meta el lugar desconocido, es mejor razonar hacia atrás. Otro ejemplo es la integración simbólica, donde el estado inicial es una fórmula que tiene el símbolo integral y el estado meta es un conjunto de fórmulas que no tengan el símbolo integral. Luego empezamos con un único estado inicial y un enorme número de estados metas. Es mejor razonar hacia delante, usando las reglas de integración a partir de la fórmula inicial, que empezar con una expresión arbitraria libre de integrales, usando las reglas de diferenciación e intentar generar la integral inicial. En los sistemas de diagnóstico hay pocas metas posibles, por lo que es mejor usar el encadenamiento hacia atrás. Sin embargo, en el juego del ajedrez la única alternativa posible es utilizar el encadenamiento hacia delante, pues el número de hipótesis en este caso es virtualmente ilimitado.
- ¿En qué dirección el factor de ramificación, o sea, el promedio de nodos hijos es mayor? Nos gustaría avanzar siempre en la dirección en que éste sea menor. Un ejemplo es el problema de demostrar teoremas. Nuestros estados iniciales son un pequeño conjunto de axiomas y los estados finales los teoremas a demostrar. Ninguno de estos conjuntos es significativamente más grande que el otro. Consideremos, entonces, el factor de ramificación. A partir de un conjunto pequeño de axiomas podemos derivar un número muy elevado de teoremas. Usando el otro enfoque de este elevado número de teoremas debe regresarse a un pequeño número de axiomas. El factor de ramificación es significativa-

mente mayor si vamos hacia delante, desde los axiomas hasta los teoremas, que al revés. Por lo tanto, es mejor razonar hacia atrás. De hecho, los matemáticos ya se han dado cuenta de esto. Uno de los primeros programas de I.A., el Lógico Teórico (Newell 1963) usaba el razonamiento hacia atrás para demostrar diversos teoremas del primer capítulo de los Principia de Russell y Whitehead.

- ¿Se le pedirá al programa que justifique su proceso de razonamiento? Es importante avanzar en la dirección que concuerde más con la forma en que piensa el usuario. Por ejemplo, los doctores se niegan a aceptar el consejo de un programa de diagnóstico que no pueda explicar su razonamiento. MYCIN, un S.E. que diagnostica enfermedades infecciosas, razona hacia atrás para determinar la causa de la enfermedad del paciente. Para ello usa reglas tales como "si el organismo tiene el siguiente conjunto de características determinadas por los resultados del laboratorio, entonces es probable que el organismo sea X". Al razonar hacia atrás puede responder a preguntas como ¿por qué debería realizar esa comprobación que acaba de pedir? y tendría respuestas como "porque ayudaría a determinar si el organismo X está presente".

1.3.5.3. Búsqueda bidireccional.

La búsqueda bidireccional es una combinación de las dos búsquedas anteriores. Consiste en realizar simultáneamente una búsqueda con encadenamiento hacia delante desde el estado inicial y una búsqueda con encadenamiento hacia atrás desde el estado objetivo, hasta que ambas se encuentren. El camino a la solución se obtiene, entonces, concatenando el camino desde el estado inicial con el inverso del camino desde el estado objetivo. Para poder garantizar que ambas búsquedas se encuentren al menos una de ellas debe seguir la estrategia de la búsqueda primero a lo ancho. La utilización de este método exige, además, que los operadores del problema sean inversibles.

Suponiendo que las comparaciones para identificar estados comunes pueden ser hechas en un tiempo constante por nodo, la complejidad temporal es $O(b^{d/2})$, ya que cada búsqueda sólo necesita llegar hasta la mitad de la profundidad de la solución. La complejidad espacial es también $O(b^{d/2})$, pues una de las búsquedas usa la estrategia de búsqueda primero a lo ancho. Recuerde que b es el máximo número de hijos (o ramas) de los nodos y d, la profundidad de la solución.

Este método parece atrayente si el número de nodos de cada paso crece exponencialmente con el número de pasos que se dan. Resultados empíricos sugieren que para buscar a ciegas, esta estrategia es realmente efectiva. Desafortunadamente, otros resultados sugieren que para una búsqueda heurística lo es mucho menos, pues puede ocurrir que las búsquedas no se encuentren y se necesite más trabajo que el que habría dado realizar una sola de ellas. En un programa construido cuidadosamente pudiera emplearse con éxito.

1.3.7. Búsqueda de soluciones múltiples.

En ocasiones necesitamos encontrar más de una solución al problema que queremos resolver. Esta búsqueda podemos implementarla de dos formas:

- Removiendo el camino de búsqueda encontrado.
  Este método consiste en lo siguiente:
  1) Encontrar una solución utilizando algún método de búsqueda.
  2) Eliminar del árbol de búsqueda todos los nodos que forman parte del camino encontrado a la solución.
  3) Ir al paso 1).
- Removiendo el último nodo del camino.
  Este método es similar al anterior. Su única diferencia radica en que en vez de eliminar el camino encontrado, se elimina sólo el último nodo de este camino, repitiéndose nuevamente el proceso.

1.4. Búsqueda heurística.

Analicemos el problema del vendedor ambulante. Para resolverlo usando alguna de las técnicas de búsqueda a ciegas estudiadas, se exploraría el árbol de todos los posibles caminos y se devolvería aquel de menor longitud. Si existen N ciudades, el número de caminos diferentes entre ellas es (N-1)!. Explorar cada camino nos tomaría un tiempo proporcional a N y, por tanto, el tiempo total sería proporcional a N!. Tomando 10 ciudades tardaríamos un tiempo de 10!=3 628 000 para solucionar el problema. Este fenómeno se llama *explosión combinatoria*.

Para solucionar eficientemente la mayoría de los problemas difíciles, a menudo es necesario comprometer los requerimientos de movilidad y sistematicidad, y construir una estructura de control que, aunque no nos garantice que encontremos la mejor respuesta, por lo menos proporcione una respuesta suficientemente buena.

3.4.1. Definición de heurística.

La palabra *heurística* viene del griego heuriskein que significa descubrir. Ella es también el origen de eureka, que se deriva de la famosa exclamación de Arquímides heurika (“lo encontré”), que lanzó al descubrir un método para determinar la pureza del oro. Feigenbaum y Feldman la definen así: “Una heurística (regla heurística o método heurístico) es una regla para engañar, simplificar o para cualquier otra clase de ardid, la cual limita drásticamente la búsqueda de soluciones en grandes espacios de solución”.

La posibilidad de efectuar la búsqueda de una forma más eficiente se basa en el uso de alguna información específica para el problema a solucionar. Esta información, aunque imprecisa e incompleta, permite distinguir cuáles de los nodos dirigen mejor el avance hacia la meta y permite realizar este avance siempre en la dirección que momentáneamente tiene la mejor perspectiva. Se llama *búsqueda heurística* a los métodos de búsqueda que usan este tipo de

información. Precisamente por esto a estos métodos, generalmente, se les llaman *métodos fuertes*. Las técnicas heurísticas son como las guías turísticas: son buenas cuando apuntan a direcciones interesantes y son malas cuando apuntan a callejones sin salida. Usando buenas técnicas heurísticas, podemos esperar lograr buenas soluciones a problemas difíciles, como el del vendedor ambulante, en un tiempo menor que el exponencial.

Analicemos un ejemplo de heurística. Supongamos que un hombre se encuentra en una extensa llanura y tiene sed. Caminando ha llegado a una pequeña elevación que es la única en esa región y se sube a ella. Su objetivo es, por supuesto, encontrar agua. Desde la elevación el hombre observa el panorama que se presenta en la figura 3.11. ¿Qué dirección debe seguir para hallar el agua?. Evidentemente la vegetación verde es un indicio de que en esa zona hay humedad, por lo que es muy probable que exista agua en la superficie o subterránea. Por otro lado, el movimiento de animales puede indicar que ellos se dirigen allí a beber, por lo que, posiblemente, el agua está en la superficie. El hombre, al usar esta información, puede decidir dirigirse primero hacia el norte, y si no encuentra solución, o sea, no halla agua, dirigirse al oeste. En este caso, la vegetación verde y el movimiento de animales han sido utilizados como heurísticos.

La búsqueda heurística representa una de las herramientas clásicas de la I.A. En favor de su uso, se pueden mencionar las razones siguientes:

- Sin ellas estaríamos atrapados en la explosión combinatoria.
- Pocas veces necesitamos la solución óptima, usualmente una buena aproximación satisface nuestras exigencias. De hecho, en la vida real las personas, generalmente, no buscan la mejor solución a un problema; tan pronto como encuentran una, abandonan la búsqueda. Por ejemplo, al tratar de parquear un carro, cuando se encuentra un espacio disponible se parquea, aunque más adelante exista uno mejor.
- Aunque las aproximaciones producidas por ellas pueden, en el peor de los casos, no ser muy buenas, los peores casos surgen raramente en el mundo real.

Esto conduce a otra forma de definir la I.A., como el estudio de técnicas para resolver problemas exponencialmente difíciles en un tiempo polinomial explotando el conocimiento sobre el dominio del problema.

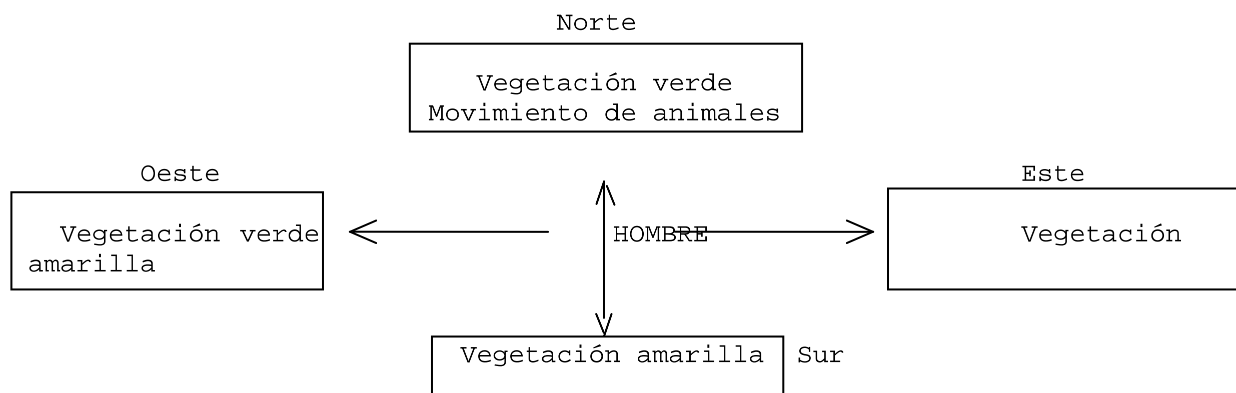


Fig. 1.6. Panorama visto por el hombre desde la elevación.

La información heurística puede incorporarse en un método de búsqueda de varias formas:

- En las reglas mismas. Por ejemplo, las reglas para el problema del ajedrez pueden describir no solamente el conjunto de movimientos legales, sino también un conjunto de movimientos "sensatos" según el escritor de las reglas.
- Como una función heurística que evalúe los estados individuales del problema y determine hasta qué punto son deseables. Las funciones heurísticas bien diseñadas guían eficientemente el proceso de búsqueda en la dirección más provechosa. El programa que las utilice puede intentar minimizar dicha función o maximizarla según sea apropiado. Ejemplos de funciones heurísticas son las siguientes:

  Ajedrez: la ventaja de piezas nuestras sobre el oponente.

  Rompecabezas de 8 piezas: el número de baldosas colocadas correctamente.

  Vendedor Ambulante: la suma de las distancias recorridas hasta ahora o tomar la distancia más pequeña de entre las de los sucesores del nodo actual.

  Titafor: Puntuación otorgada de la siguiente forma: un punto para cada fila, columna o diagonal en donde podamos ganar y en la que tengamos una pieza y dos puntos para cada fila, columna o diagonal que sea posible ganadora y en la que tengamos 2 piezas.

  El papel exacto de las funciones heurísticas se estudiará en cada uno de los métodos de búsqueda heurística.

3.4.2. Búsqueda por el incremento mayor (hill-climbing).

Este método consiste en realizar en cada momento la transición hacia el estado que se encuentre más cerca del estado objetivo. Este criterio puede ser implementado definiendo una función de selección de estados que escoja el estado en el cual la función de evaluación de estados produzca el mayor incremento.

Una de las ventajas de este método de búsqueda es que tiende a reducir el número de nodos visitados para alcanzar una solución. Sin embargo, este método tiene el problema de que puede no terminar nunca, debido a que como el algoritmo sólo almacena el estado actual no tiene manera de saber dónde ha estado y puede caer en un ciclo. Una forma de solucionar esta dificultad es mantener una lista de estados visitados y nunca revisitar uno de éstos, lo cual garantiza que el algoritmo terminará siempre que el espacio de estados del problema sea finito.

Apliquemos este método al problema del vendedor ambulante:

1) Seleccionar arbitrariamente una ciudad de partida.
2) Para seleccionar la siguiente ciudad, mirar todas las ciudades que aún no se han visitado. Seleccionar aquella que sea la más cercana a la ciudad actual e ir a ella.
3) Repetir el paso 2) hasta que se hayan visitado todas las ciudades.

Este procedimiento se ejecuta, para N ciudades, en un tiempo proporcional a N^2, lo cual constituye una mejora significativa sobre el N! que se obtuvo utilizando un método de búsqueda a ciegas.

3.4.3. Búsqueda por el mejor nodo (best-first search).

La búsqueda por el mejor nodo es una forma de combinar las ventajas de las búsquedas en profundidad y de anchura en un único método. En cada paso del proceso de búsqueda por el mejor nodo, seleccionamos el más prometedor de aquellos nodos que se han generado hasta el momento. Esto se realiza aplicando una función heurística apropiada a cada uno de ellos. Entonces expandimos el nodo elegido usando las reglas para generar a sus sucesores. Si uno de ellos es una solución podemos terminar. Si no, todos esos nodos se añaden al conjunto de nodos generados hasta ahora. Se selecciona de nuevo el más prometedor y el proceso continúa. Lo que sucede usualmente es que se realiza un proceso de búsqueda en profundidad mientras se explora una rama prometedora. Como ocasionalmente no se encuentra solución, se explora otra rama previamente ignorada y que ahora parece más prometedora.

En este método de búsqueda la función heurística que estima los méritos de cada nodo generado está definida como:

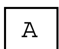
$$f(n) = g(n) + h(n),$$

donde g(n) es una medida del costo del camino desde el nodo inicial al nodo actual n y h(n) es un estimado del costo adicional de llegar desde el nodo n al nodo meta. g(n) no es, necesariamente, el costo del camino óptimo desde el nodo inicial al nodo actual, pues puede haber caminos mejores no recorridos todavía. g(n) simplemente se calcula como la suma de los costos de los arcos del camino actual. La función h(n), sin embargo, es típicamente heurística y explota el conocimiento sobre el domino del problema, ya que el "mundo" entre el nodo n y el meta no ha sido explorado. Por supuesto, no existe un método general para construir h(n), sino que depende del problema particular.

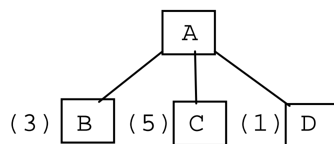
La figura 3.12 muestra el principio de un procedimiento de búsqueda por el mejor nodo. Los números que aparecen al lado de los nodos son los valores de f(n). Note cómo en cada paso se selecciona el nodo de menor valor de f(n), generando sus sucesores.

Al algoritmo de búsqueda que utiliza la función f(n) se le llama *algoritmo A*. Este algoritmo no garantiza encontrar el camino óptimo a la solución, pues éste sólo calcula un estimado del costo para alcanzar el nodo objetivo.

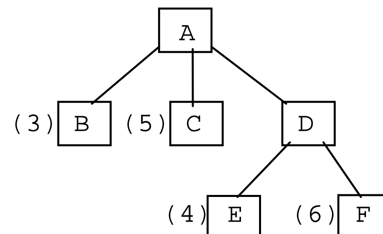
Paso 1



Paso 2



Paso 3



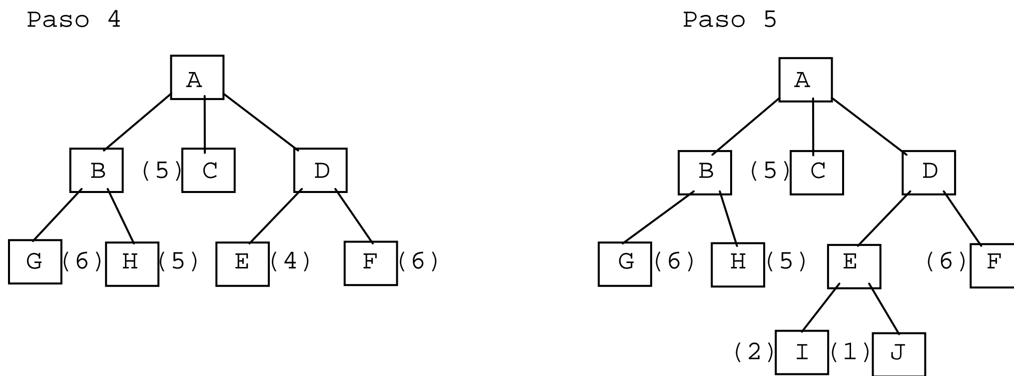


Fig. 3.12. Un ejemplo de búsqueda por el mejor nodo.

Se puede definir la función $f^*(n)$ que, en cualquier nodo n, calcule el costo real del camino óptimo desde el nodo inicial al nodo n ($g^*(n)$) más el costo del camino óptimo desde el nodo n al nodo objetivo ($h^*(n)$), es decir

$$f^*(n) = g^*(n) + h^*(n),$$

donde $h^*(n)$ es una función de evaluación heurística admisible, o sea, $\forall n\ h^*(n)$ es menor o igual que la distancia real mínima a la meta. Es precisamente por esto que el algoritmo que usa este estimador garantiza siempre encontrar, si existe, el camino óptimo a la solución. A este algoritmo se le conoce con el nombre de *algoritmo A^*.*

Por ejemplo, para el problema del rompecabezas de 8 piezas puede tomarse $h^*(n)$ como el número de piezas que no están en su posición final. Otra variante, que contiene más información heurística pero requiere más cálculos, es la suma de las distancias desde la posición actual de cada pieza a su posición correcta.

Podemos hacer algunas observaciones interesantes sobre este algoritmo:

- Al incorporar g^* en f^* no siempre se elige como el siguiente nodo a expandir el que parece más cercano a la meta. Esto es útil si nos preocupa el camino que elijamos. Si sólo nos importa llegar a una solución de la forma que sea, ponemos $g^*=0$ y con esto estamos eligiendo el nodo más cercano a la meta. Si queremos encontrar un camino en el menor número de pasos, entonces ponemos el costo de un nodo a su sucesor constante igual a 1. Si el costo de los operadores varía, se refleja en cada nodo y se calcula el camino de costo mínimo.
- Cuando $h^* = 0$ y $g^* = d$, donde *d* es el nivel de profundidad del nodo, el algoritmo A^* es idéntico al método de búsqueda primero a lo ancho.

Este algoritmo pudiera ser implementado mediante los siguientes pasos:

Entrada: Nodos inicial y meta.

Salida: Camino de costo mínimo del nodo inicial al meta, si existe. En otro caso, retorna NO.

1) Inicializar la lista Abiertos con el nodo inicial y Cerrados como la lista vacía.
2) Si Abiertos es la lista vacía, entonces TERMINAR y RETORNAR NO.
3) Si no,
   a) Remover el primer nodo de Abiertos y llamarle MejorNodo. Colocarlo en Cerrados.
   b) Si MejorNodo es el nodo meta, entonces TERMINAR y RETORNAR EL CAMINO, el cual se obtiene moviéndonos a través de los apuntadores hacia atrás en la lista Cerrados.

c) Si no, generar la lista de los sucesores de MejorNodo y para cada uno de ellos realizar lo siguiente:

i) Poner a Sucesor apuntando a MejorNodo. Estos apuntadores hacia atrás permiten recuperar el camino.

ii) Calcular f^*(Sucesor) = g^*(Sucesor) + h^*(Sucesor), donde g^*(Sucesor) = g^*(MejorNodo) + costo de ir de MejorNodo a Sucesor.

iii) Insertar a Sucesor en Abiertos según su valor de f^* ordenados de menor a mayor.

d) Ir al paso 2).

La principal desventaja del algoritmo A^* es el requerimiento de memoria para mantener la lista de nodos generados no procesados, por lo que, en este aspecto, este algoritmo no es más práctico que la búsqueda primero a lo ancho. Dos variantes del mismo son:

- Mantener sólo en la lista de nodos no visitados los mejores n nodos. El costo de esta técnica es la pérdida de la solución óptima, pues un nodo malo localmente se puede sacar de la lista y él podría conducir al óptimo global. Esta técnica se conoce como *beam search*.
- El algoritmo IDA^* (Iterative-Deepening-A^*), el cual reduce drásticamente los requerimientos de memoria del algoritmo A^* sin sacrificar la optimalidad de la solución encontrada. Cada iteración del algoritmo es una búsqueda primero en profundidad completa que mantiene la pista del costo $f^*(n)$ de cada nodo generado. Tan pronto como este costo exceda alguna cota se produce un corte y se realiza un retroceso al nodo más recientemente generado. La cota comienza con el estimado heurístico del nodo inicial y en cada iteración se incrementa con el valor mínimo que excedió la cota previa.

3.4.4. Búsqueda heurística en árboles y/o o algoritmo AO*.

Para encontrar soluciones en un árbol y/o necesitamos un algoritmo similar al A^*, pero con la capacidad de manejar los arcos *Y* apropiadamente. Este algoritmo debería encontrar un camino desde del nodo inicial a un conjunto de nodos que representan los estados meta. Nótese que puede ser necesario obtener más de un estado meta, pues cada rama de un arco *Y* puede conducir a su propio nodo meta.

Para ver por qué el algoritmo A^* no es adecuado para la búsqueda en árboles y/o, consideremos la figura 3.13. Los números representan el valor de f^* en cada nodo. El problema radica en que la elección del nodo a expandir no sólo depende del valor de f^* en ese nodo, sino también de si ese nodo es parte del mejor camino actual a partir del nodo inicial. El nodo simple más prometedor es C, pero no forma parte del mejor camino actual, pues hay que incluir al nodo D.

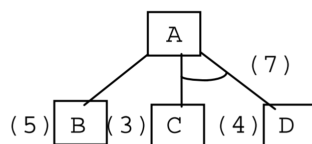


Fig. 3.13. Un árbol y/o donde no es posible utilizar A^*.

Los nodos de un árbol y/o pueden ser clasificados en nodos primitivos, que son los que se corresponden con problemas que se resuelven directamente, nodos muertos, que son los que se corresponden con problemas no descomponibles y que no tienen solución y nodos intermedios. Para cada uno de ellos el estimado heurístico $h^*(n)$ se puede definir como:

- Si n es un nodo primitivo, entonces $h^*(n) = 0$
- Si n es un nodo muerto, entonces $h^*(n) = \infty$
- Si n es un nodo intermedio con j conectores, entonces $h^*(n)$ es el mínimo de los costos calculados para cada uno de los j conectores. Si el conector es *Y*, entonces su costo es $\sum_k (\cos to(n, n_k) + h^*(n_k))$, donde k es el número de nodos del conector. Si por el contrario, el conector es *O*, entonces su costo es costo(n,n_k) + $h^*(n_k)$.

Para estimar la bondad de un nodo sólo se usa h^*. No se usa g^* como en el algoritmo A^*, pues no es posible calcular tal valor único, ya que puede haber muchos caminos para un mismo nodo y además, tampoco es necesario, porque la travesía hacia abajo del mejor camino conocido garantiza que sólo aquellos nodos que estén en el mejor camino serán considerados para la expansión.

Si quisiéramos implementar este algoritmo usando un grafo y/o es necesario realizar tres cosas en cada paso:

- Atravesar el grafo comenzando por el nodo inicial y siguiendo el mejor camino actual, acumulando el conjunto de nodos que van en ese camino y aún no han sido expandidos.
- Seleccionar uno de estos nodos no expandidos y expandirlo. Añadir sus sucesores al grafo y calcular h^* para cada uno de ellos.
- Cambiar la h^* estimada del nodo recientemente expandido para reflejar la nueva información proporcionada por sus sucesores. Propagar este cambio hacia atrás a través del grafo. Para cada nodo que se visita mientras se va avanzando en el grafo, decidir cuál de sus arcos sucesores es más prometedor y marcarlo como parte del mejor camino actual, lo cual cambiará el mejor camino actual.

En [Ric88] y [Bel92] pueden verse los pasos de este algoritmo expresados más detalladamente. [Bra86] contiene una implementación del mismo en PROLOG.

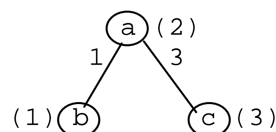
La figura 3.14 muestra un ejemplo de una traza de este algoritmo. Los números entre paréntesis indican el valor de h^* en los nodos y los que no tienen paréntesis, indican los costos de los arcos. Los nodos encerrados entre dos círculos son nodos primitivos.

Diferencias entre el algoritmo A^* y el AO^*

En el algoritmo A^* el camino deseado de un nodo a otro siempre era el de menor costo. Pero éste no es siempre el caso cuando se está buscando en un árbol y/o. Un ejemplo de esto se muestra en la figura 3.15. El nuevo camino al nodo 5 es más largo que yendo a través del nodo 3. Pero el camino por el nodo 3 no lleva a solución, pues necesita también una solución del nodo 4, la cual no existe. Por tanto el camino a través del nodo 10 es mejor.

Paso 1

Paso 2



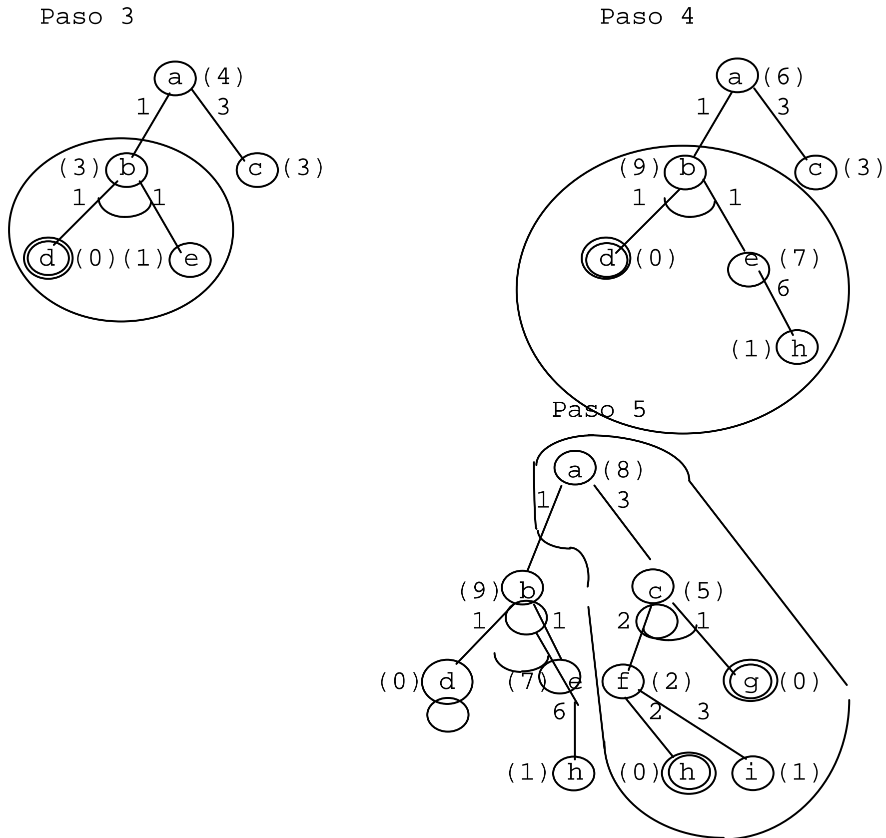


Fig. 3.14. Una traza del algoritmo AO*.

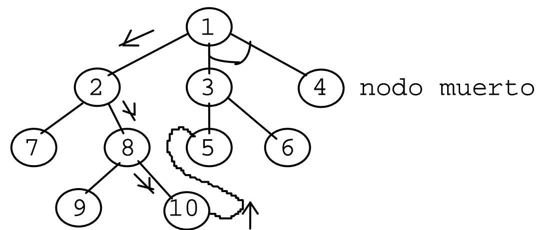


Fig. 3.15. Ejemplo que muestra las diferencias entre A* y AO*.