ra $k \leq 1$, todo vector V es *k-prometedor*. Las soluciones del problema de las 8 reinas se corresponden con aquellos vectores que son 8-prometedores.

Sea N el conjunto de vectores k-prometedores, $0 \leq k \leq 8$. Sea $G = \langle N, A \rangle$ el grafo dirigido tal que $(U, V) \in A$ si y sólo si existe un entero k, con $0 \leq k < 8$, tal que

◊ U es k-prometedor
◊ V es $(k + 1)$-prometedor, y
◊ $U[i] = V[i]$ para todo $i \in [1..k]$

Este grafo es un árbol. Su raíz es el vector vacío correspondiente a $k = 0$. Sus hojas son o bien soluciones ($k = 8$) o posiciones sin salida ($k < 8$) tales como [1, 4, 2, 5, 8]: en tal situación, resulta imposible colocar una reina en la fila siguiente sin amenazar por lo menos a una de las reinas que ya están en el tablero. Las soluciones del problema de las ocho reinas se pueden obtener explorando este árbol. Sin embargo, no generamos explícitamente el árbol para explorarlo después. Más bien, se generan y abandonan los nodos en el transcurso de la exploración. El recorrido en profundidad es el método evidente, sobre todo si sólo necesitamos una solución.

Esta técnica posee dos ventajas con respecto al algoritmo que va probando sistemáticamente todas las permutaciones. En primer lugar, el número de nodos del árbol es menor que 8! = 40.320. Aunque no resulta sencillo calcular teóricamente este número, es fácil contar los nodos empleando una computadora: hay 2.057. De hecho, basta con explorar 114 nodos para obtener una primera solución. En segundo lugar, para decidir si un vector es k-prometedor, sabiendo que es una extensión de un vector $(k-1)$-prometedor, sólo necesitamos comprobar la última reina que haya que añadir. Esto se puede acelerar si asociamos a cada nodo prometedor el conjunto de columnas, el de diagonales positivas (a 45 grados) y el de diagonales negativas (a 135 grados) controlados por las reinas que ya están puestas.

En el procedimiento siguiente, *sol*[1..8] es una matriz global. Para imprimir todas las soluciones del problema de las ocho reinas, se llama a *reinas*(0, Ø, Ø, Ø):

```
procedimiento reinas(k, col, diag45, diag135)
    {sol[1..k] es k-prometedor,
    col = {sol[i] | 1 ≤ i ≤ k},
    diag45 = {sol[i] − i + 1 | 1 ≤ i ≤ k} y
    diag135 = {sol[i] + i − 1 | 1 ≤ i ≤ k} }
  si k = 8 entonces {un vector 8-prometedor es una solución}
                    escribir sol
  sino {explorar las extensiones (k+1)-prometedoras de sol}
    para j ← 1 hasta 8 hacer
      si j ∉ col y j - k ∉ diag45 y j + k ∉ diag135
      entonces sol[k + 1] ← j
        {sol[1..k+1] es (k+1)-prometedor}
        reinas(k+1, col ∪ {j},
          diag45 ∪ {j - k}, diag135 ∪ {j + k})
```


Está claro que el problema se puede generalizar a un número arbitrario de reinas: ¿cómo se pueden situar n reinas en un «tablero» $n \times n$ de tal manera que ninguna de ellas amenace a ninguna de las demás? Tal como cabe esperar, la ventaja obtenida al utilizar la vuelta atrás en lugar de un enfoque exhaustivo se vuelve más pronunciada a medida que crece n. Por ejemplo, para $n = 12$ son 479.001.600 las posibles permutaciones que hay que considerar. Empleando el generador de permutaciones dado anteriormente, la primera solución que se encuentra corresponde a la 4.546.044-ésima situación examinada. Por otra parte, el árbol explorado por el algoritmo de vuelta atrás contiene solamente 856.189 nodos, y se obtiene una solución al visitar el nodo 262. El problema se puede generalizar todavía más colocando las «reinas» en tres dimensiones en un tablero $n \times n \times n$; véase el Problema 9.49.

9.6.3 El caso general

Los algoritmos de vuelta atrás se pueden utilizar aun cuando las soluciones buscadas no tengan todas necesariamente la misma longitud. Véase el esquema general:

```
procedimiento vueltaatrás(v[1..k])
    {v es un vector k-prometedor}
    si v es una solución entonces escribir v
    sino para cada vector (k+1)-prometedor w
                tal que w[1..k] = v[1..k]
                hacer vueltaatrás(w[1..k+1])
```

El **sino** debería estar presente si y sólo si resulta imposible disponer de dos soluciones diferentes tales que una sea prefijo de la otra.

Tanto el problema de la mochila como el problema de las n reinas se resolvían empleando una búsqueda en profundidad en el árbol correspondiente. Algunos problemas que se pueden formular en términos de explorar un grafo implícito tienen la propiedad consistente en que corresponden a un grafo infinito. En este caso puede ser necesario emplear un recorrido en anchura para evitar la exploración interminable de alguna rama infinita carente de resultados positivos. El recorrido en anchura también es apropiado si tenemos que encontrar una solución comenzando a partir de una cierta situación inicial y efectuando el menor número de pasos posible.

9.7 RAMIFICACIÓN Y PODA

Al igual que la vuelta atrás, la ramificación y poda es una técnica para explorar un grafo dirigido implícito. Una vez más, este grafo es normalmente acíclico, o incluso un árbol. Esta vez vamos a buscar la solución óptima de algún problema. En cada nodo, calculamos una cota del posible valor de aquellas soluciones que pudieran encontrarse más adelante en el grafo. Si la cota muestra que cualquiera de estas soluciones tiene que ser necesariamente peor que la mejor solución hallada hasta el momento, entonces no necesitamos seguir explorando esta parte del grafo.

En su versión más sencilla, el cálculo de cotas se combina con un recorrido en anchura o en profundidad, y solamente sirve para podar ciertas ramas de un árbol o para cerrar caminos de un grafo. Con más frecuencia, sin embargo, la cota calculada se utiliza también para seleccionar el camino que, entre los abiertos, parezca más prometedor para explorarlo primero.

En términos generales, podemos decir que las búsquedas en profundidad acaban de explorar los nodos por orden inverso al de su creación, empleando una pila para almacenar los nodos que se han generado pero no han sido examinados aún. Las búsquedas en amplitud terminan de explorar los nodos por el mismo orden en que son creados, utilizando una cola para almacenar los nodos que se han generado pero no han sido examinados aún. Ramificación y poda utiliza cálculos auxiliares para decidir en cada momento qué nodo debe explorarse a continuación y una lista con prioridad para almacenar los nodos que se han generado pero no han sido examinados aún. Recordemos que los montículos suelen ser ideales para almacenar listas con prioridad; véase la Sección 5.7. Ilustraremos la técnica con dos ejemplos.

9.7.1 El problema de la asignación

En el *problema de la asignación*, hay que asignar n tareas a n agentes, de forma que cada agente realice exactamente una tarear. Si al agente i, con $1 \leq i \leq n$, se le asigna la tarea j, con $1 \leq j \leq n$, entonces el coste de realizar esta tarea concreta será c_{ij}. Dada la matriz de costes completa, el problema consiste en asignar tareas a los agentes de tal manera que se minimice el coste total de ejecutar las n tareas.

Por ejemplo, consideremos tres agentes a, b y c, a los que hay que asignar las tareas 1, 2 y 3 con la matriz de costes siguiente:

	1	2	3
a	4	7	3
b	2	6	1
c	3	9	4

Si asignamos la tarea 1 al agente a, la tarea 2 al agente b y la tarea 3 al agente c, entonces nuestro coste total será $4 + 6 + 4 = 14$, mientras que si asignamos la tarea 3 al agente a, la tarea 2 al agente b y la tarea 1 al agente c, el coste será solamente $3 + 6 + 3 = 12$. En este caso concreto, el lector puede verificar que la asignación óptima es $a \rightarrow 2$, $b \rightarrow 3$ y $c \rightarrow 1$, cuyo coste es $7 + 1 + 3 = 11$.

El problema de la asignación tiene numerosas aplicaciones. Por ejemplo, en lugar de hablar de agentes y tareas, podemos formular el problema en términos de edificios y solares, donde c_{ij} es el coste de construir el edificio i en el solar j, y deseamos minimizar el coste total de los edificios. Es fácil inventar otros ejemplos. En general, con n agentes y n tareas, hay $n!$ posibles asignaciones para considerar que son demasiadas incluso para valores moderados de n. Por tanto, recurriremos a la ramificación y poda.

Supongamos que es preciso resolver el caso cuya matriz de costes se muestra en la figura 9.13. Para obtener una cota superior de la respuesta, obsérvese que $a \rightarrow 1$, $b \rightarrow 2$, $c \rightarrow 3$, $d \rightarrow 4$ es una posible solución cuyo coste es $11 + 15 + 19 + 28 = 73$. La solución óptima del problema no puede costar más que esto. Otra posible solución es $a \rightarrow 4$, $b \rightarrow 3$, $c \rightarrow 2$, $d \rightarrow 1$, cuyo coste se obtiene sumando los elementos de la otra diagonal de la matriz de costes, dando $40 + 13 + 17 + 17 = 87$. En este caso, la segunda solución no supone mejora respecto a la primera. Para obtener una cota inferior de la solución, podemos argumentar que sea quien sea el que ejecute la tarea 1, el coste será 11 como mínimo; sea quien sea el que ejecute la tarea 2, el coste será de 12 como mínimo, y así sucesivamente. Por tanto, sumando los elementos más pequeños de cada columna obtenemos una cota inferior de la respuesta. En el ejemplo, esto es $11 + 12 + 13 + 22 = 58$. Se obtiene una segunda cota inferior sumando los elementos más pequeños de cada fila, basándonos en que todos los agentes tienen que hacer algo. En este caso obtenemos $11 + 13 + 11 + 14 = 49$, que es menos útil que la cota inferior anterior. Juntando estos resultados, sabemos que la respuesta de nuestro caso se encuentra en algún lugar de [58..73].

	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Figura 9.13. La matriz de costes para el problema de asignación

Para resolver el problema mediante ramificación y poda, exploraremos un árbol cuyos nodos corresponden a asignaciones parciales. En la raíz del árbol no se han hecho asignaciones. En lo sucesivo, en cada nivel se determina la asignación de un agente más. Para cada nodo, calculamos una cota de las soluciones que se pueden obtener completando la asignación parcial correspondiente, y utilizamos esta cota para cerrar caminos y para guiar la búsqueda. Supongamos por ejemplo que empezando por la raíz decidimos efectuar en primer lugar la asignación del agente a. Dado que hay cuatro formas de hacer esto, hay cuatro ramas que parten de la raíz. La figura 9.14 ilustra la situación.

La cifra que aparece junto a cada nodo es una cota inferior de las soluciones que se pueden obtener completando la asignación parcial correspondiente. Ya hemos mostrado la forma en que se puede obtener la cota 58 en la raíz. Para calcular la cota del nodo $a \rightarrow 1$, por ejemplo, observemos en primer lugar que con esta asignación parcial la tarea 1 tendrá un coste 11. La tarea 2 será ejecutada por b, c o d así

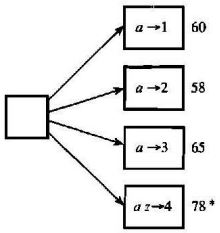

Figura 9.14. Después de asignar una tarea al agente a

que el mínimo coste posible es 14. De forma similar, las tareas 3 y 4 también serán ejecutadas por *b*, *c* o *d*, y sus costes mínimos posibles son 13 y 22 respectivamente. De esta manera, una cota inferior de cualquier solución que se obtenga completando la asignación parcial $a \rightarrow 1$ es 11 + 14 + 13 + 22 = 60. De forma similar, para el nodo $a \rightarrow 2$, la tarea 2 será ejecutada por el agente *a* con un coste 12, mientras que las tareas 1, 3 y 4 serán ejecutadas por los agentes *b*, *c* y *d* con un coste mínimo de 11, 13 y 22 respectivamente. Por tanto, toda solución que incluya la asignación $a \rightarrow 2$ costará como mínimo 12 + 11 + 13 + 22 = 58. Las otras dos cotas inferiores se obtienen de forma similar. Dado que sabemos que la solución óptima no puede sobrepasar 73, ya está claro que no tiene sentido seguir explorando el nodo $a \rightarrow 4$: toda solución obtenida completando esta asignación parcial costará al menos 78, así que no puede ser óptima. El asterisco de este nodo denota que está «muerto». Sin embargo, los otros tres nodos siguen estando vivos. El nodo $a \rightarrow 2$ tiene la cota inferior más pequeña. Argumentando que por tanto parece más prometedor que los otros, será éste el que exploremos a continuación. Hacemos esto fijando un elemento más de la asignación parcial, digamos el *b*. De esta manera llegamos a la situación que se muestra en la figura 9.15.

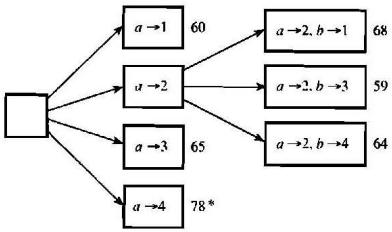

Figura 9.15. Después de asignar una tarea al agente b

Una vez más, la cifra que aparece junto a cada nodo nos da una cota inferior del coste de las soluciones que se pueden obtener completando la asignación parcial

correspondiente. Por ejemplo, en el nodo $a \rightarrow 2, b \rightarrow 1$, la tarea 1 costará 14 y la tarea 2 costará 12. Las tareas restantes 3 y 4 deben ser ejecutadas por *c* o *d*. El coste más pequeño posible para la tarea 3 es por tanto 19, mientras que para la tarea 4 es 23. Por tanto, una cota inferior de las soluciones posibles es 14 + 12 + 19 + 23 = 68. Las otras dos cotas nuevas se calculan de forma similar.

El nodo más prometedor del árbol es ahora $a \rightarrow 2$, $b \rightarrow 3$ con una cota inferior de 59. Para seguir explorando el árbol comenzando en este nodo, fijamos un elemento más de la asignación parcial, digamos el *c*. Cuando se fijan las asignaciones de *a*, *b* y *c*, sin embargo, ya no nos queda opción alguna para la asignación de *d*, así que la solución está completa. Los nodos de la derecha de la figura 9.16, que muestra la siguiente fase de nuestra exploración, corresponden por tanto a soluciones completas.

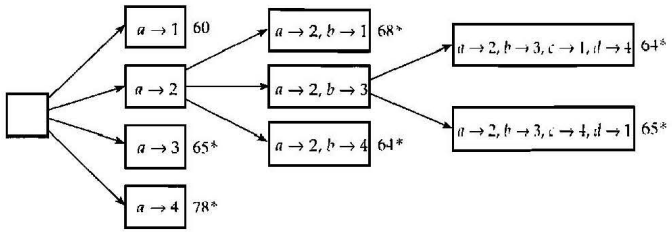

Figura 9.16. Después de asignar una tarea al agente c

La solución $a \rightarrow 2$, $b \rightarrow 3$, $c \rightarrow 1$, $d \rightarrow 4$, con un coste de 64, es mejor que cualquiera de las soluciones halladas al empezar, y nos proporciona una nueva cota superior para el óptimo. Gracias a esta nueva cota superior, podemos descartar los nodos $a \rightarrow 3$ y $a \rightarrow 2$, $b \rightarrow 1$ a efectos de posteriores exploraciones, tal como indican los asteriscos. Ninguna solución que complete estas asignaciones parciales puede ser tan buena como la que acabamos de encontrar. Si solamente deseamos una solución del caso, podemos eliminar también el nodo $a \rightarrow 2$, $b \rightarrow 4$.

El único nodo que sigue mereciendo la pena explorar es $a \rightarrow 1$. Procediendo igual que antes, al cabo de dos pasos obtenemos la situación final que se muestra en la figura 9.17. La mejor solución hallada es $a \rightarrow 1$, $b \rightarrow 3$, $c \rightarrow 4$ y $d \rightarrow 2$, con un coste de 61. En los nodos restantes no explorados, la cota inferior es mayor que 61, así que no tiene sentido seguir estudiándolos. La solución anterior es por tanto la solución óptima de nuestro caso.

El ejemplo ilustra que aun cuando en una fase anterior el nodo $a \rightarrow 2$ era el más prometedor, la solución óptima no llegó a surgir de él. Para obtener nuestra respuesta, hemos construido 15 de los 41 nodos (1 raíz, 4 de profundidad 1, 12 de profundidad 2, y 24 de profundidad 3) que están presentes en un árbol completo del tipo ilustrado. De las 24 soluciones posibles, sólo 6 (incluyendo las dos utilizadas para determinar la cota superior inicial) han sido examinadas.

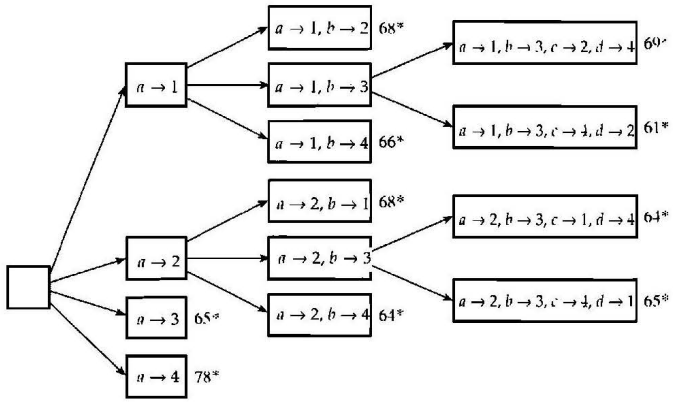

Figura 9.17. El árbol explorado en su totalidad

9.7.2 El problema de la mochila (4)

Como segundo ejemplo, consideremos el problema de la mochila; véanse las Secciones 8.4. y 9.6.1. Aquí se nos pide maximizar $\sum_{i=1}^{n} x_i v_i$ sometido a la restricción $\sum_{i=1}^{n} x_i w_i \leq W$, donde los v_i y w_i son todos estrictamente positivos, y los x_i son enteros no negativos. Este problema también se puede resolver por ramificación y poda.

Supongamos sin pérdida de generalidad que las variables están numeradas de tal manera que $v_i / w_i \geq v_{i+1} / w_{i+1}$. Entonces si los valores de $x_1,x_2,...,x_k, 0 \leq k \leq n$ quedan fijados, con $\sum_{i=1}^{n} x_i w_i \leq W$, es fácil ver que el valor que se puede obtener sumando más elementos de los tipos k+1,...,n a la mochila no puede sobrepasar el valor

$$\sum_{i=1}^{k} x_i v_i + \left(w - \sum_{i=1}^{k} x_i w_i \right) v_{k+1} / w_{k+1}$$

Donde el primer término da el valor de los elementos que ya están en la mochila, y el segundo es una cota del valor que se puede añadir.

Para resolver el problema por ramificación y poda, exploramos un árbol en cuya raíz no está fijado el valor de ninguno de los x_i, y en cada nivel sucesivo se va determinando el valor de una variable más, por orden numérico de variables. En cada nodo que exploremos, sólo generamos aquellos sucesores que satisfagan la restricción de peso, de tal manera que cada nodo tiene un número finito de sucesores. Siempre que se genera un nodo, calculamos una cota superior del valor de la solución que se puede obtener completando la carga parcialmente especificada, y utilizamos estas cotas superiores para cortar ramas inútiles y para guiar la exploración del árbol. Dejamos los detalles para el lector.

9.7.3 Consideraciones generales

La necesidad de mantener una lista de nodos que han sido generados pero que no han sido explorados en su totalidad, situados en diferentes niveles del árbol y preferiblemente ordenados por orden de las cotas correspondientes, hace que la ramificación y poda resulten difíciles de programar. El montículo es una estructura de datos ideal para almacenar esta lista. A diferencia del recorrido en profundidad, y de las técnicas relacionadas, el programador no dispone de una formulación recursiva elegante de la ramificación y poda. Sin embargo, la técnica es tan potente que suele emplearse en aplicaciones prácticas.

Resulta casi imposible dar una idea precisa de lo bien que se va a comportar esta técnica en un problema dado, empleando una cota dada. Siempre hay que llegar a un compromiso en lo concerniente a la calidad de la cota calculada. Con una cota mejor, examinamos menos nodos, y si tenemos suerte se nos guiará a una solución óptima con más rapidez. Por otra parte, lo más probable es que pasemos más tiempo en cada nodo calculando la cota correspondiente. En el caso peor, puede ocurrir que incluso una cota excelente no nos permita cortar ninguna rama del árbol, así que se desperdiciará todo el trabajo adicional efectuado en cada nodo. En la práctica, sin embargo, para problemas del tamaño que suele encontrarse en las aplicaciones, casi siempre es rentable invertir el tiempo necesario para calcular la mejor cota posible (dentro de lo razonable). Por ejemplo, se encuentran aplicaciones tales como programación lineal y que se manejan mediante ramificación y poda, obteniéndose la cota en cada nodo mediante la resolución de un problema asociado de programación lineal, con variables continuas.

9.8 EL PRINCIPIO DE MINIMAX

Sea cual sea la técnica utilizada, persiste el molesto hecho de que para un juego tal como el ajedrez queda descartado un examen exhaustivo del grafo asociado. En esta situación, tenemos que conformarnos con una búsqueda parcial en torno a la situación actual. Éste es el principio que subyace a una importante heurística denominada *minimax*. Aunque esta heurística no nos permite asegurar que ganaremos, siempre que esto sea posible, halla una jugada de la cual puede esperarse razonablemente que se cuente entre las mejores jugadas disponibles, explorando solamente una parte del grafo a partir de alguna posición dada. La exploración del grafo se detiene normalmente antes de alcanzar las posiciones terminales, empleando uno entre varios criterios posibles, y las posiciones en que se detiene la exploración se evalúan de forma heurística. En cierto sentido, esto no es sino una versión sistemática del método utilizado por ciertos jugadores humanos, que consiste en examinar anticipadamente un pequeño número de jugadas. Nos limitaremos a esbozar la técnica.