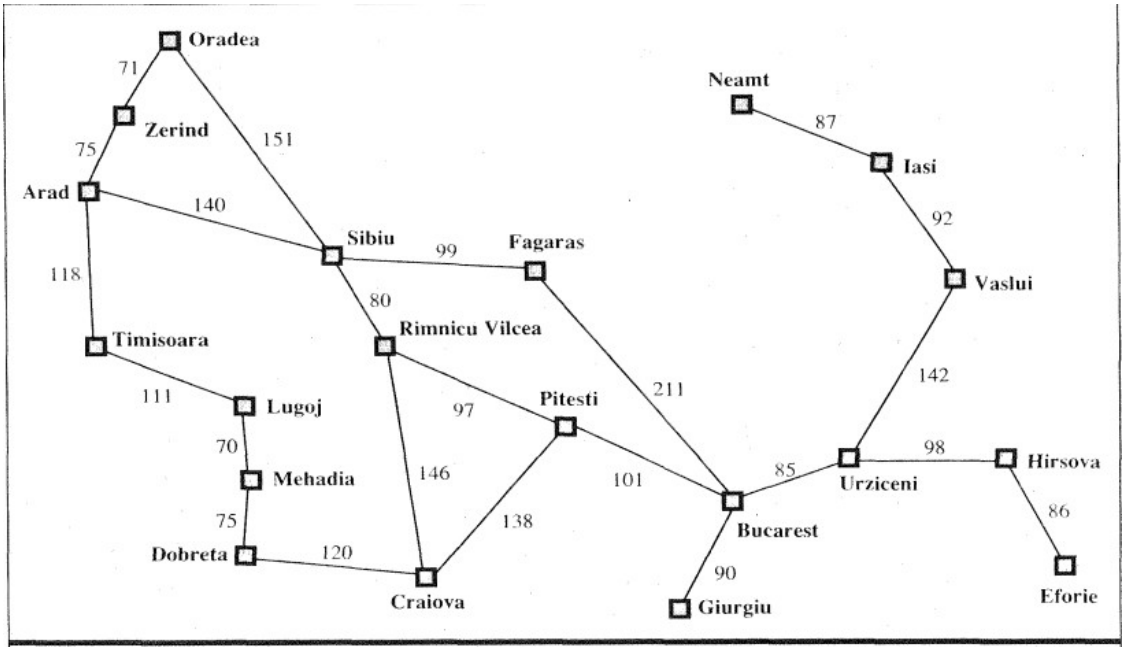

Figura 3.2 Un mapa de carreteras simplificado de parte de Rumanía.

Estrategias de búsqueda informada (heurísticas)

Búsqueda primero el mejor

La búsqueda primero el mejor es un caso particular del algoritmo general de búsqueda sobre árboles, en el cual se selecciona un nodo para la expansión basada en una función de evaluación, f(n). El nombre de búsqueda primero el mejor es un tanto inexacto. A fin de cuentas, si nosotros realmente pudiéramos expandir primero al mejor nodo, esto no sería una búsqueda en absoluto, sería una marcha directa al objetivo. Todo lo que podemos hacer es escoger el nodo que parece ser el mejor según la función de evaluación. Si la función de evaluación es exacta, entonces de verdad sería el mejor nodo, en realidad la función de evaluación no será así, y puede dirigir la búsqueda por un mal camino.

Hay una familia entera de algoritmos de búsqueda primero el mejor con funciones de evaluación diferentes. Un componente clave de estos algoritmos es una función heurística, denotada h(n).

Por ejemplo, en el grafo de Rumanía, podríamos estimar el costo del camino más barato desde Arad a Bucarest con la distancia en linea recta desde Arad a Bucarest.

Búsqueda voraz primero el mejor

Esta búsqueda trata de expandir el nodo más cercano al objetivo, suponiendo que probablemente conduzca rápidamente a una solución. Así, evalúa los nodos utilizando solamente la función heurística h(n).

Veamos un ejemplo, para el problema de encontrar una ruta en Rumanía utilizando la heurística **distancia en linea recta** (h_{DLR}). Si el objetico es Bucarest, tendremos que conocer la distancia en línea recta desde todas las ciudades hacia Bucarest, Figura4.1.

Entonces, la herurística para Arad sería h_{DLR}(Arad) = 366. Notese que los valores de h_{DLR} no pueden calcularse de la descripción del problema en si mismo. Además, debemos tener una cierta experiencia para decidir que heurística aplicar, en este caso saber que la distancia en línea recta está correlacionada con las distancias del camino y es, por lo tanto, una heurística útil.

Arad	366	**Mehadia**	241
Bucarest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Figura 4.1 Valores de h_{DLR}. Distancias en línea recta a Bucarest.

En la figura 4.2 se muestra el progreso de una búsqueda voraz primero el mejor con h_{DLR} para encontrar un camino desde Arad a Bucarest. El primer nodo a expandir será Sibiu, porque es el más cercano a Bucarest. El siguiente nodo a expandir será Fagaras, porque es el más cerca. Fagaras en su turno genera a Bucarest, y el siguiente nodo a explorar será Bucarest que tiene un h_{DLR} de cero. Como puede darse cuenta, la solución no es óptima, el camino encontrado es 32 km más largo que el camino que pasa por Rimnicu Vilcea y Pitesti.

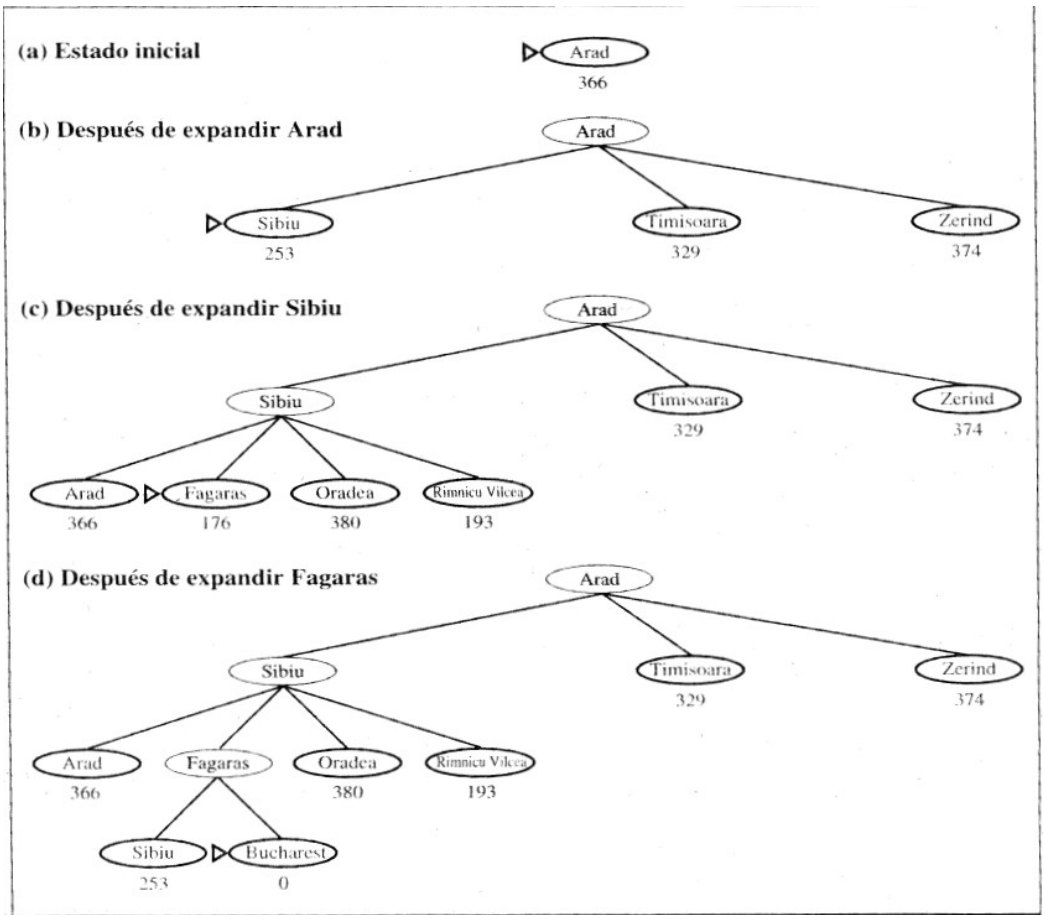

Figura 4.2 Etapas en una búsqueda primero el mejor avara para Bucarest utilizando la heurística distancia en línea recta h_{DLR}. Etiquetamos lo nodos con sus h-valores.

Minimizar h(n) nos puede dar ventajas falsas. Considere el problema de ir de Iasi a Fagaras. La heurística sugiere que Neamt sea expandido primero, porque es la más cercana a Fagaras, pero esto es un callejón sin salida. La solución es ir primero a Vaslui (un paso que en realidad está más lejano del objetivo según la heruística) y luego a Urziceni, Bucarest y Fagaras. En este caso, entonces, la heurística provoca nodos innecesarios para expandir. Además, si no somos cuidadosos en descubrir estados repetidos (visitados), la solución nunca se encontrará, la búsqueda oscilará entre Neamt e Iasi.

La búsqueda voraz primero el mejor se parece a la búsqueda primero en profundidad en el modo que prefiere seguir un camino hacia el objetivo, pero volverá atrás cuando llegue a un callejón sin salida. Sufre los mismos defectos que la búsqueda primero en profundidad, no es óptima, y es incompleta.

Búsqueda A^*: minimizar el costo estimado total de la solución

A la forma más ampliamente conocida de la búsqueda primero el mejor se le llama búsqueda $\mathbf{A^*}$ (pronunciada búsqueda A estrella). Evaluá los nodos combinando g(n), el costo de ir del nodo raíz al nodo n, y h(n), el costo estimado de ir de n al nodo objetivo.

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

f(n) = costo más barato estimado de la solución a través de n

Así, si tratamos de encontrar la solución más barata, es razonable intentar primero el nodo con el valor más bajo de g(n)+h(n). Resulta que esta estrategia es más que razonable, siempre y cuando la heurística h(n) sea admisible, es decir, que no sobrestime el costo de alcanzar el objetivo. La búsqueda $\mathbf{A^*}$ es tanto completa como óptima.

Regresemos al ejemplo de ir de Arad a Bucarest. La distancia en linea recta es admisible porque el camino más corto entro dos puntos cualquiera es una línea recta, entonces la línea recta no puede ser una sobrestimación. En la figura 4.3, mostramos el progreso de un árbol de búsqueda $\mathbf{A^*}$ de Arad a Bucarest. Los valores de g(n) se calculan directamente de la Figura 3.2, y los valores de h(n) son los de la figura 4.1. Notemos en particular que Bucarest aparece primero en la frontera del paso e, pero no se selecciona para expansión porque su costo de 450 es más alto que el de Pitesti de 417.

Para la búsqueda $\mathbf{A^*}$, la mayor desventaja es la cantidad de nodos que mantiene en memoria, por lo general se queda sin espacio antes de que el tiempo sea un problema. Por esta razón $\mathbf{A^*}$ no es práctico para problemas grandes.

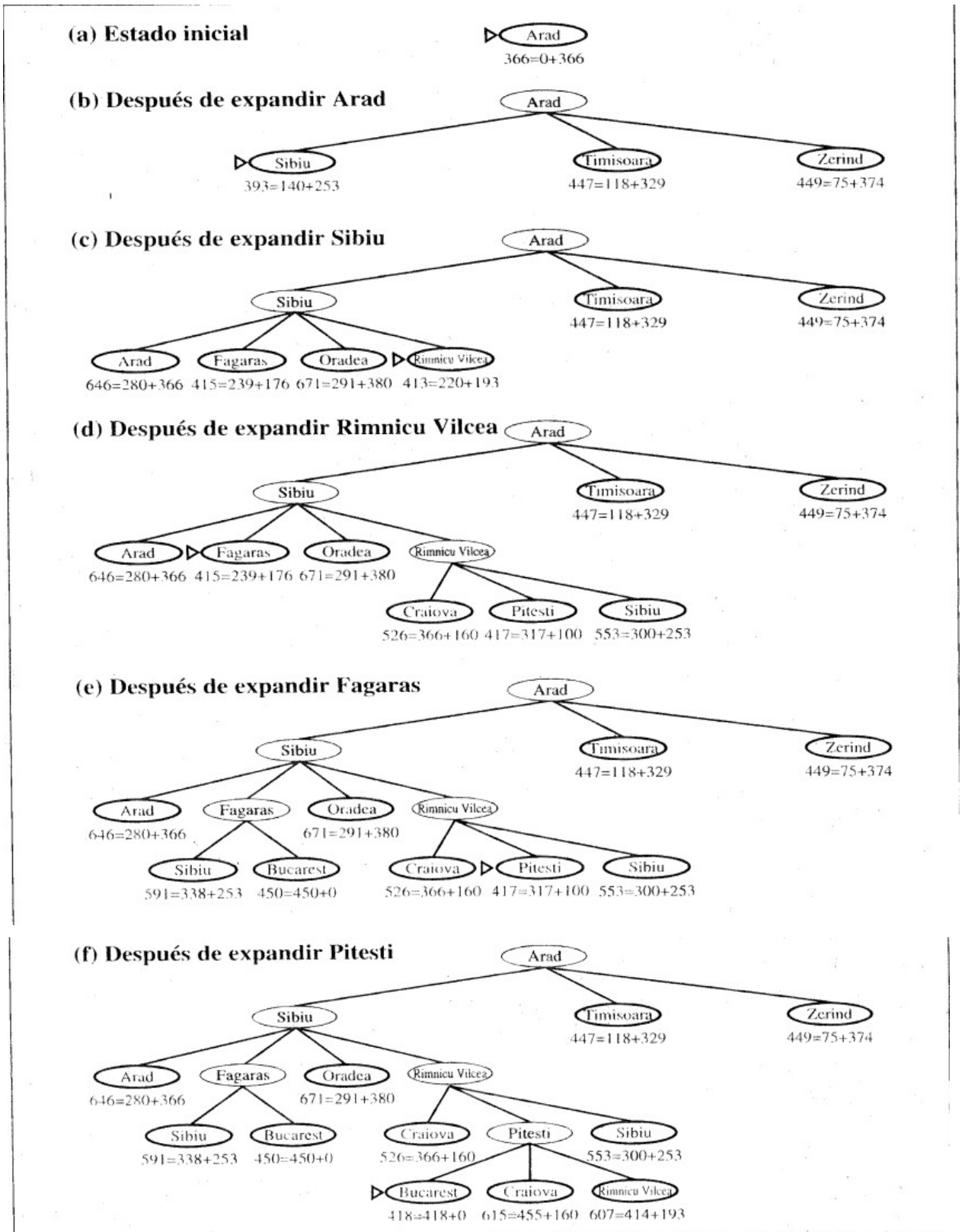

Figura 4.3 Etapas en una búsqueda A* para Bucarest. Etiquetamos lo nodos con $f = g + h$. Los valores h son las distancias en línea recta a Bucarest tomadas de la Figura 4.1.

Búsqueda recursiva primero del primero mejor (BRPM)

Es un algoritmo sencillo recursivo que intenta imitar la operación de la búsqueda primero el mejor estándar, pero utilizando sólo un espacio lineal. EL algoritmo explora a profundidad de manera recursiva, mantiene el f-valor del mejor camino alternativo disponible desde cualquier antepasado al nodo actual. Si el nodo actual excede este límite, la recursividad vuelve atrás, la BRPM sustituye los f-valores de cada nodo a lo largo del camino con el mejor f-valor de su hijo. De este modo, la BRPM recuerda el f-valor de la mejor hoja en el subárbol olvidado y por lo tanto puede decidir si merece la pena expandir el subárbol más tarde. La figura 4.6 muestra cómo la BRPM alcanza Bucarest.

En el ejemplo de la Figura 4.6, la BRPM sigue primero el camino vía Rimnicu Vilcea, entonces "cambia de opinión" e intenta Fagaras, y luego cambia de opinión hacia atrás otra vez. Estos cambios de opinión ocurren porque cada vez que el mejor camino actual se extiende, hay una buena posibilidad de que aumente su f-valor. Cuando esto pasa, en particular en espacios de búsqueda grandes, el segundo mejor camino podría convertirse en espacios de búsqueda grandes, el segundo mejor camino podría vertirse en el mejor camino, entonces la búsqueda tiene que retroceder para seguirlo. Se podrían requerir muchos cambios de opinión, y pot lo tanto, muchas nuevas expansiones de nodos para crear el mejor camino.

```
función BÚSQUEDA-RECURSIVA-PRIMERO-MEJOR(problema) devuelve una solución, o fallo
  BRPM(problema, HACER-NODO(ESTADO-INICIAL[problema]),∞)

función BRPM(problema,nodo,f_límite) devuelve una solución, o fallo y un nuevo límite
f-costo
  si TEST-OBJETIVO[problema](estado) entonces devolver nodo
  sucesores ← EXPANDIR(nodo,problema)
  si sucesores está vacío entonces devolver fallo, ∞
  para cada s en sucesores hacer
    f[s] ← max(g(s) + h(s), f[nodo])
  repetir
    mejor ← nodo con f-valor más pequeño de sucesores
    si f[mejor] > f_límite entonces devolver fallo, f[mejor]
    alternativa ← nodo con el segundo f-valor más pequeño entre los sucesores
    resultado,f[mejor] ← BRPM(problema,mejor,min(f_límite,alternativa))
    si resultado ≠ fallo entonces devolver resultado
```

Figura 4.5 Algoritmo para la búsqueda primero el mejor recursiva.

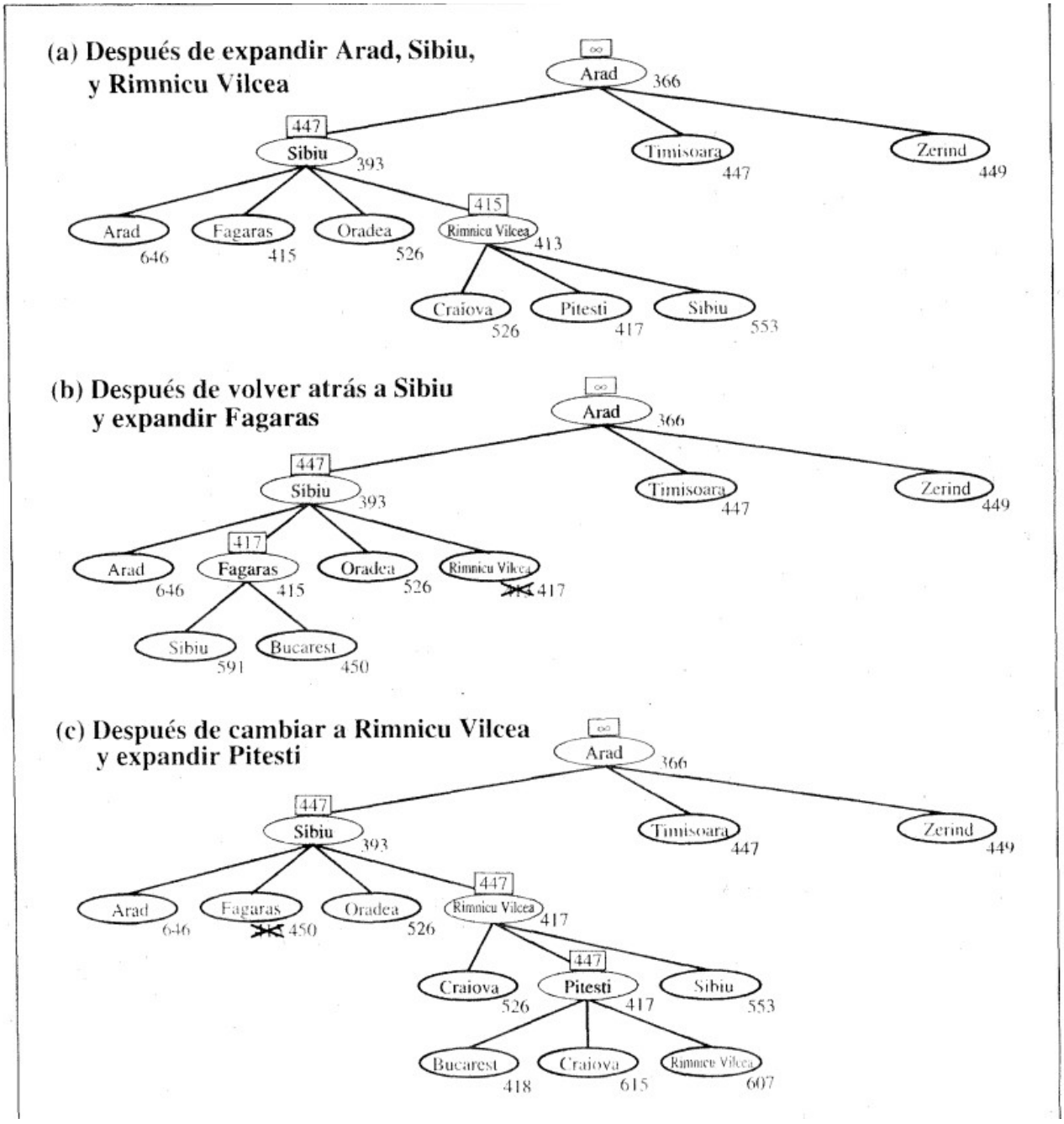

Figura 4.6 Etapas en una búsqueda BRPM para la ruta más corta a Bucarest. Se muestra el valor del *f*-límite para cada llamada recursiva sobre cada nodo actual. (a) Se sigue el camino vía Rimnicu Vilcea hasta que la mejor hoja actual (Pitesti) tenga un valor que es peor que el mejor camino alternativo (Fagaras). (b) La recursividad se aplica y el mejor valor de las hojas del subárbol olvidado (417) se le devuelve hacia atrás a Rimnicu Vilcea; entonces se expande Fagaras, revela un mejor valor de hoja de 450. (c) La recursividad se aplica y el mejor valor de las hojas del subárbol olvidado (450) se le devuelve hacia atrás a Fagaras; entonces se expande Rimnicu Vilcea. Esta vez, debido a que el mejor camino alternativo (por Timisoara) cuesta por lo menos 447, la expansión sigue por Bucarest.

Funciones heurísticas

El 8-puzle fue uno de los primeros problemas de búsqueda heurística. El objetivo es deslizar las fichas horizontalmente o verticalmente al espacio vacío hasta que la configuración empareje con la configuración objetivo.

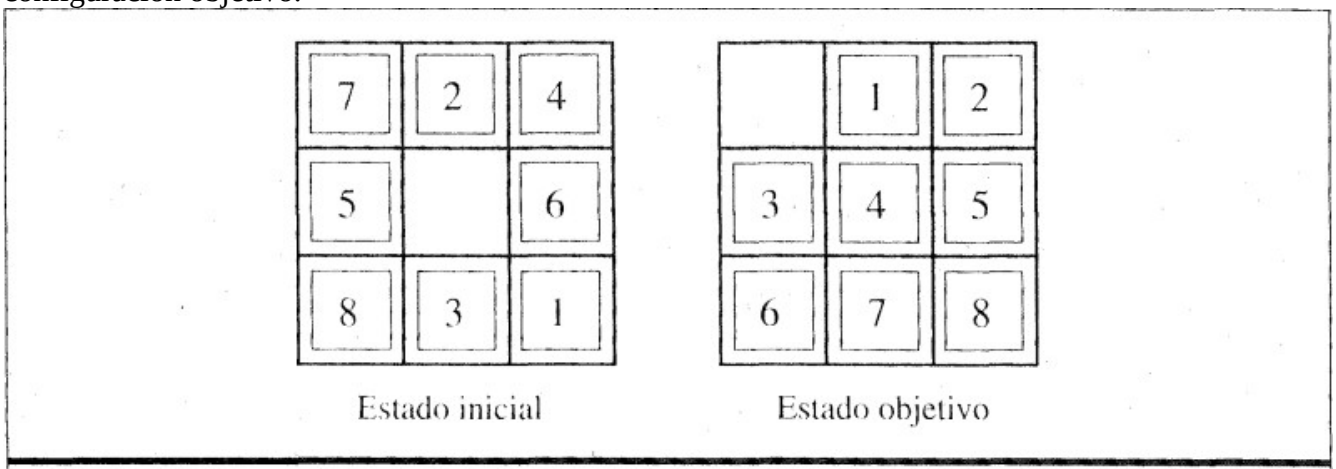

Figura 4.7 Un caso típico del 8-puzle. La solución tiene 26 pasos.

El costo promedio de la solución para los casos generados al azar del 8-puzle son aproximadamente 22 pasos. El factor de ramificación es aproximadamente 3. Esto quiere decir que una búsqueda exhaustiva a profundidad 22 miraría sobre 3^{22} estados. Si tenemos cuidado de los estados repetidos podemos reducir este número a aproximadamente 170000 estados. Este número es manejable, sin embargo el número correspondiente para el 15-puzle es aproximadamente 10^{13}, entonces lo siguiente es encontrar una buena función heurística. Si queremos encontrar soluciones más cortas utilizando **A***, necesitamos una función heurística que nunca sobrestima el número de pasos al objetivo. Hay una larga historia de tales heurísticas para el 15-puzle, aquí están dos de las más comunes:

- h_1 = número de piezas mal colocadas, también llamada distancia de Hamming. Para la figura 4.7, las 8 piezas están fuera de su posición, así que el espacio inicial tiene $h_1 = 8$. h_1 es admisible, porque está claro que cualquier pieza que está fuera de su lugar debe moverse por lo menos una vez.
- H2 = suma de las distancias de las piezas a sus posiciones en el objetivo. Como las piezas no pueden moverse en diagonal, la distancia que contaremos será la suma de las distancias horizontales y verticales. Esto se llama distancia en la ciudad o distancia Manhattan. H2 es también admisible, porque cualquier movimiento que se pueda hacer es mover una pieza un paso más cerca del objetivo. Las piezas 1 a 8 en el estado inicial de la Figura 4.7 nos da una distancia de Manhattan de:

$$h2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$$

Como es deseado, ninguna de las heurísticas sobrestima el costo de la solución verdadera que es 26.