



RESOLUCIÓN DE PROBLEMAS

- GENERALIDADES
- ESPACIO DE ESTADOS
- CARACTERÍSTICAS GENERALES DE LOS PROCESOS DE BÚSQUEDA
- ESTRATEGIAS DE EXPLORACIÓN DEL ESPACIO DE ESTADOS
- CONCLUSIONES



RESOLUCIÓN DE PROBLEMAS

- Moret et al., Fundamentos de Inteligencia Artificial, Servicio de Publicaciones UDC, 2004
- Nilsson, Problem-Solving Methods in Artificial Intelligence, McGraw-Hill, eds., 1971
- Rich & Knight, Inteligencia Artificial, McGraw-Hill, eds., 1994
- Russell & Norvig, Inteligencia Artificial: Un Enfoque Moderno, Pearson Prentice Hall, eds., 2004



RESOLUCIÓN DE PROBLEMAS

- El comportamiento inteligente obliga a utilizar de forma eficaz y eficiente un conjunto mínimo de conocimientos... ¿por qué?
- Distintos problemas requieren distintas técnicas:
 - ¿Se requiere inteligencia para resolver: $Ax^2 + Bx + C = 0$?
- La IA debe producir programas que:
 - Capten generalizaciones
 - Incluyan conocimiento explícito
 - Sean fácilmente actualizables
 - Puedan ser aplicados en muchas situaciones diferentes



RESOLUCIÓN DE PROBLEMAS

- En IA el tipo de técnica está condicionado por el dominio de aplicación
- Normalmente, para resolver un problema real, no hay planteamientos exclusivos

Programa de IA	Programa Convencional
Dominios Simbólicos	Dominios Numéricos
Procesos Heurísticos	Procesos Algorítmicos
Pasos Implícitos	Pasos Explícitos
Información y Control Separados	Información y Control Integrados



RESOLUCIÓN DE PROBLEMAS

- El problema de los dos cubos:
 - Disponemos de dos cubos inicialmente vacíos, uno de 8 litros, y el otro de 6 litros. Ninguno de los cubos tiene marcas ni divisiones. Disponemos también de un grifo que puede emplearse para llenar los cubos. ¿Qué tenemos que hacer para llenar el cubo de 8 litros justamente hasta la mitad?
 - NOTA: Llamaremos A al cubo de 8 litros, y B al cubo de 6 litros



RESOLUCIÓN DE PROBLEMAS

Número de orden	Acción	Resultado
1	Llenar B	A vacío, 6 l en B
2	Vaciar B en A	6 l en A, B vacío
3	Llenar B	6 l en A, 6 l en B
4	Llenar A con B	8 l en A, 4 l en B
5	Vaciar A	A vacío, 4 l en B
6	Vaciar B en A	4 l en A, B vacío



RESOLUCIÓN DE PROBLEMAS

- ¿Es la única solución?
 - Si → Fin
 - No → Encontrar otra solución
- ¿La nueva solución encontrada es equivalente a la anterior?
- ¿Cuál es la mejor solución?
- ¿De qué depende que una solución sea mejor que otra?
- ¿Cuáles son los requisitos mínimos que debe cumplir una solución para ser aceptable?



RESOLUCIÓN DE PROBLEMAS

- Una representación del problema

- A = Cubo de 8 litros
- B = Cubo de 6 litros
- [A] = Contenido de A
- [B] = Contenido de B
- ([A],[B]) = Estado actual del problema
- Número "n" = Acción ejecutada



RESOLUCIÓN DE PROBLEMAS

Número de orden	Precondiciones	Acción
1	A no lleno	Llenar A
2	B no lleno	Llenar B
3	A no vacío	Vaciar A
4	B no vacío	Vaciar B
5	A no vacío, y B no lleno, y $[A] + [B] \leq 6$	Vaciar A en B
6	B no vacío, y A no lleno, y $[A] + [B] \leq 8$	Vaciar B en A
7	B no vacío, y A no lleno, y $[8] + [6] \geq 8$	Llenar A con B
8	A no vacío, y B no lleno, y $[8] + [6] \geq 6$	Llenar B con A



RESOLUCIÓN DE PROBLEMAS

- Secuencia de acciones que nos lleva a una solución del problema:

(0,0) 2 (0,6) 6 (6,0) 2 (6,6) 7 (8,4) 3 (0,4) 6 (4,0)

- Espacio de Estados (EE) = Descripción formal del universo de discurso
 - Conjunto de estados iniciales
 - Conjunto de operadores que definen operaciones permitidas entre estados
 - Conjunto de estados meta u objetivos: soluciones aceptables, aunque no necesariamente la mejor



RESOLUCIÓN DE PROBLEMAS

- La resolución de un problema en IA consiste en:
 1. La aplicación de un conjunto de técnicas conocidas, cada una de ellas definida como un paso simple en el espacio de estados
 2. Un proceso de búsqueda, o estrategia general de exploración del espacio de estados

Para abordar un problema desde la óptica de la inteligencia artificial tenemos que ser capaces de construir un modelo computacional del universo de discurso, o dominio del problema



RESOLUCIÓN DE PROBLEMAS

- FORMALMENTE...
 - **I** es el conjunto de estados iniciales...
 $I = \{i1, i2, \dots, in\}$
 - **O** es el conjunto de operaciones permitidas...
 $O = \{o1, o2, \dots, om\}$
 - **M** es el conjunto de metas o soluciones aceptables...
 $M = \{m1, m2, \dots, mt\}$
 - **Búsqueda** = proceso de exploración en el espacio de estados tal que...
 $O : (I \rightarrow M)$
 - **Paso Simple**... $ox : (iz \rightarrow iw)$ con $iz, iw \in I, ox \in O$
 - Si $iw \in M$, entonces iw verifica la prueba de meta, o test de realización, y es una solución del problema



RESOLUCIÓN DE PROBLEMAS

- El espacio de estados es el soporte físico del dominio de discurso
- Necesitamos también procesos de búsqueda o mecanismos generales de exploración del espacio de estados
 - Criterios de selección y aplicación de operadores relevantes
 - Capacidad de decisión sobre cuál será el próximo movimiento
 - Búsqueda sistemática
 - Tratar de generar siempre estados nuevos, no generados previamente



RESOLUCIÓN DE PROBLEMAS

- Volviendo al problema de los dos cubos...
 - I = (0,0)
 - M = (4,x) / $x \in [0,6]$
 - O = {op1, op2, op3, op4, op5, op6, op7, op8}
- Estrategia 1
 1. *Identificar estado inicial*
 2. *Prueba de meta*
 - Si meta → Fin
 - Si no meta → Seguir (ir a 3)
 3. *Aplicar 1er operador que cumpla requisitos de estado actual, según su número de orden*
 4. *Prueba de meta*
 - Si meta → Fin
 - Si no meta → Seguir (ir a 3)
- Secuencia de estados generada: (0,0)1(8,0)2(8,6)3(0,6)1(8,6)...
- Hemos caído en un bucle
- La estrategia es sistemática, pero no siempre genera estados nuevos, y hay operadores que se repiten



RESOLUCIÓN DE PROBLEMAS

-Estrategia 2

1. *Identificar estado inicial*
2. *Prueba de meta*
 - *Si meta* → *Fin*
 - *Si no meta* → *Ir a 3*
3. *Seleccionar operadores que verifiquen precondiciones*
4. *Descartar los ya aplicados*
5. *Aplicar el primero de los supervivientes*
6. *Prueba de meta*
 - *Si meta* → *Fin*
 - *Si no meta* → *Ir a 3*

-Secuencia de estados: (0,0)1(8,0)2(8,6)3(0,6)4(0,0)

-Búsqueda detenida sin solución

op1 a op4 ya aplicados, op5 a op8 no verifican restricciones

-Necesidad de utilizar estructuras adicionales (autoconocimiento)



RESOLUCIÓN DE PROBLEMAS

-Estrategia 3

1. *Identificar estado inicial*
2. *Prueba de meta*
 - *Si meta* → *Fin*
 - *Si no meta* → *Ir a 3*
3. *Seleccionar operadores que verifiquen precondiciones*
4. *Descartar los ya aplicados*
5. *Descartar operadores que no generen estados nuevos*
6. *Aplicar el primero de los supervivientes*
7. *Prueba de meta*
 - *Si meta* → *Fin*
 - *Si no meta* → *Ir a 3*

-Secuencia de estados: (0,0)1(8,0)2(8,6)3(0,6)6(6,0)

-Búsqueda detenida sin solución

op1, op2, op3, op6 ya se han aplicado; op4, op7 y op8 no verifican restricciones, op5 genera estado repetido.

-Más estructuras adicionales (incrementar el autoconocimiento)



RESOLUCIÓN DE PROBLEMAS

-Estrategia 4

1. *Identificar estado inicial*
2. *Prueba de meta*
 - *Si meta* → *Fin*
 - *Si no meta* → *Ir a 3*
3. *Seleccionar operadores que verifiquen precondiciones*
4. *Descartar operadores que no generen estados nuevos*
5. *Aplicar el primero de los supervivientes*
6. *Prueba de meta*
 - *Si meta* → *Fin*
 - *Si no meta* → *Ir a 3*

-Secuencia de estados...

(0,0)1(8,0)2(8,6)3(0,6)6(6,0)2(6,6)7(8,4)3(0,4)6(4,0)

-Solución aceptable, aunque no óptima

-Más estructuras adicionales (incrementar el autoconocimiento)

Estrategias de resolución de problemas. Op. Internas (I)



Estado	Op	Aplicable	Motivos	NuevoEst.	Solución
(0,0)	1	SI	verifica restricciones	(8,0)	NO
(8,0)	1	NO	no verifica restricciones		
(8,0)	2	SI	verifica restricciones	(8,6)	NO
(8,6)	1	NO	no verifica restricciones		
(8,6)	2	NO	no verifica restricciones		
(8,6)	3	SI	verifica restricciones	(0,6)	NO
(0,6)	1	NO	verifica restricciones, pero estado ya generado		
(0,6)	2	NO	no verifica restricciones		
(0,6)	3	NO	no verifica restricciones		
(0,6)	4	NO	verifica restricciones, pero estado ya generado		
(0,6)	5	NO	no verifica restricciones		
(0,6)	6	SI	verifica restricciones	(6,0)	NO
(6,0)	1	NO	verifica restricciones, pero estado ya generado		
(6,0)	2	SI	verifica restricciones	(6,6)	NO

Estrategias de resolución de problemas. Op. Internas (II)



(6,6)	1	NO	verifica restricciones, pero estado ya generado		
(6,6)	2	NO	no verifica restricciones		
(6,6)	3	NO	verifica restricciones, pero estado ya generado		
(6,6)	4	NO	verifica restricciones, pero estado ya generado		
(6,6)	5	NO	no verifica restricciones		
(6,6)	6	NO	no verifica restricciones		
(6,6)	7	SI	verifica restricciones	(8,4)	NO
(8,4)	1	NO	no verifica restricciones		
(8,4)	2	NO	verifica restricciones, pero estado ya generado		
(8,4)	3	SI	verifica restricciones	(0,4)	NO
(0,4)	1	NO	verifica restricciones, pero estado ya generado		
(0,4)	2	NO	verifica restricciones, pero estado ya generado		
(0,4)	3	NO	no verifica restricciones		
(0,4)	4	NO	verifica restricciones, pero estado ya generado		
(0,4)	5	NO	no verifica restricciones		
(0,4)	6	SI	verificarestricciones	(4,0)	SI



RESOLUCIÓN DE PROBLEMAS

Cuestiones importantes:

- Hemos definido de forma independiente el conocimiento (operadores y precondiciones) de la forma (estrategias) en que dicho conocimiento es empleado.
 - Estrategias de control de conocimiento reutilizables.
- Hay una separación clara entre el conocimiento del sistema y las estrategias de control del conocimiento
- Son soluciones aceptables aquéllas que cumplen las restricciones de la prueba de meta (¿qué pasaría si del grifo manara Hg?)
- El sistema necesita AUTOCONOCIMIENTO (en forma de estructuras adicionales auxiliares), para funcionar correctamente



RESOLUCIÓN DE PROBLEMAS

- CARACTERÍSTICAS GENERALES DE LOS PROCESOS DE BÚSQUEDA
 - 1. DIRECCIÓN DE LA BÚSQUEDA
 - 2. TOPOLOGÍA DEL PROCESO
 - 3. REPRESENTACIÓN DE LOS ESTADOS POR LOS QUE DISCURRE LA SOLUCIÓN DEL PROBLEMA
 - 4. CRITERIOS DE SELECCIÓN Y APLICACIÓN DE OPERADORES RELEVANTES
 - 5. OPTIMIZACIÓN DE LA BÚSQUEDA MEDIANTE EL USO DE FUNCIONES HEURÍSTICAS



RESOLUCIÓN DE PROBLEMAS

- DIRECCIÓN DEL PROCESO DE BÚSQUEDA
 - Desde los estados iniciales hacia los estados meta
 - Proceso progresivo o dirigido por los datos
 - Desde los estados meta hacia los estados iniciales
 - Proceso regresivo, evocativo, o dirigido por objetivos



RESOLUCIÓN DE PROBLEMAS

- ¿Cuándo, y en función de qué, es mejor una dirección que otra?
 1. Tamaño relativo de los conjuntos I y M del espacio de estados
 2. Factor de ramificación, o número promedio de estados que podemos alcanzar directamente desde un estado dado
 3. Inclusión de estructuras explicativas como requisito inicial en el diseño de nuestro sistema inteligente
 1. De menor a mayor número de estados
 2. En el sentido del menor factor de ramificación
 3. De la forma que mejor se ajuste al modo de razonar de expertos y usuarios



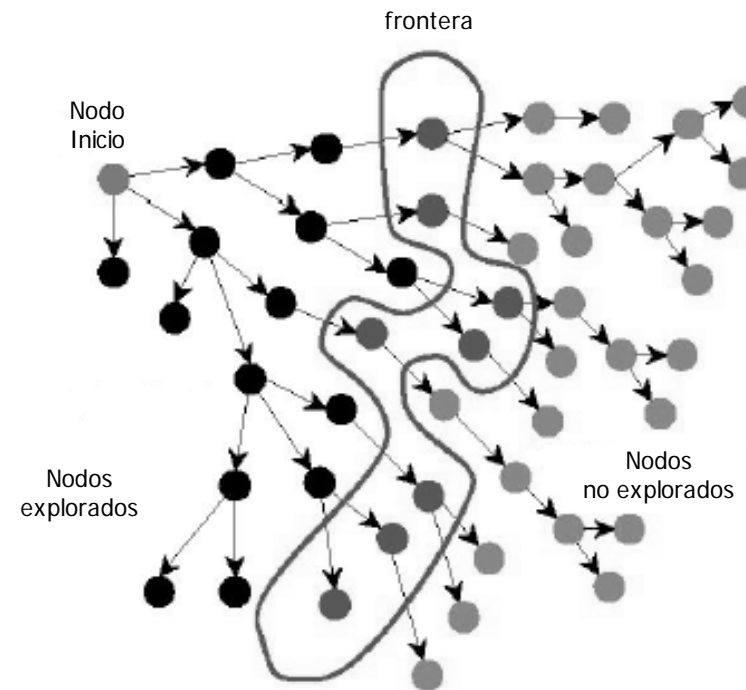
RESOLUCIÓN DE PROBLEMAS

- TOPOLOGÍA DEL PROCESO DE BÚSQUEDA
 - ÁRBOL
 - GRAFO

Topología del proceso de búsqueda



- Construcción de un árbol de búsqueda superpuesto al espacio de estados
 - Raíz (estado inicial), hojas (estados sin sucesor)
 - En cada paso seleccionar una hoja y expandir



Topología del proceso de búsqueda



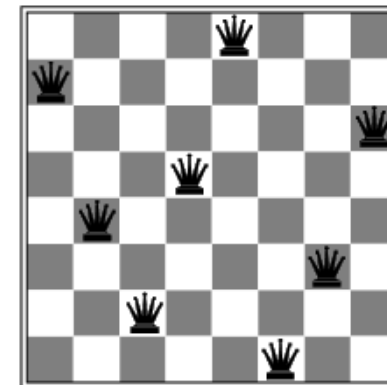
```
funcion BUSQUEDA –GENERAL (problema, estrategia) produce solucion o fallo
  inicializa el arbol de busqueda empleando el estado inicial del problema
  bucle hacer
    si no hay candidatos para la expansion devolver fallo
    escoger un nodo hoja para hacer la expansion, de acuerdo a la estrategia
    si el nodo hoja contiene un estado meta, devolver la solucion
    sino expandir el nodo y anadir los nodos resultantes al arbol de
    busqueda
  fin
```

Topología del proceso de búsqueda



- El espacio de estados (finito) no es igual al árbol de búsqueda (infinito?)
- ¿Árbol o Grafo? La formulación eficiente del problema evita repetir estados. Ejemplo: El problema de las 8 reinas con formulación incremental:
 - Opción 1:
 - Estados: cualquier combinación de 0 a 8 reinas en el tablero
 - Estado inicial: ninguna reina
 - Operador: Añadir una reina en una casilla libre
 - N° combinaciones a investigar = $64 * 63 * \dots * 57 \approx 3 * 10^{14}$
 - Opción 2:
 - Estado: 0 a 8 reinas, una por columna desde la columna más a la izquierda, sin atacarse
 - Operador: Añadir una reina en cualquier casilla en la columna más a la izquierda vacía, tal que no se vea atacada
 - N° combinaciones a investigar $\Rightarrow 2057$. Cada estado se alcanza por un único camino.

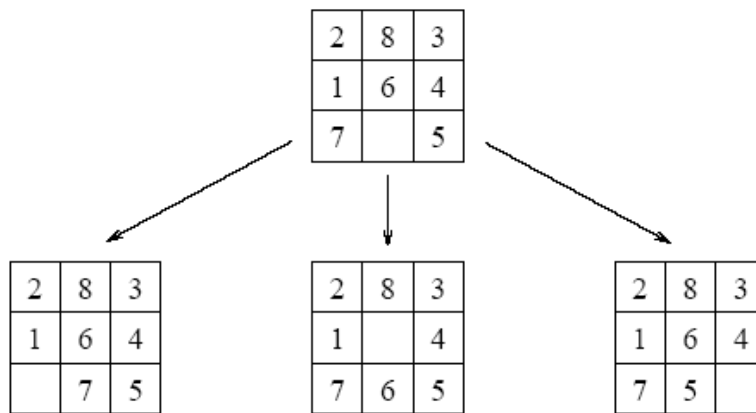
	Árbol	Grafo
Memoria	↑	↓
Eficiencia	↑	↓



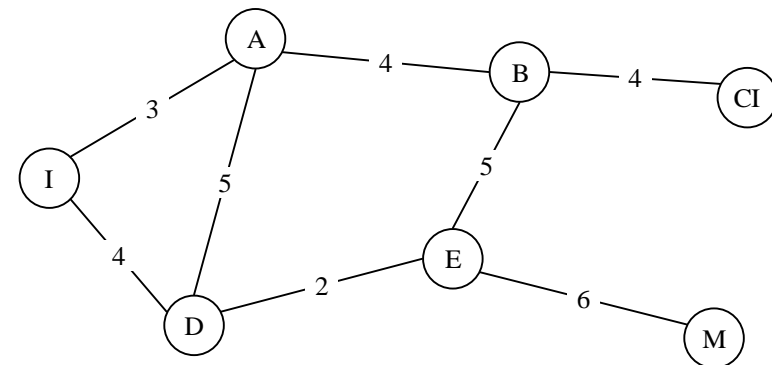
Topología del proceso de búsqueda



- Cuando existen operadores reversibles, la repetición es inevitable



Problema del 8 puzzle



Problema del viajante de comercio

Topología del proceso de búsqueda



- Transformación de árbol en grafo
 1. Empezar generando estado(s) tras aplicar operador(es) relevante(s)
 2. Para cada nuevo estado generado
 - Si es nuevo añadirlo y volver a (1)
 - Si no es nuevo descartarlo e ir a (3)
 3. Añadir un nuevo enlace entre el nodo que se está expandiendo y el sucesor
 4. Recorrer el nuevo camino desde el principio
 - Si es más corto insertarlo como mejor camino, propagar el cambio, reorganizar el grafo y volver a (1)
 - Si no es más corto volver a (1)



RESOLUCIÓN DE PROBLEMAS

- La utilización de grafos de búsqueda reduce los esfuerzos de exploración en el espacio de estados
- La utilización de grafos de búsqueda obliga a comprobar si cada estado generado ha sido generado ya en pasos anteriores
- La utilización de grafos de búsqueda demanda menos recursos de memoria, pero suponen un coste computacional mayor
- Los árboles de búsqueda son más eficientes desde una perspectiva computacional, pero tienen mayores problemas de memoria



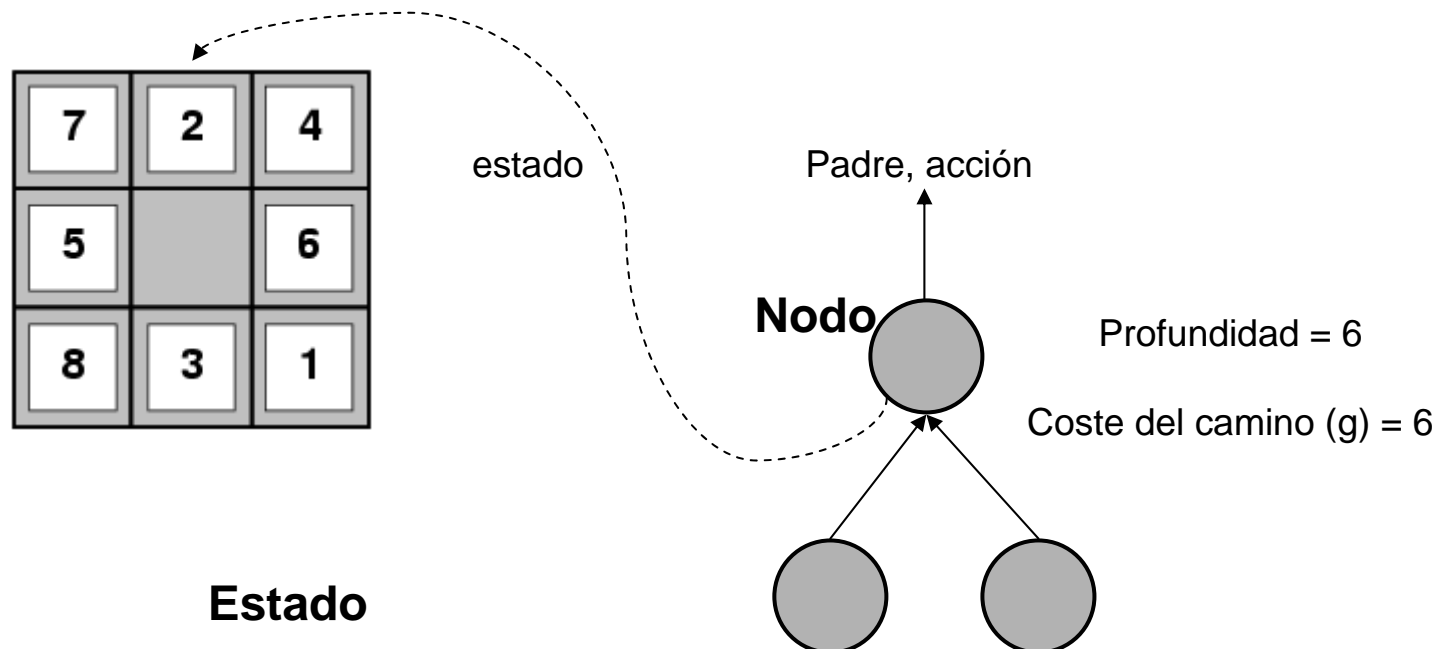
RESOLUCIÓN DE PROBLEMAS

- EL PROBLEMA DE LA REPRESENTACIÓN
 1. Representación de objetos, entidades relevantes, o hechos del dominio (NODOS)
 - Representación estática del dominio
 2. Representación de relaciones entre objetos, entidades relevantes, o hechos del dominio (OPERADORES)
 - Representación dinámica del dominio
 3. Representación de las secuencias de estados surgidas durante los procesos de búsqueda (ESTRATEGIAS)
 - Representación dinámica del proceso de búsqueda de soluciones
- Las tres representaciones están estrechamente relacionadas



Estados vs Nodos

- El nodo es una estructura de datos contable
- El estado es una configuración del mundo
- Dos nodos diferentes pueden contener el mismo estado





RESOLUCIÓN DE PROBLEMAS

- SELECCIÓN SISTEMÁTICA DE OPERADORES RELEVANTES
 - Al proceso de selección sistemática de operadores relevantes se denomina emparejamiento.
 - Reconocer operadores aplicables al estado actual
 - Es una de las tareas más costosas de los programas de IA
 - Dos tipos:
 - n literal
 - n con variables.



RESOLUCIÓN DE PROBLEMAS

➤ Emparejamiento literal

- Búsqueda simple de operadores del conjunto O del espacio de estados
 - n A través de las precondiciones del operador si el proceso es dirigido por los datos
 - n A través de los consecuentes del operador si el proceso es dirigido por los objetivos
- Problemas:
 - n Si O es muy grande el emparejamiento literal es muy ineficiente.
 - n No es siempre evidente qué operador es aplicable



RESOLUCIÓN DE PROBLEMAS

- Casos particulares del emparejamiento literal
 - El operador empareja completamente con el estado
 - Las precondiciones del operador son un subconjunto de la descripción del estado actual
 - Las precondiciones del operador coinciden sólo parcialmente con la descripción del estado actual
 - Las precondiciones del operador no coinciden con la descripción del estado actual
- Útil en dominios pequeños



RESOLUCIÓN DE PROBLEMAS

- **Emparejamiento con variables**

- De naturaleza no literal
- Útil cuando el problema requiere una búsqueda extensa en la que haya variables involucradas.

- **EJEMPLO:**

- **Hechos de un dominio:**

- Juan es hijo de María → HIJO(María,Juan)
- Pedro es hijo de Juan → HIJO(Juan,Pedro)
- Tomás es hijo de Pedro → HIJO(Pedro,Tomás)
- Rosa es hija de Pedro → HIJA(Pedro,Rosa)
- Ana es hija de Juan → HIJA(Juan,Ana)
- Rosa es hija de Ana → HIJA(Ana,Rosa)

- **Operadores del dominio:**

- Op1: HIJO(x,y) and HIJO(y,z) → NIETO(x,z)
- Op2: HIJA(x,y) and HIJO(y,z) → NIETO(x,z)
- Op3: HIJO(x,y) and HIJA(y,z) → NIETA(x,z)

Emparejamiento con variables



Hechos del dominio

Juan es hijo de María HIJO (María, Juan)
Pedro es hijo de Juan HIJO (Juan, Pedro)
Tomás es hijo de Pedro HIJO (Pedro, Tomás)
Rosa es hija de Pedro HIJA (Pedro, Rosa)
Ana es hija de Juan HIJA (Juan, Ana)
Rosa es hija de Ana HIJA (Ana, Rosa)

Operadores del dominio

op1: HIJO(x,y) and HIJO(y,z) \rightarrow NIETO (x,z)
op2: HIJA(x,y) and HIJO(y,z) \rightarrow NIETO (x,z)
op3: HIJO(x,y) and HIJA(y,z) \rightarrow NIETA (x,z)

Problema

Dados los hechos anteriores (estado inicial) encontrar un estado meta que incluya los mismos hechos y un hecho nuevo que indique quién es el nieto de Juan.

Emparejamiento con variables



Hechos del dominio

Juan es hijo de María HIJO (María, Juan)
Pedro es hijo de Juan HIJO (Juan, Pedro)
Tomás es hijo de Pedro HIJO (Pedro, Tomás)
Rosa es hija de Pedro HIJA (Pedro, Rosa)
Ana es hija de Juan HIJA (Juan, Ana)
Rosa es hija de Ana HIJA (Ana, Rosa)

Operadores del dominio

op1: HIJO(x,y) and HIJO(y,z) \rightarrow NIETO (x,z)

op2: HIJA(x,y) and HIJO(y,z) \rightarrow NIETO (x,z)

op3: HIJO(x,y) and HIJA(y,z) \rightarrow NIETA (x,z)

Solución

- Interesa usar op1-op2, porque son los operadores que concluyen sobre la existencia de algún nieto (x=Juan)
- Hay que encontrar un “y” que verifique, para algún valor de “z”, que

HIJO(Juan,y) and HIJO(y,z)

Emparejamiento con variables



Hechos del dominio

Juan es hijo de María HIJO (María, Juan)
Pedro es hijo de Juan HIJO (Juan, Pedro)
Tomás es hijo de Pedro HIJO (Pedro, Tomás)
Rosa es hija de Pedro HIJA (Pedro, Rosa)
Ana es hija de Juan HIJA (Juan, Ana)
Rosa es hija de Ana HIJA (Ana, Rosa)

Operadores del dominio

op1: $\text{HIJO}(x,y) \text{ and } \text{HIJO}(y,z) \rightarrow \text{NIETO}(x,z)$

op2: $\text{HIJA}(x,y) \text{ and } \text{HIJO}(y,z) \rightarrow \text{NIETO}(x,z)$

op3: $\text{HIJO}(x,y) \text{ and } \text{HIJA}(y,z) \rightarrow \text{NIETA}(x,z)$

Solució

- Opción 1: Verificar a todos los hijos de Juan y verificar que alguno de ellos tiene un hijo
- Opción 2: Comprobar que de todos los que tienen algún hijo, hay alguno que es hijo de Juan



RESOLUCIÓN DE PROBLEMAS

- Resolución de Conflictos
 - Relacionado con el emparejamiento, un conflicto es una situación en la que más de un operador empareja con nuestro estado actual
 - En función de la estrategia, cuando aparece un conflicto ¿Qué operador(es) ejecutamos?
- Reglas de carácter general
 1. Si podemos evitarlo no aplicaremos operadores ya aplicados
 2. Trataremos de aplicar primero operadores que emparejen con hechos recientemente incorporados
 3. Aplicaremos primero operadores con precondiciones más restrictivas
 4. De no poder discriminar con las pautas anteriores, ejecutaremos uno (o más, depende de la estrategia), al azar



RESOLUCIÓN DE PROBLEMAS

- Funciones Heurísticas
 - Funciones de carácter numérico que nos permiten estimar el beneficio de una determinada transición en el espacio de estados
 - Se utilizan para optimizar los procesos de búsqueda
 - Tratan de guiar la exploración del espacio de estados de la forma más provechosa posible
 - Sugieren el mejor camino a priori, cuando disponemos de varias alternativas
 - El estudio de las funciones heurísticas se denomina Heurética



RESOLUCIÓN DE PROBLEMAS

- Estrategias de exploración del espacio de estados (búsqueda)
 - La eficacia de los procesos de búsqueda suele venir determinada por la estrategia empleada, y por los mecanismos diseñados para controlar la aplicación del conocimiento del dominio
- Técnicas de búsqueda de propósito específico
 - Diseñadas “ad hoc” para resolver un problema concreto
- Técnicas de búsqueda de propósito general
 - Métodos débiles de exploración del espacio de estados
 - Cualquier método débil de exploración configura una búsqueda que será:
 - En anchura
 - En profundidad
 - Mixta anchura-profundidad

Estrategias de búsqueda: Evaluación



- Una estrategia se define por el orden de expansión de los nodos.
- Se evalúan de acuerdo a:
 - Completitud:
 - ¿Encuentra siempre la solución? (si existe)
 - Complejidad temporal:
 - ¿Cuánto tarda en encontrar una solución?
 - Complejidad espacial:
 - ¿Cuánta memoria necesita?
 - Optimización:
 - ¿encuentra siempre la solución de mayor calidad? (si existan varias soluciones)

Estrategias de búsqueda:

Evaluación



- La complejidad temporal y espacial se miden en términos de:
 - b: factor de ramificación (número promedio de sucesores por nodo)
 - d: profundidad de la solución menos costosa (o del nodo meta óptimo)
 - m: máxima profundidad del espacio de estados (o de cualquier camino)
 - Complejidad Temporal → máx. número nodos generados
 - Complejidad Espacial → máx. número nodos almacenados
- Efectividad:
 - Coste total = coste búsqueda + coste camino

Estrategias de búsqueda: Implementación



- El espacio de estados define un grafo de búsqueda implícito (nodo inicial y función sucesor), no un árbol. Los algoritmos sólo conservarán un padre para cada nodo.
- *Expansión* de un nodo: aplicación de operadores relevantes y generación de sucesores
- Listas de *abiertos* (*frontera*): nodos en espera de ser expandidos.
- Lista de *cerrados*: nodos expandidos. A efectos de implementación son colas.
- Estrategia de búsqueda: selección del siguiente nodo a expandir de la lista de abiertos. El tipo de cola afectará al orden de la búsqueda.
- Tratamiento de nodos repetidos: No se expande el nodo actual si pertenece a la lista de cerrados.

Estrategias de búsqueda: Implementación



```
función BUSQUEDA–GRAFO (problema, abiertos)
devuelve una solución o fallo
  cerrados ← un conjunto vacío
  abiertos ← INSERTAR (HACER_NODO (
                                ESTADO_INICIAL [problema]), abiertos)

  bucle hacer
    /* No hay candidatos para la expansion */
    if ESVACIA (abiertos) entonces devolver fallo

    /* Elegir un nodo hoja para expansion */
    nodo ← QUITAR_FRENTE (abiertos)

    si TEST_META [problema] (ESTADO [nodo])
      entonces devuelve SOLUCIÓN (nodo)
    si ESTADO [nodo] no esta en cerrados entonces
      añadir ESTADO [nodo] a cerrados
      abiertos ← INSERTAR–TODOS (
                                EXPANDIR (nodo, problema), abiertos)
```

Estrategias de búsqueda: Implementación



```
funcion EXPANDIR (nodo,problema)
devuelve un conjunto de nodos

    sucesores <- el conjunto vacio
    para cada par <accion,resultado> en
        FUNCION_SUCESOR[problema] aplicada a ESTADO[nodo] hacer
            s <- un nuevo NODO
            ESTADO[s] <- resultado
            NODO_PADRE[s] <- nodo
            ACCION[s] <- accion
            COSTE_CAMINO[s] <- COSTE_CAMINO[nodo] +
                                COSTE_PASO (nodo,accion,s)
            PROFUNDIDAD[s] <- PROFUNDIDAD[nodo] + 1
            anadir s a sucesores
    devolver sucesores
```



Estrategias no informadas

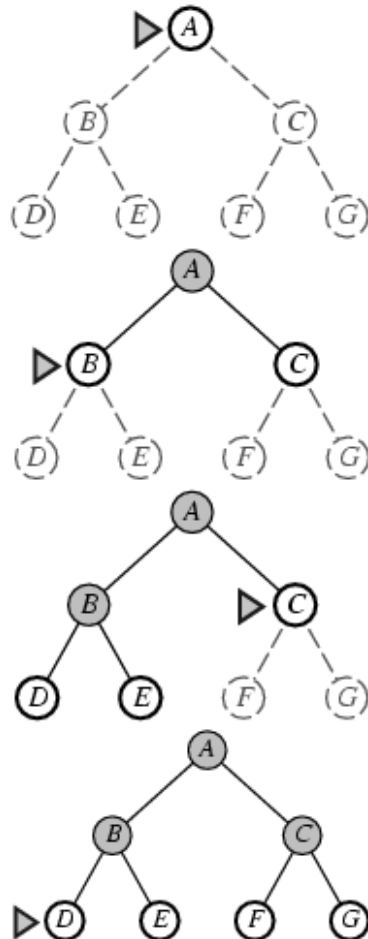
- Las búsquedas ciegas no tienen preferencia sobre el siguiente estado (nodo) a expandir.
- Los distintos tipos de búsquedas ciegas se caracterizan por el orden de expansión de los nodos.
- Esto tendrá un efecto directo sobre la calidad del proceso con respecto a los cuatro criterios definidos.
- Tipos:
 - Preferente por amplitud: Variante de coste uniforme
 - Preferente por profundidad: Variante limitada en profundidad y profundización iterativa

Búsqueda preferente en amplitud (“breadth-first”)



- Expandir los nodos no expandidos menos profundos.
- Se expanden los nodos de un nivel antes de acceder a nodos de niveles inferiores.
- Implementación: *Abiertos* es una cola FIFO, *i.e. nuevos sucesores al final*.

Búsqueda preferente en amplitud ("breadth-first")



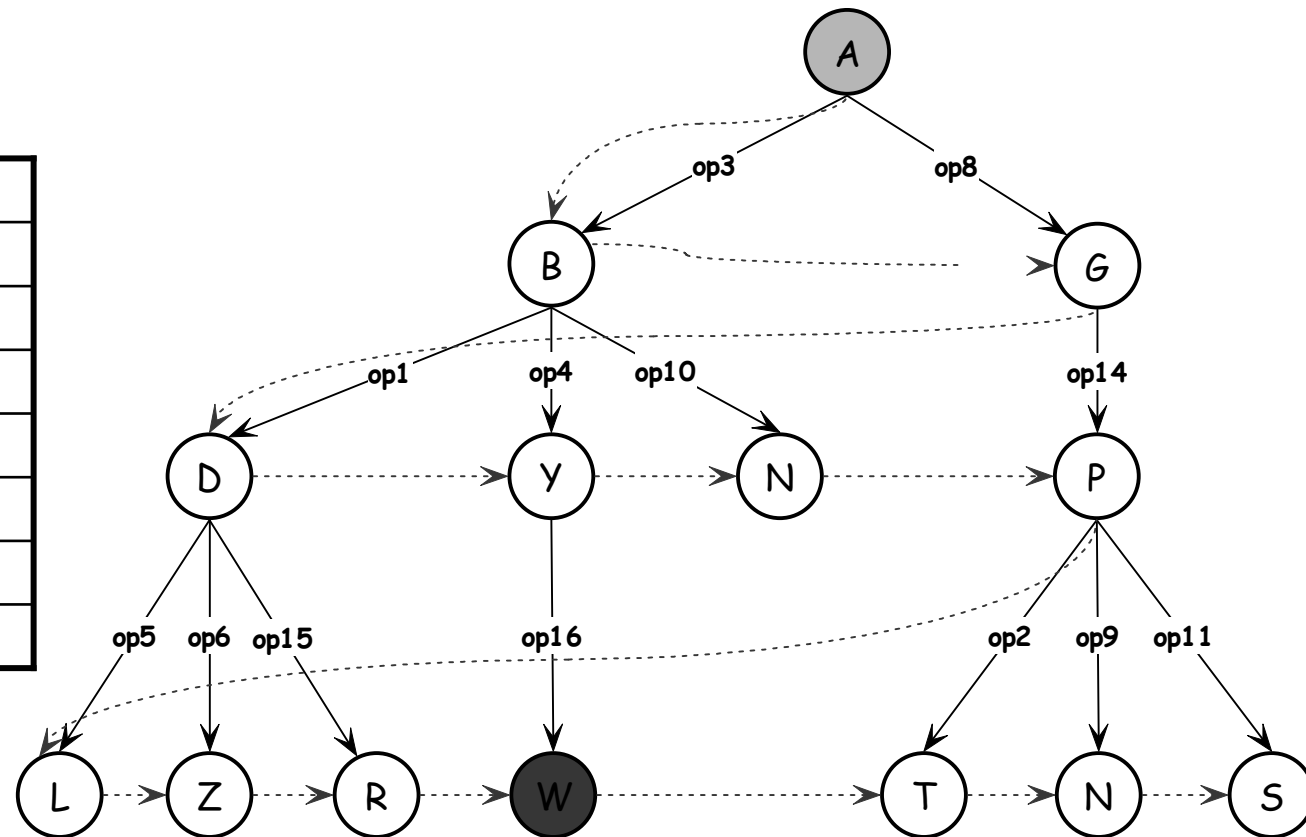
- Cola ← camino conteniendo sólo la raíz
- mientras no está vacía la cola y la meta no se ha alcanzado hacer
 - Eliminar el primer nodo de la cola
 - Aplicar operadores relevantes
 - Crear los sucesores
 - Añadir los nuevos nodos al final de la cola
- Si se ha alcanzado la meta devolver éxito sino fallo

Búsqueda preferente en amplitud ("breadth-first")



- $I = [A]$; $M = [W, F, K]$;

1	$B \rightarrow D$	9	$P \rightarrow N$
2	$P \rightarrow T$	10	$B \rightarrow N$
3	$A \rightarrow B$	11	$P \rightarrow S$
4	$B \rightarrow Y$	12	$Q \rightarrow Y$
5	$D \rightarrow L$	13	$E \rightarrow F$
6	$D \rightarrow Z$	14	$G \rightarrow P$
7	$H \rightarrow V$	15	$D \rightarrow R$
8	$A \rightarrow G$	16	$Y \rightarrow W$





RESOLUCIÓN DE PROBLEMAS

- Procedimiento sistemático
- Solución aceptable
 - A 3 B 4 Y 16 W
- Otros caminos explorados
 - A 3 B 1 D 5 L
 - A 3 B 1 D 6 Z
 - A 3 B 1 D 15 R
 - A 3 B 10 N
 - A 8 G 14 P 2 T
 - A 8 G 14 P 9 N
 - A 8 G 14 P 11 S
- Caminos no resolutivos, pero no necesariamente desaprovechables
- Consume grandes recursos de memoria
- Es ineficiente temporalmente por la gran expansión de nodos
- Siempre encuentra la mejor solución, si existe, y si puede

Búsqueda preferente en amplitud (“breadth-first”)



- El problema del 8 puzzle

- Estado inicial

2	8	3
1	6	4
7		5

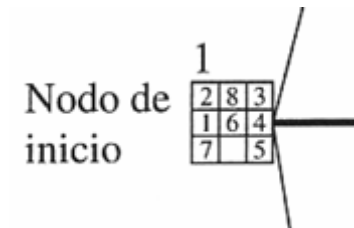
- Operadores:

- Mover a la derecha
 - Mover abajo
 - Mover a la izquierda
 - Mover arriba

- Estado final

1	2	3
8		4
7	6	5

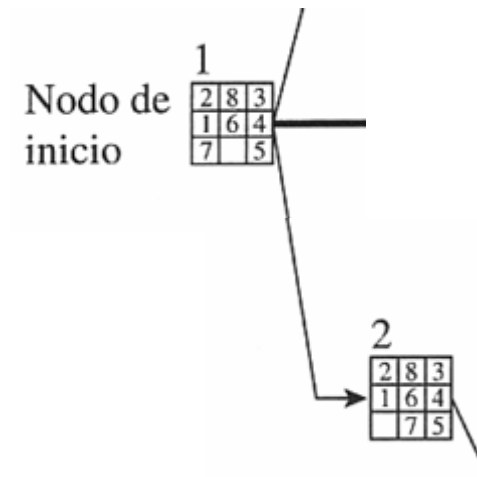
Búsqueda preferente en amplitud ("breadth-first")





El problema del 8-puzzle

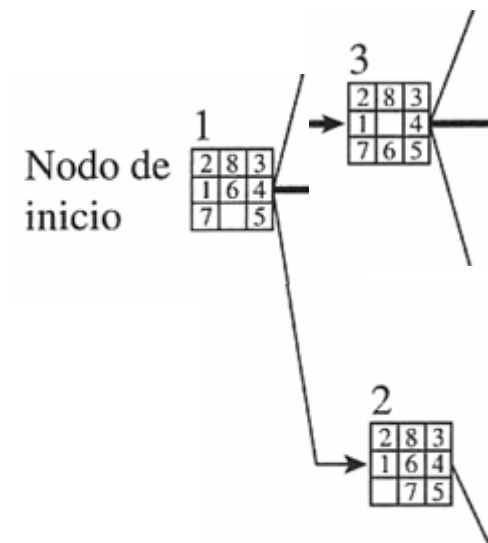
- Operador: Mover a la derecha





El problema del 8-puzzle

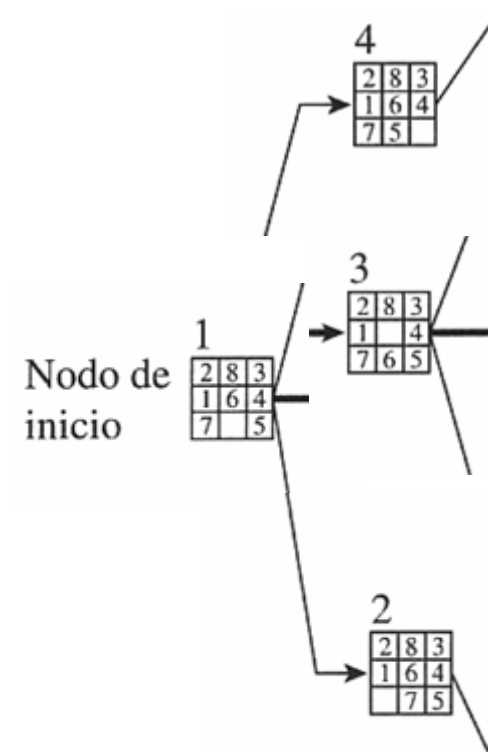
- Operador: Mover abajo

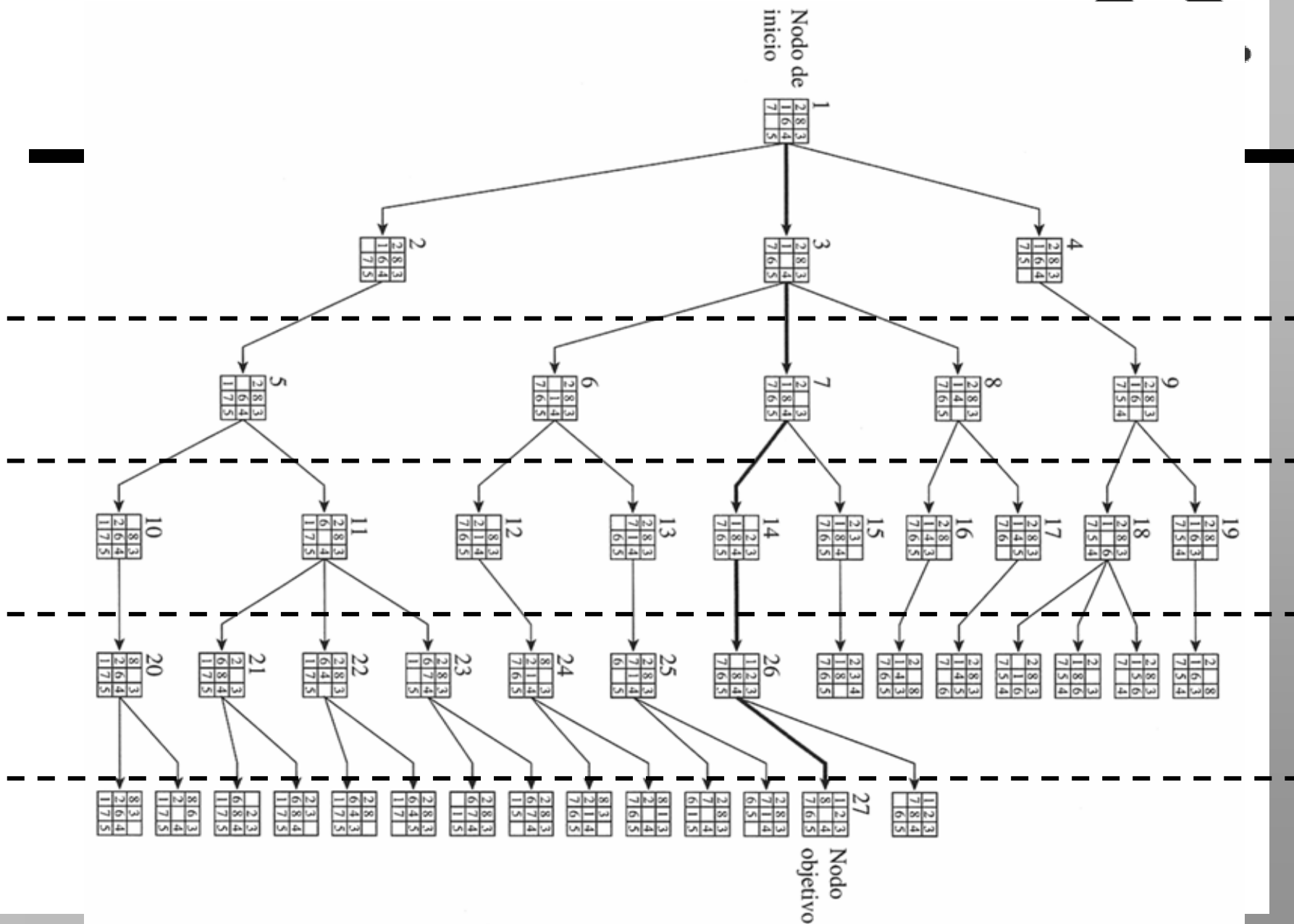




El problema del 8-puzzle

- Operador: Mover a la izquierda





Búsqueda preferente en amplitud ("breadth-first")

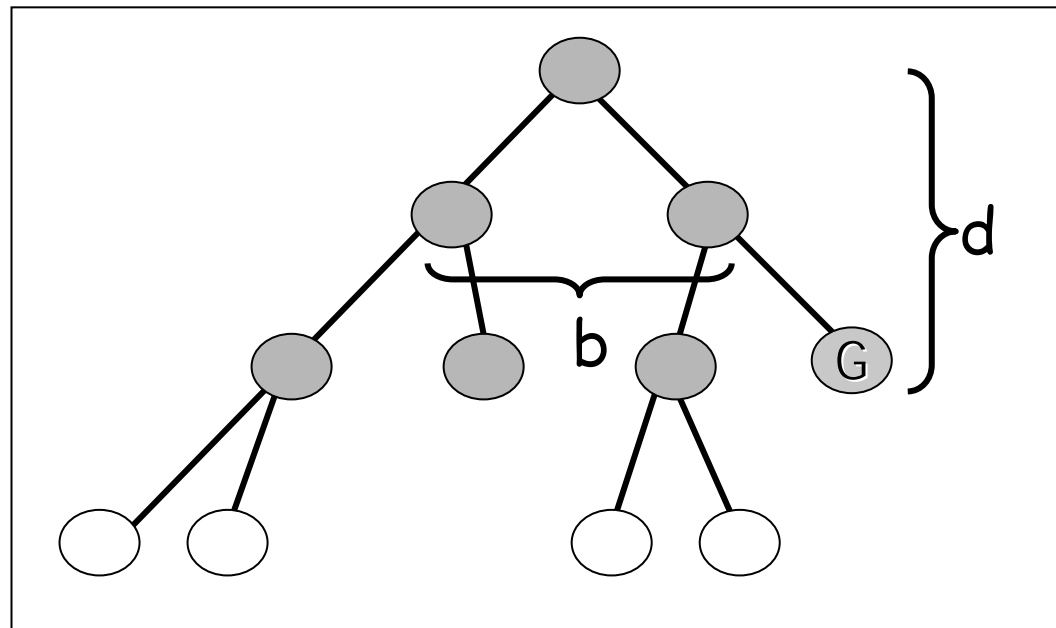


- Completa:
 - Si, si el nodo meta más superficial está a una profundidad finita d , la búsqueda en anchura lo encuentra expandiendo antes los nodos más próximos (suponiendo b finito).
- Óptima:
 - En general, no es óptima (el nodo meta más cercano no tiene por qué ser el óptimo)
 - Si, si el coste del camino no decrece al aumentar la profundidad (ejemplo: todas las acciones tienen el mismo coste)

Búsqueda preferente en amplitud: Complejidad temporal



- Si la meta se encuentra a una profundidad d , hay que crear todos los nodos de ese nivel y en el caso peor $b^{d+1}-b$ del siguiente

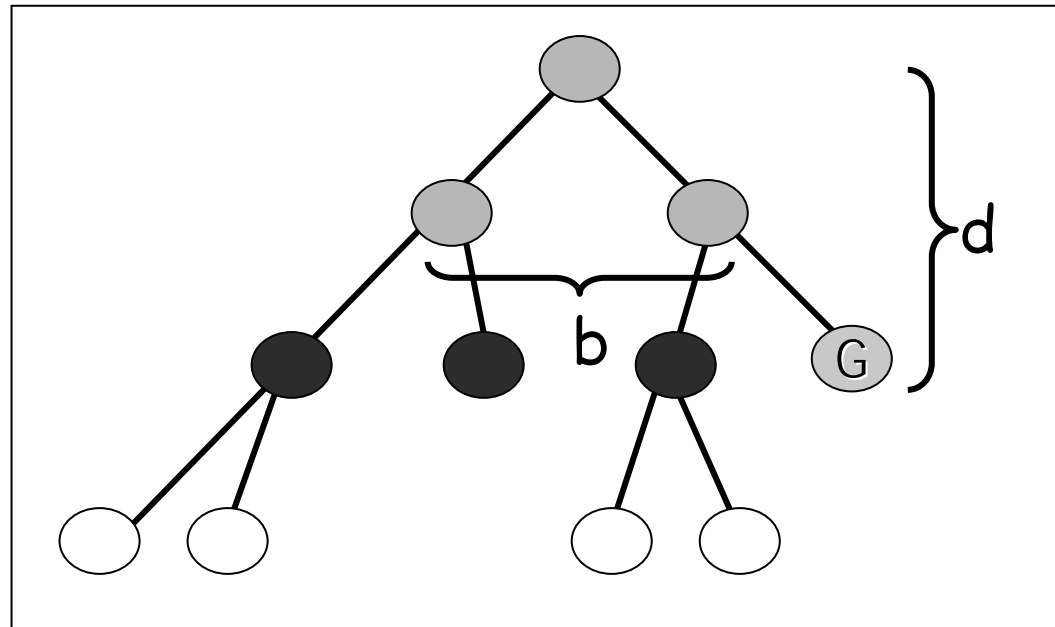


- Así pues: $b+b^2 +b^3 +\dots +(b^{d+1} -b)=O(b^{d+1})$

Búsqueda preferente en amplitud: Complejidad espacial



- Al nivel del nodo meta se almacena la mayor cantidad de nodos.



- $1 + b + b^2 + b^3 + \dots + (b^{d+1} - b) = O(b^{d+1})$

Búsqueda preferente en amplitud (“breadth-first”)



Profundidad	Nodos	Tiempo	Espacio
0	1	1 ms.	100 b
2	111	0.1 seg.	11 Kb
4	11.111	11 seg.	1 Mb
6	10^6	18 min.	111 Mb
8	10^8	31 horas	11 Gb
10	10^{10}	128 días	1 Tb
12	10^{12}	35 años	111 Tb
14	10^{14}	3.500 años	11.111 Tb

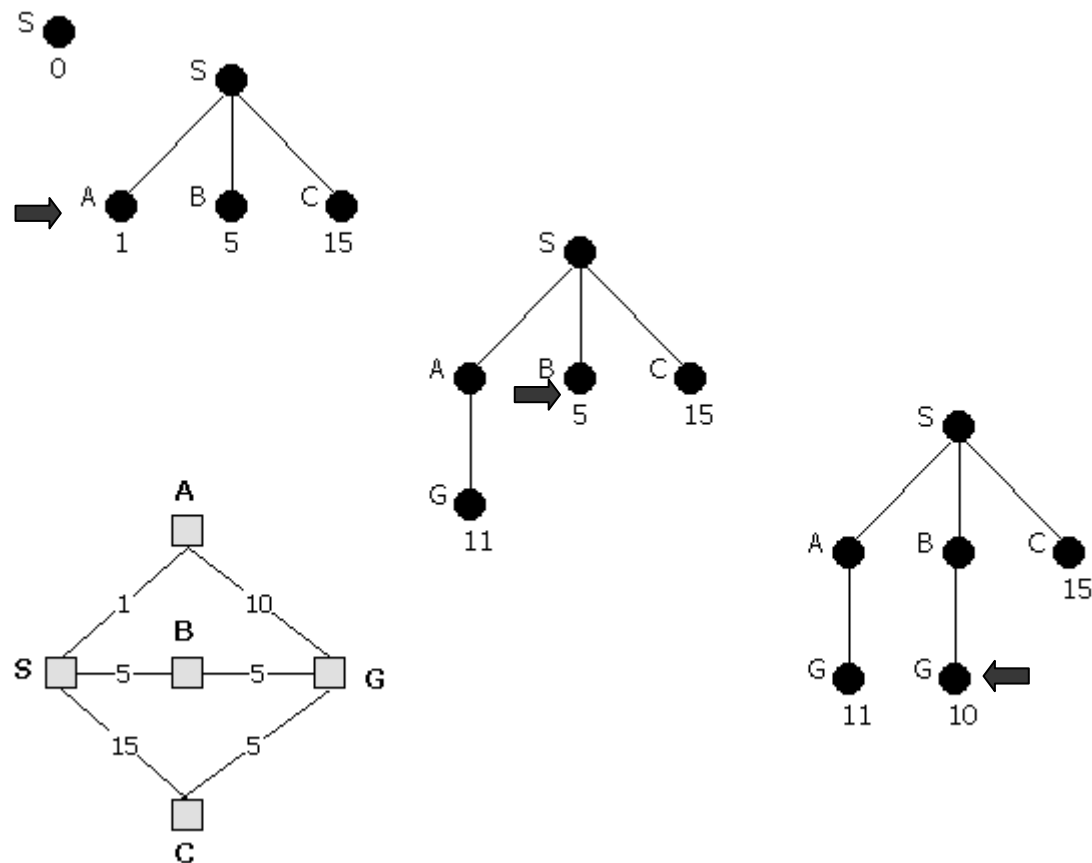
b=10, 1000 nodos/seg y 100
bytes por nodo

Búsqueda de coste uniforme



- Anchura es óptimo si los costes de transición son constantes.
- Por extensión: algoritmo óptimo con cualquier función de coste.
- Expandir nodos con el menor coste.
- Implementación: Lista de abiertos ordenada por coste

Búsqueda de coste uniforme



Búsqueda de coste uniforme



- Completa: Si, si el coste de cada paso es $\geq \epsilon$ (cte positiva)
- Óptima: nodos se expanden en orden creciente de coste (el primer nodo meta será el óptimo)
- Complejidad:
 - Espacio-Temporal: número de nodos con $g \leq \text{coste solución óptima}$ $O(b^{\lceil C^*/\epsilon \rceil})$
 - C^* coste solución óptima, y ϵ coste de cada acción
 - Si coste de los pasos es el mismo $b^{\lceil C^*/\epsilon \rceil} = b^d$

Búsqueda preferente en profundidad (“depth-first”)



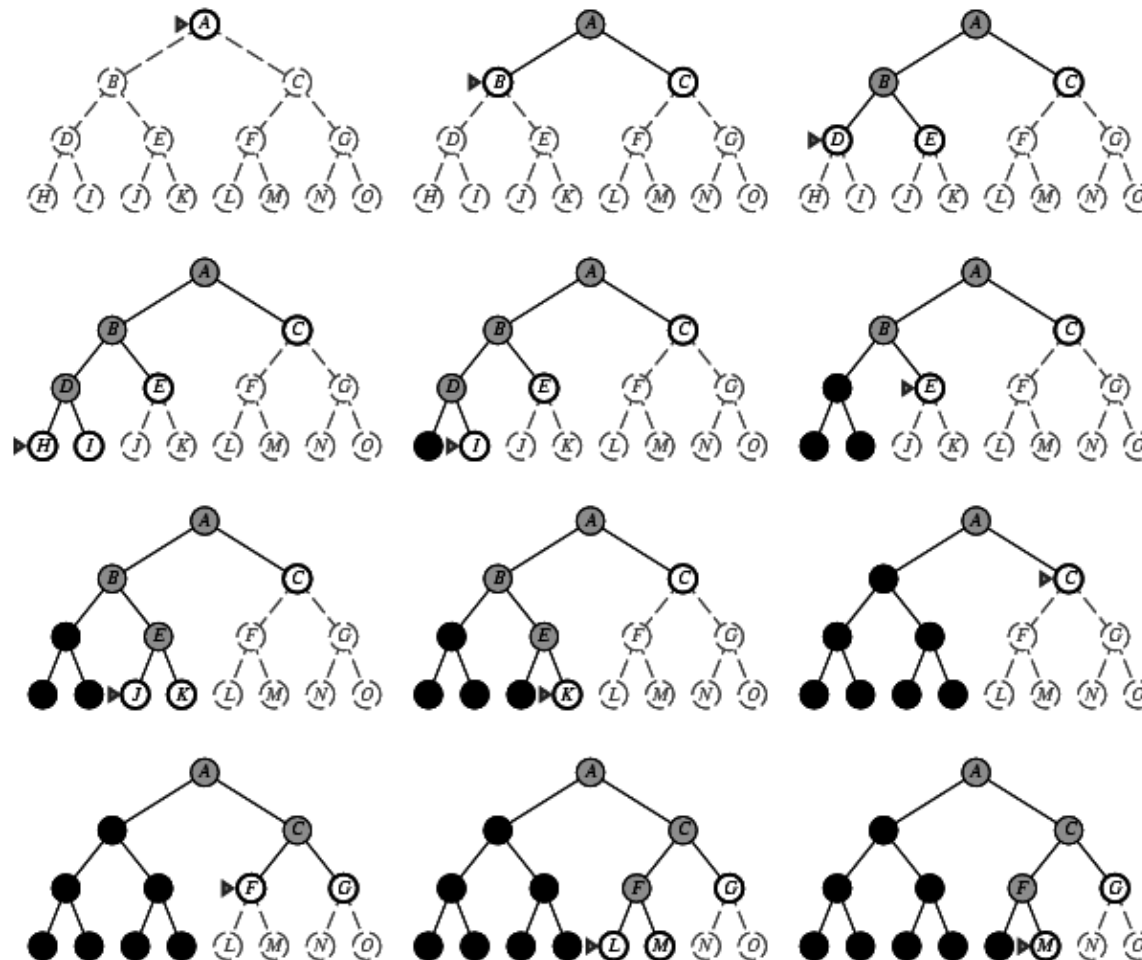
- Expandir el nodo más profundo de la frontera actual del árbol.
- Implementación: *Abiertos* es una cola LIFO (pila), *i.e. nuevos sucesores al principio*.
- Si se produce un callejón sin salida, la búsqueda vuelve al siguiente nodo más profundo con sucesores sin explorar

Búsqueda preferente en profundidad (“depth-first”)



- Cola ← camino conteniendo sólo la raíz
- Mientras no está vacía la cola y la meta no se ha alcanzado hacer
 - Eliminar el primer nodo de la cola
 - Aplicar operadores relevantes
 - Crear los sucesores
 - Añadir los nuevos nodos al principio de la cola
- Si se ha alcanzado la meta devolver éxito, si no fallo

Búsqueda preferente en profundidad ("depth-first")

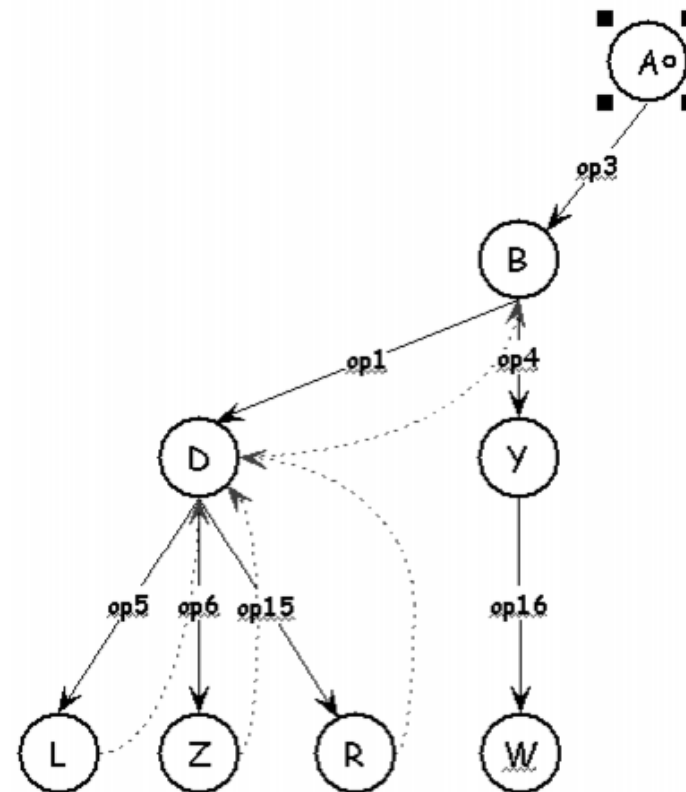




Búsqueda preferente en profundidad ("depth-first")

- $I = [A]$; $M = [W, F, K]$;

1	$B \rightarrow D$	9	$P \rightarrow N$
2	$P \rightarrow T$	10	$B \rightarrow N$
3	$A \rightarrow B$	11	$P \rightarrow S$
4	$B \rightarrow Y$	12	$Q \rightarrow Y$
5	$D \rightarrow L$	13	$E \rightarrow F$
6	$D \rightarrow Z$	14	$G \rightarrow P$
7	$H \rightarrow V$	15	$D \rightarrow R$
8	$A \rightarrow G$	16	$Y \rightarrow W$



Búsqueda preferente en profundidad (“depth-first”)

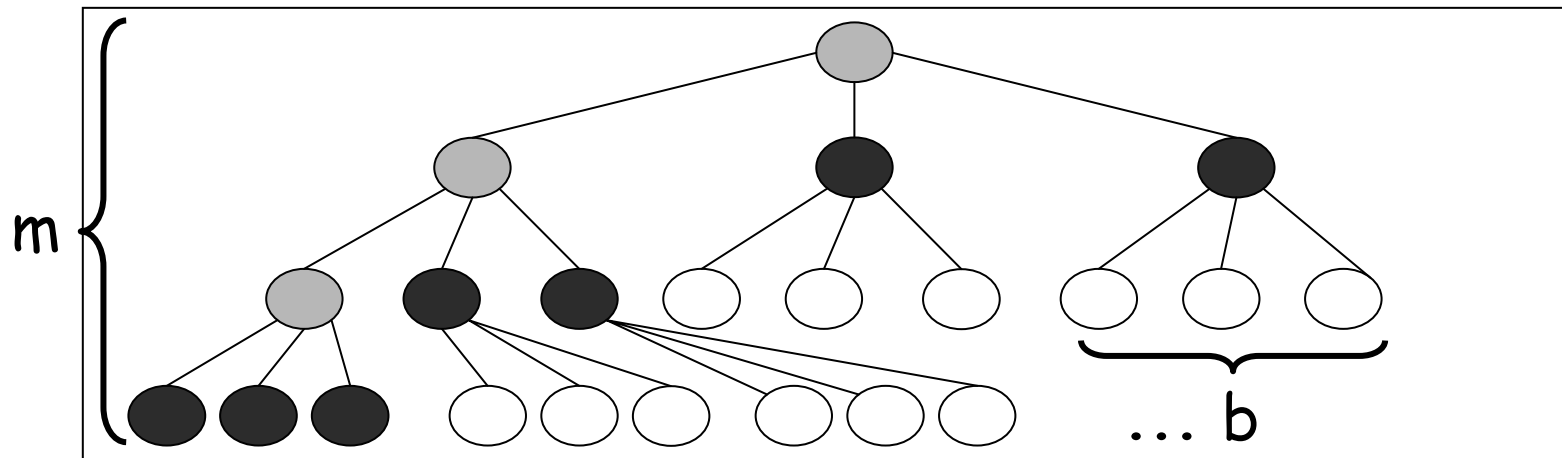


- Completo:
 - No, falla en espacios de búsqueda infinitos, espacios con bucles, ...
 - Sí lo es, considerando estados repetidos.
- Óptimo:
 - No, puede encontrar una solución más profunda que otra en una rama no expandida



Búsqueda preferente en profundidad: Complejidad espacial

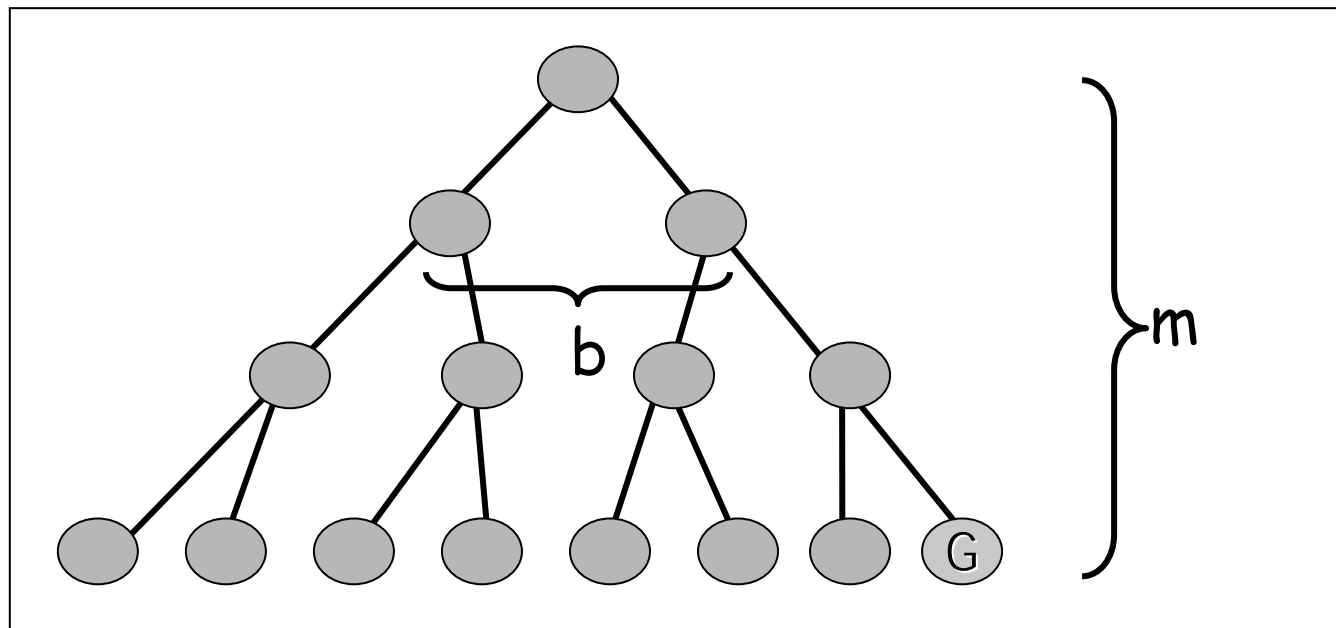
- El mayor número de nodos almacenados se alcanza en el nodo inferior de más a la izquierda.
- Sólo almacena un camino (raíz-hoja), el resto sin solución se eliminan
- En general: $1 + b + b + \dots^m \dots + b = 1 + b * m = O(bm)$





Búsqueda preferente en profundidad: Complejidad temporal

- En el peor caso genera todos los b^m nodos del árbol (m puede ser mucho mayor que d)
- $b^m + b^{m-1} + \dots + 1 = O(b^m)$



Búsqueda preferente en profundidad (“depth-first”)



- Variantes:
 - Limitada en profundidad:
 - fijar un límite (l) a la profundidad máxima
 - Soluciona caminos infinitos
 - Si $l < d$ (meta más allá del límite) incompleto
 - Si $l > d$ no es óptimo
 - Complejidad espacial y temporal $O(b^l)$
 - Profundización iterativa:
 - Incrementar gradualmente el límite (i.e. 0, 1, 2, ...)
 - Combina los beneficios de la búsqueda en anchura y en profundidad
 - Para espacios de búsqueda muy grandes en los que no se conoce la profundidad de la solución

Estrategias de búsqueda no informadas: Comparativa



	Amplitud	Coste Uniforme	Profundidad	Profundidad limitada	Profundización iterativa
Completa	Si ^a	Si ^{a,b}	No	Si ($l \geq d$)	Si ^a
Óptima	Si (coste cte)	Si	No	No	Si ^c
Complejidad Espacial	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Complejidad Temporal	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$

b: factor de ramificación

d: profundidad de la solución menos profunda

m: máxima profundidad del árbol de búsqueda

l: límite de profundidad

^a completa si b es finito

^b completa si coste de paso $\geq \epsilon$

^c óptima si los costes de transición son iguales



RESOLUCIÓN DE PROBLEMAS

- Método en profundidad
 - Sigue caminos completos hasta agotarlos o hasta que se encuentra una solución
 - Si no se encuentra una solución por un camino, se organiza una vuelta atrás o "backtracking"
 - En ocasiones, los caminos generados no se explotan completamente, pudiendo ser abandonados llegados a una determinada profundidad
 - Permiten que la solución se encuentre por casualidad
 - Demandan menos recursos de memoria que el procedimiento en anchura
 - Demandan más recursos computacionales que el procedimiento en anchura
 - No siempre se encuentra la mejor solución (si se encuentra o si existe)

Estrategias de búsqueda heurística*



- Poseen conocimiento acerca de si un estado no meta es más prometedor que otro estado dado.
- *Búsqueda preferente por el mejor:*
 - Basados en la búsqueda en anchura
 - Examinar primero nodos que, de acuerdo con la heurística, están situados en el mejor camino hacia el objetivo.
- *Búsqueda local:*
 - Tratar de mejorar el estado actual en lugar de explorar de manera sistemática los caminos desde el estado inicial.
 - El coste del camino es irrelevante y lo único importante es alcanzar el estado meta.

*Del verbo griego *heuriskein*, que significa “encontrar” o “descubrir”

Búsquedas por el mejor nodo



- Función de evaluación heurística, f , definida sobre la descripción del estado.
- Decide cuál es el mejor nodo a expandir. Tomará valores pequeños en los nodos más prometedores.
- Se expandirá el nodo n para el que se obtenga el menor valor f .
- Se termina el proceso cuando el nodo a expandir sea un nodo objetivo.
- Se implementa con una cola de prioridad que mantiene en orden ascendente la lista de nodos abiertos según el valor de f .

Tipos de búsqueda por el mejor nodo

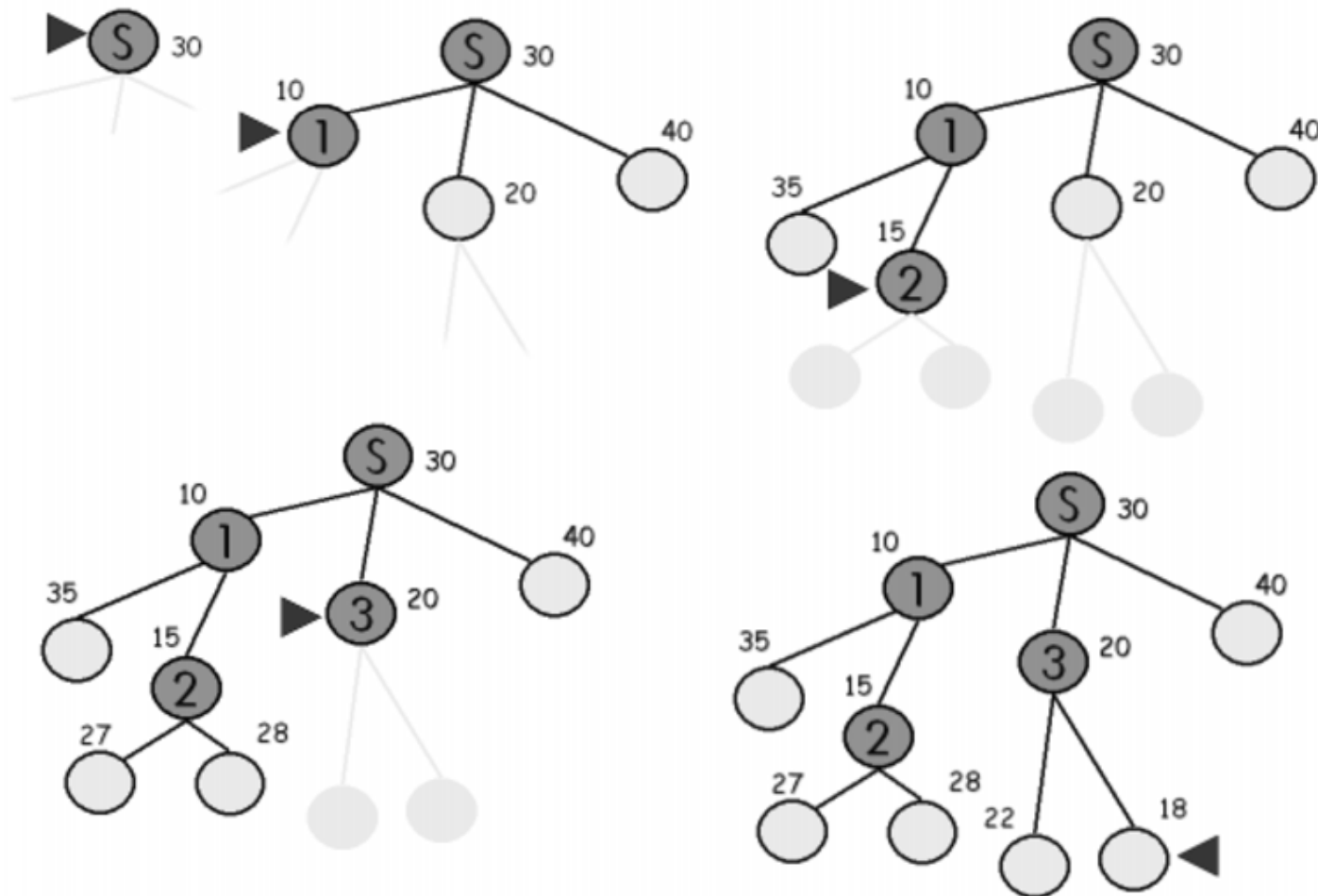


- Si $f(n)$ sólo considera el coste mínimo estimado para llegar a una solución a partir de un nodo n , $h(n)$ o función heurística.
 - $f(n)=h(n) \rightarrow$ **Búsqueda avara***
- Si $f(n)$ considera el coste mínimo total del camino a un nodo solución que pase por el nodo n
 - $f(n)=g(n)+h(n) \rightarrow$ **Búsqueda A* (A asterisco o estrella)**
 - $g(n)=$ coste consumido para llegar a n
- $h(n)=0$ si n es un nodo meta.

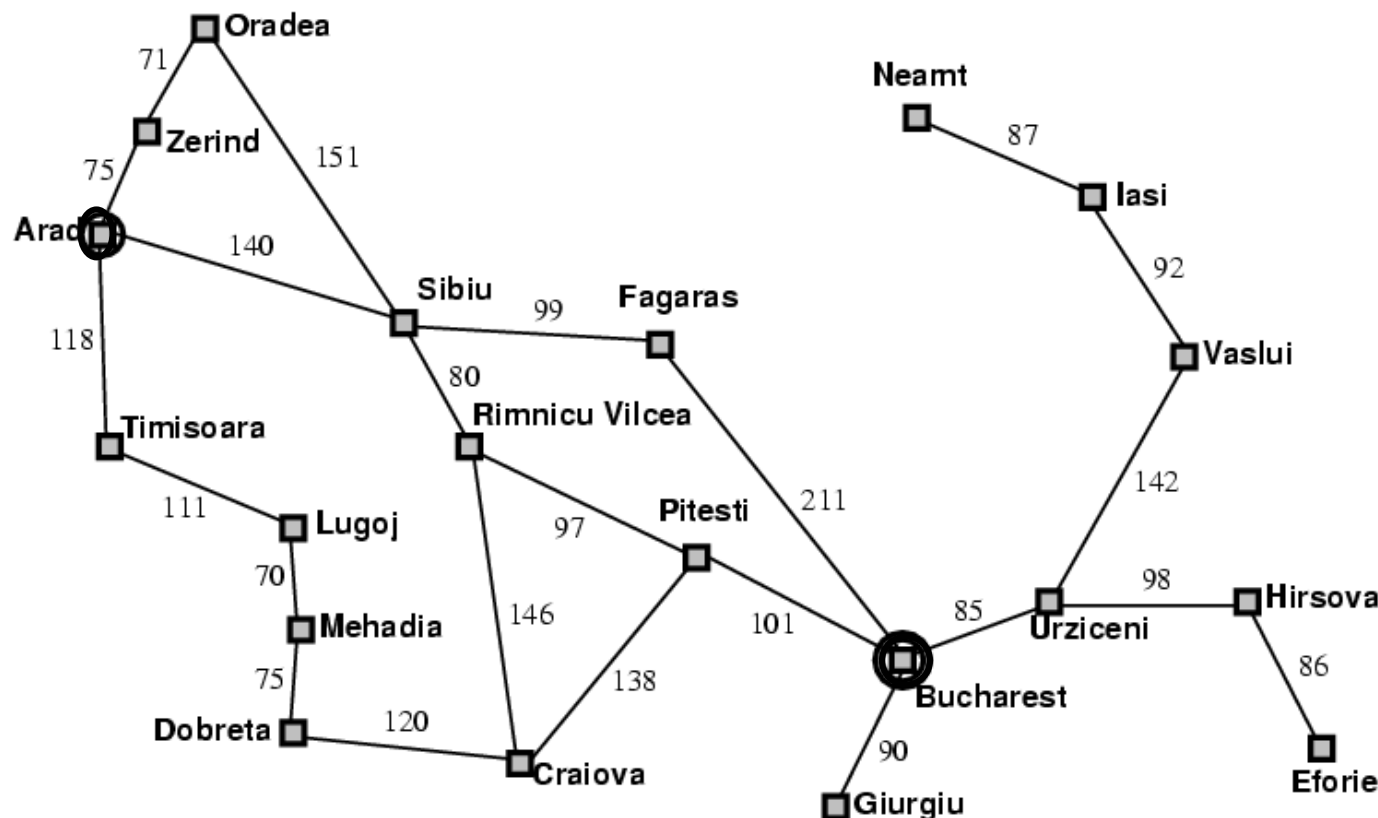
*Algunos autores la conocen como búsqueda por el mejor nodo o best-first



Búsqueda avara



Búsqueda avara: El problema del viajante



Heurística = Distancia en línea recta



Búsqueda avara: El problema del viajante



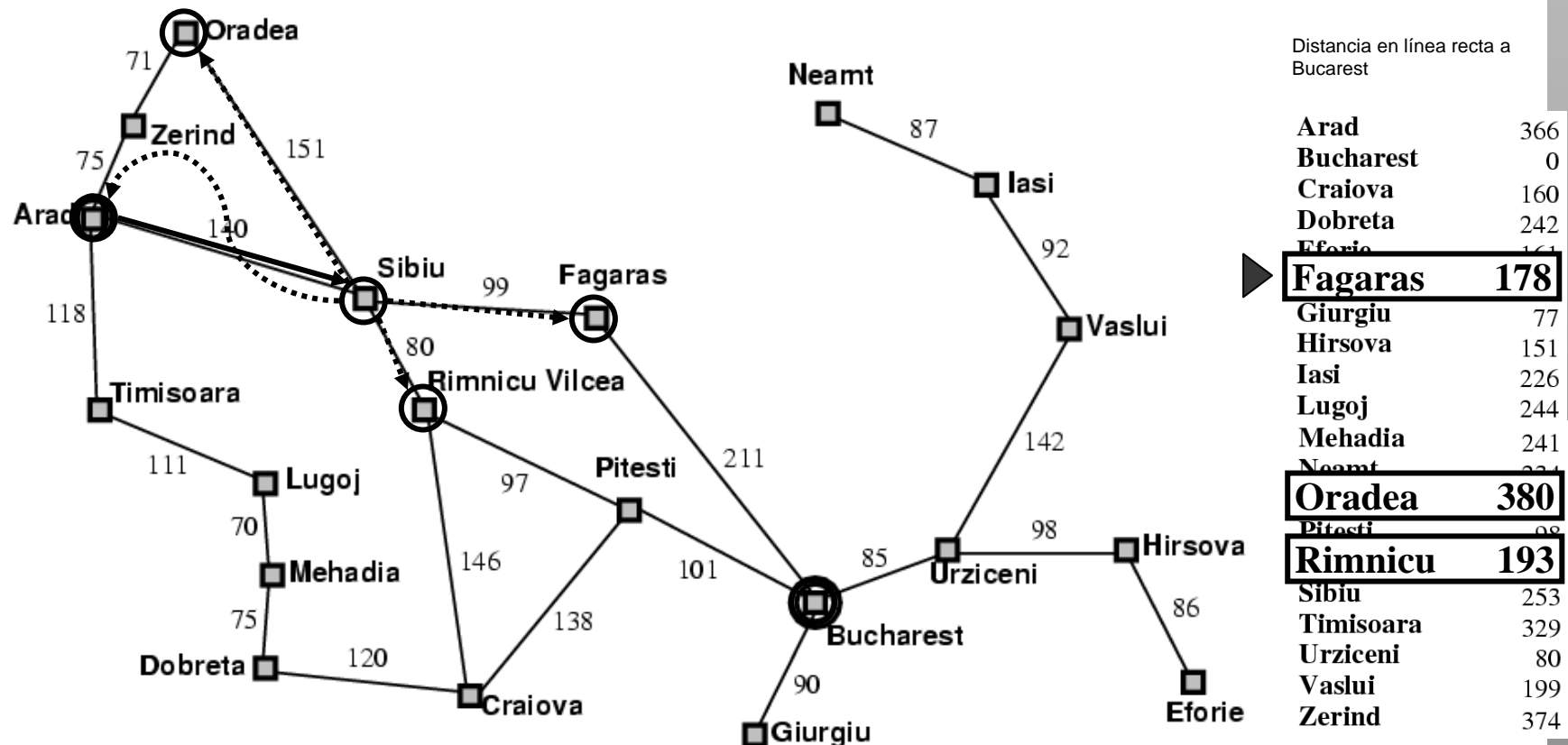
(a) El estado inicial



(b) Después de expandir Arad



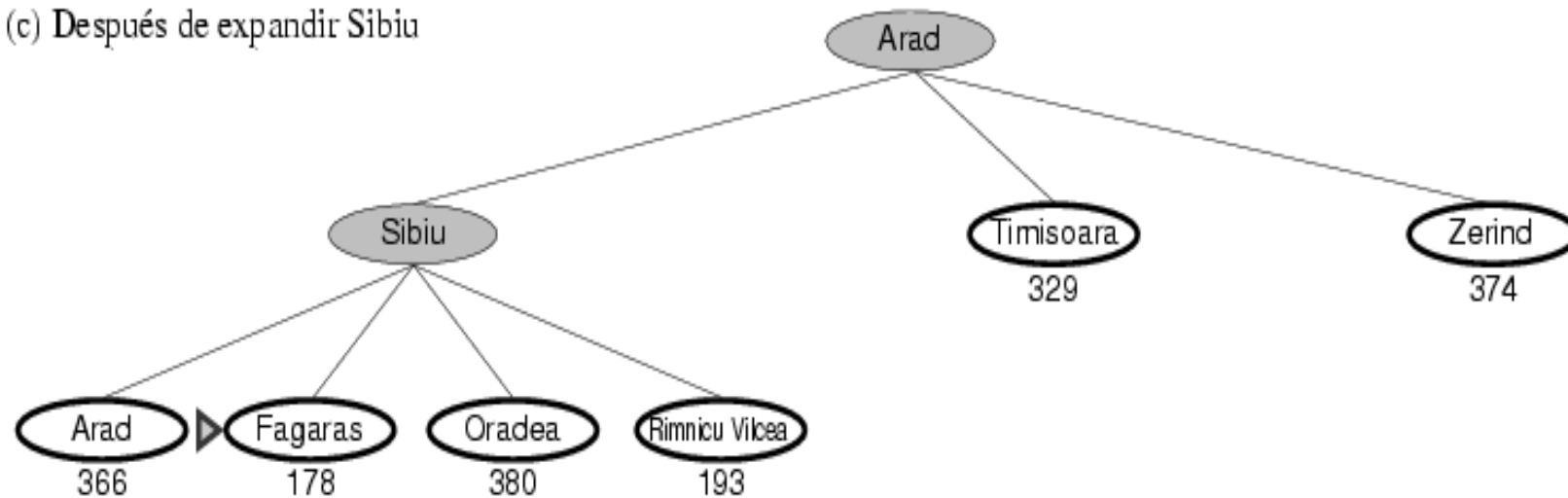
Búsqueda avara: El problema del viajante



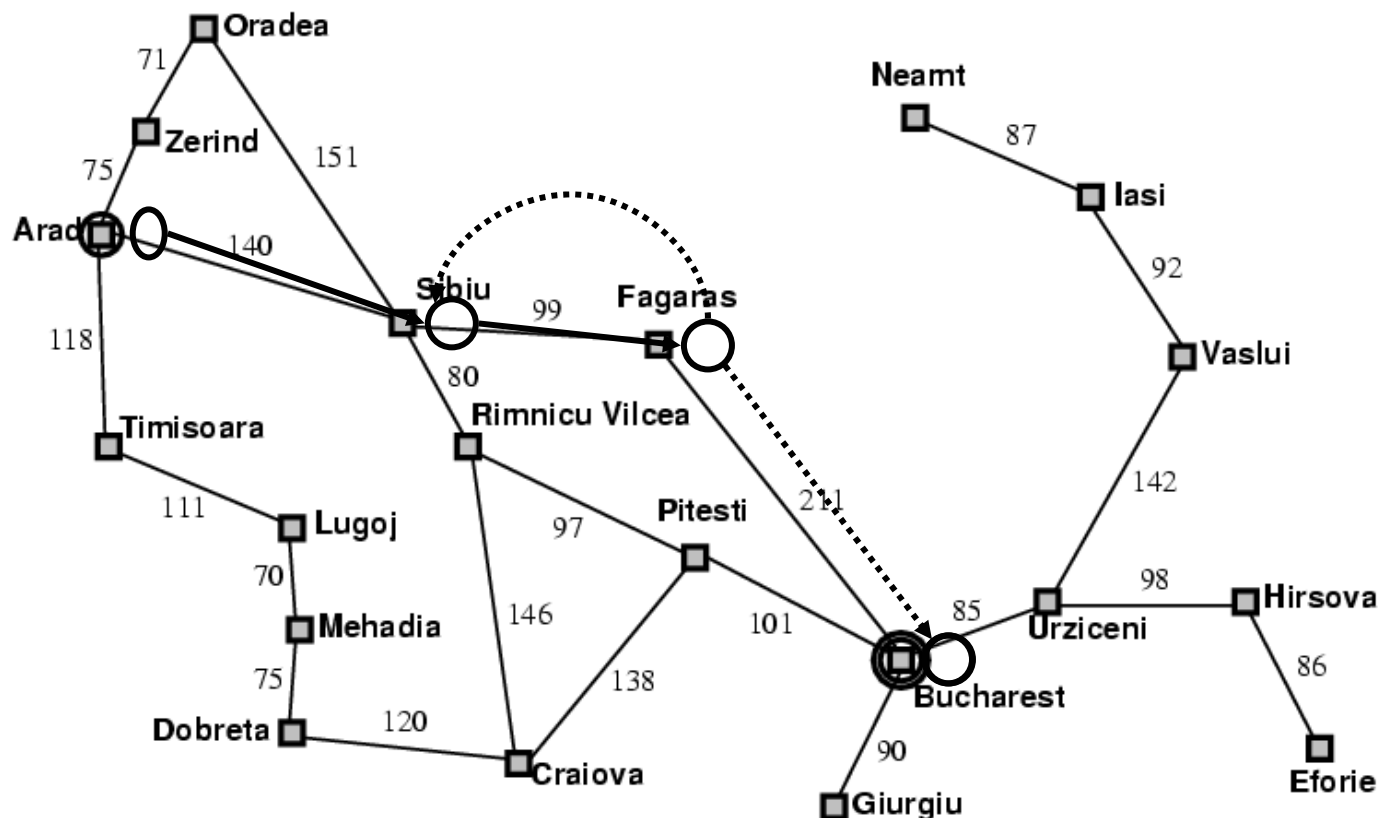
Búsqueda avara: El problema del viajante



(c) Después de expandir Sibiu



Búsqueda avara: El problema del viajante



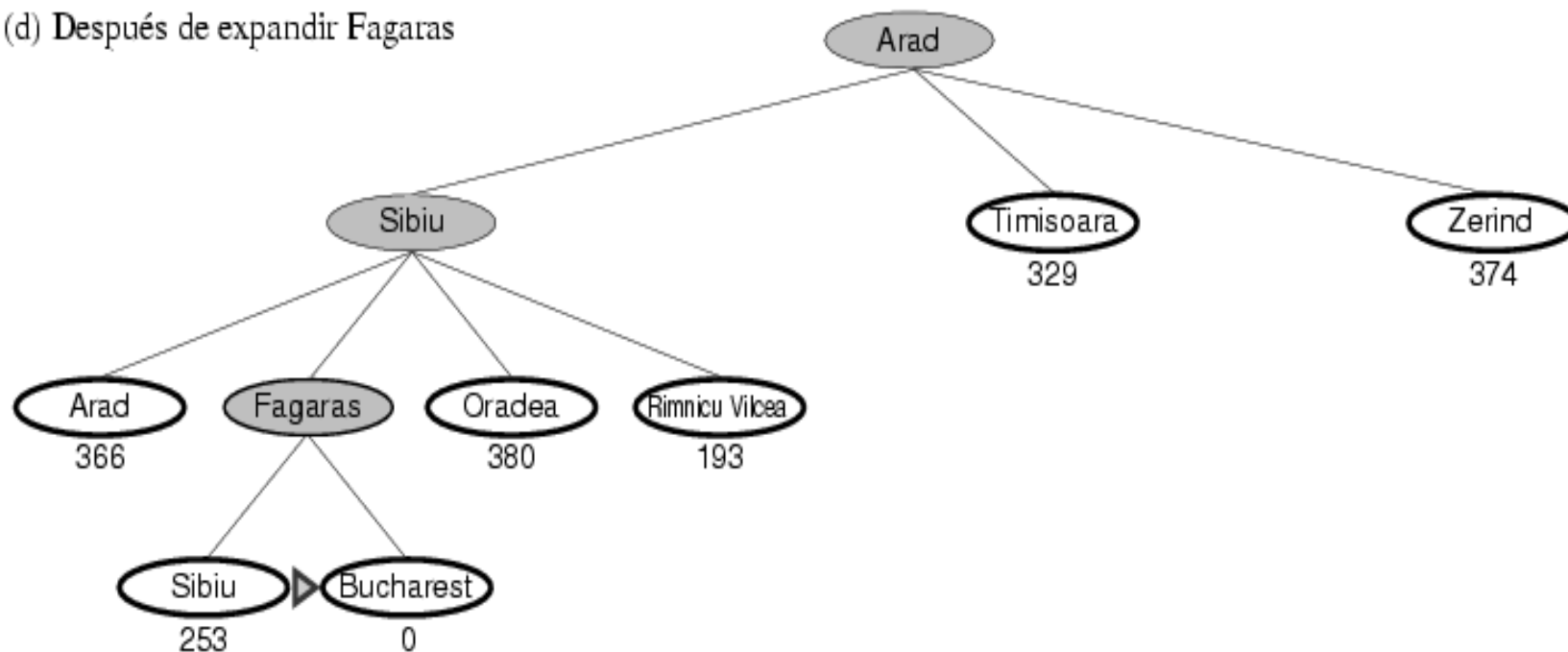
Distancia en línea recta a
Bucarest

Arad	366
Bucarest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Búsqueda avara: El problema del viajante

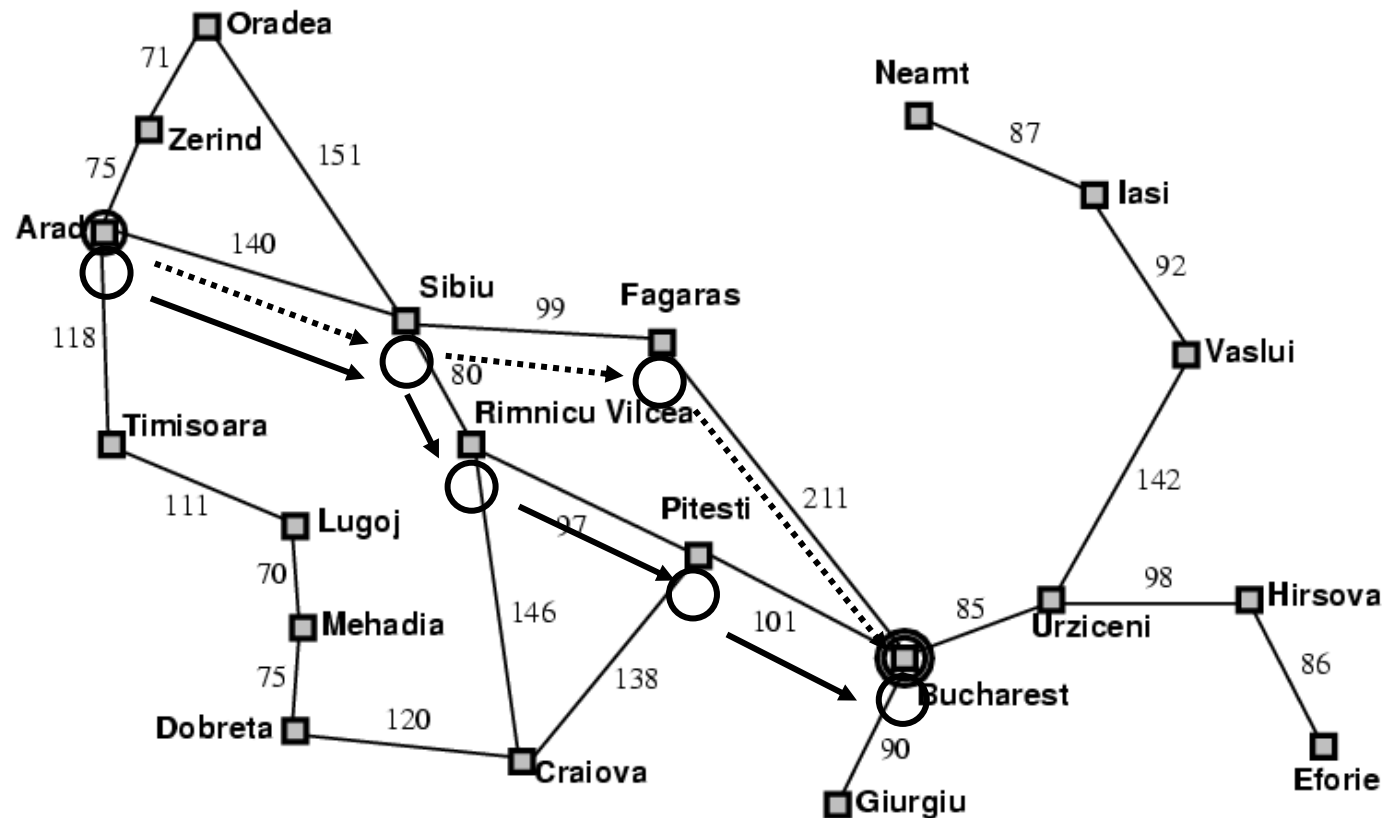


(d) Después de expandir Fagaras



Solución encontrada: Arad – Sibiu – Fagaras – Bucarest = 450
Función de evaluación: Arad → Bucarest = 366

Búsqueda avara: El problema del viajante



.....> Solución encontrada: Arad – Sibiu – Fagaras – Bucarest = 450
—> Ruta óptima: Arad - Sibiu - Rimniu - Pitesti – Bucharest = 418



Búsqueda Avara

- Óptima: No
- Completa: No
 - (puede perderse en bucles—recorrer rutas infinitas)
- Complejidad:
 - Temporal: $O(b^m)$, siendo m la profundidad máxima del árbol de búsqueda
 - Espacial: $O(b^m)$, mantiene todos los nodos en memoria



Búsqueda A*

- Búsqueda avara:
 - minimizar el coste de alcanzar la meta, $h(n)$
 - no es óptima ni completa.
 - Eficiente
- Búsqueda de coste uniforme:
 - minimizar el coste del camino, $g(n)$
 - es óptima y completa
 - a veces muy ineficiente.
- ¿Por qué no combinamos ambos métodos, en uno óptimo, completo y más eficiente?



Búsqueda A*

- Sean:
 - $g^*(n)$: coste del mejor camino entre el nodo inicial y n
 - $h^*(n)$: coste real del camino de mínimo coste entre el nodo n y la meta
 - $f^*(n) = g^*(n) + h^*(n)$: coste del camino de mínimo coste desde el nodo inicial al meta que pasa por el nodo n .
 - Para el nodo de partida, se cumplirá que el valor de f^* es igual a h^* .
- Sin embargo utilizaremos estimaciones de estas funciones
 - $g(n)$: coste del mejor camino para alcanzar el nodo n que el algoritmo ha encontrado hasta el momento
 - $h(n)$ o heurística: estimación de $h^*(n)$, y representa el coste estimado de la ruta menos costosa que parte de n y llega a la meta
 - $f(n) = g(n) + h(n)$ o función de evaluación: coste estimado de la solución más barata, que pasa por n .



Búsqueda A*

- El algoritmo A* intentará probar primero aquellos nodos para los cuales existe un camino que los une al nodo inicial, cuyo valor de f sea el más bajo.
- Para implementarlo, la lista de nodos abiertos se ordena en base al valor de la función de evaluación f .
- Es óptima y completa si la heurística es **admisible**, es decir, si nunca sobreestima el coste real de alcanzar la meta:
 - $\forall n, h(n) \leq h^*(n)$, siendo n cualquier nodo
- Problema del viajante: La distancia en línea recta es una heurística admisible puesto que es la distancia más corta entre dos puntos y nunca podrá sobreestimar.



Búsqueda A*

(a) The initial state





Búsqueda A*

(b) After expanding Arad





Búsqueda A*

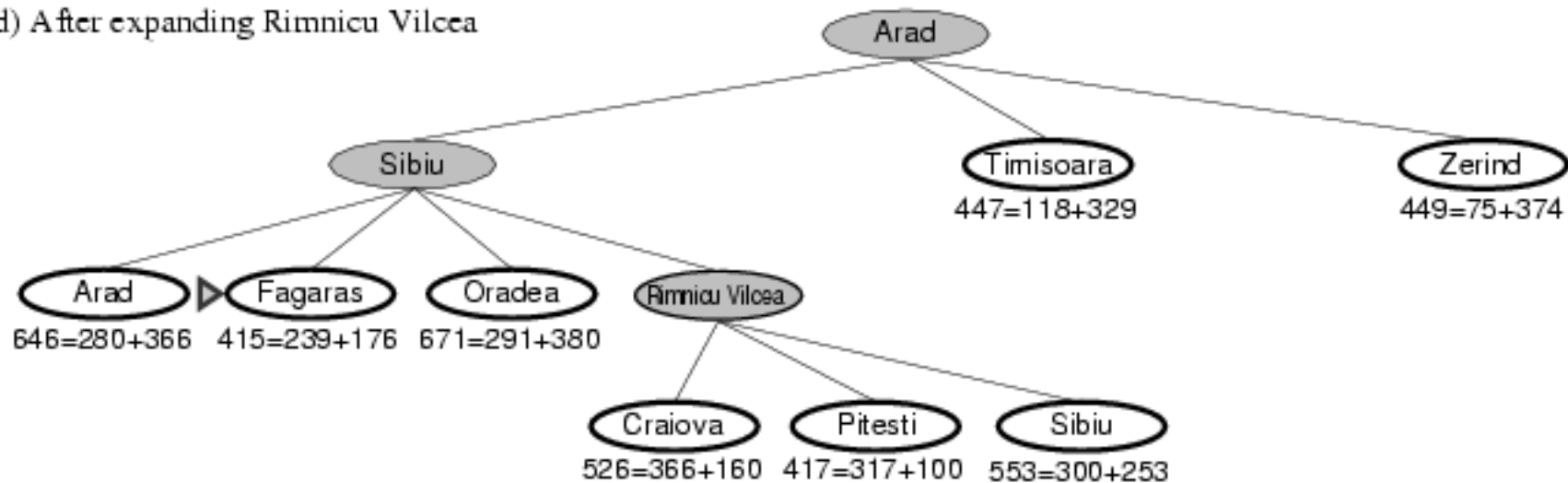
(c) After expanding Sibiu





Búsqueda A*

(d) After expanding Rimnicu Vilcea





e) After expanding Fagaras

```
graph TD; Arad --> Sibiu; Arad --> Timisoara; Arad --> Zerind; Sibiu --> Arad; Sibiu --> Fagaras; Sibiu --> Oradea; Sibiu --> Rimnicu_Vilcea; Fagaras --> Sibiu; Fagaras --> Bucharest; Rimnicu_Vilcea --> Craiova; Rimnicu_Vilcea --> Pitesti; Rimnicu_Vilcea --> Sibiu; style Arad fill:#d3d3d3; style Sibiu fill:#d3d3d3; style Timisoara fill:#fff,stroke:#000; style Zerind fill:#fff,stroke:#000; style Fagaras fill:#d3d3d3; style Oradea fill:#fff,stroke:#000; style Rimnicu_Vilcea fill:#d3d3d3; style Craiova fill:#fff,stroke:#000; style Pitesti fill:#fff,stroke:#000; style Sibiu2 fill:#fff,stroke:#000; style Bucharest fill:#fff,stroke:#000;
```

Arad

Sibiu

Timisoara
 $447=118+329$

Zerind
 $449=75+374$

Arad
 $646=280+366$

Fagaras

Oradea
 $671=291+380$

Rimnicu Vilcea

Sibiu
 $591=338+253$

Bucharest
 $450=450+0$

Craiova
 $526=366+160$

Pitesti
 $417=317+100$

Sibiu
 $553=300+253$



c) After expanding Pitesti

```
graph TD; Arad --> Sibiu; Arad --> Timisoara; Arad --> Zerind; Sibiu --> Arad; Sibiu --> Fagaras; Sibiu --> Oradea; Sibiu --> Rimnicu_Vilcea; Fagaras --> Sibiu; Fagaras --> Bucharest; Rimnicu_Vilcea --> Craiova; Rimnicu_Vilcea --> Pitesti; Rimnicu_Vilcea --> Sibiu; Pitesti --> Bucharest; Pitesti --> Craiova; Pitesti --> Rimnicu_Vilcea; style Arad fill:#d3d3d3; style Sibiu fill:#d3d3d3; style Fagaras fill:#d3d3d3; style Pitesti fill:#d3d3d3; style Bucharest fill:#d3d3d3; style Bucharest2 fill:#d3d3d3; style Craiova2 fill:#d3d3d3; style Rimnicu_Vilcea2 fill:#d3d3d3;
```

Arad

Sibiu

Timisoara
 $447=118+329$

Zerind
 $449=75+374$

Arad
 $646=280+366$

Fagaras

Oradea
 $671=291+380$

Rimnicu Vilcea

Sibiu
 $591=338+253$

Bucharest
 $450=450+0$

Craiova
 $526=366+160$

Pitesti

Sibiu
 $553=300+253$

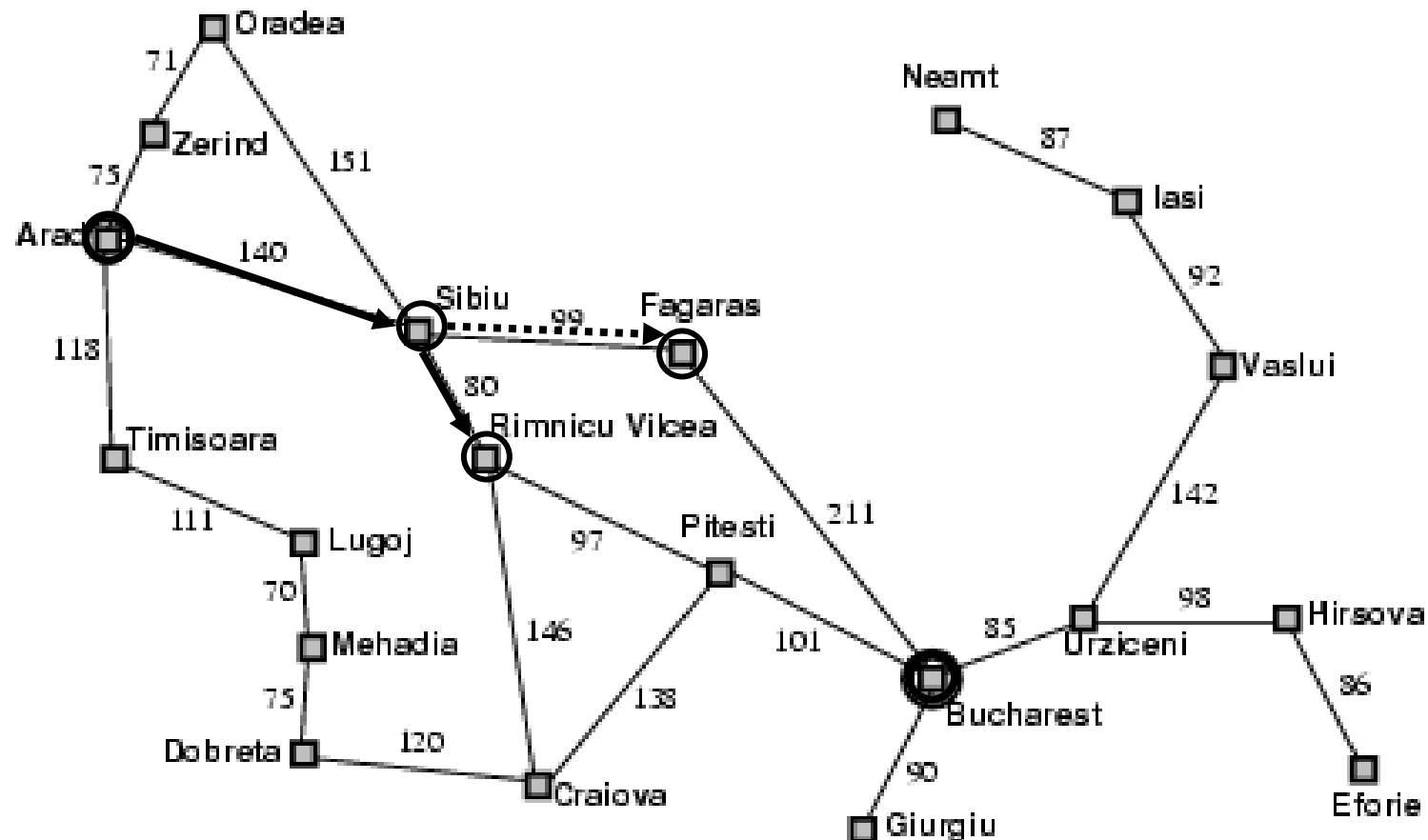
Bucharest
 $418=418+0$

Craiova
 $615=455+160$

Rimnicu Vilcea
 $607=414+193$



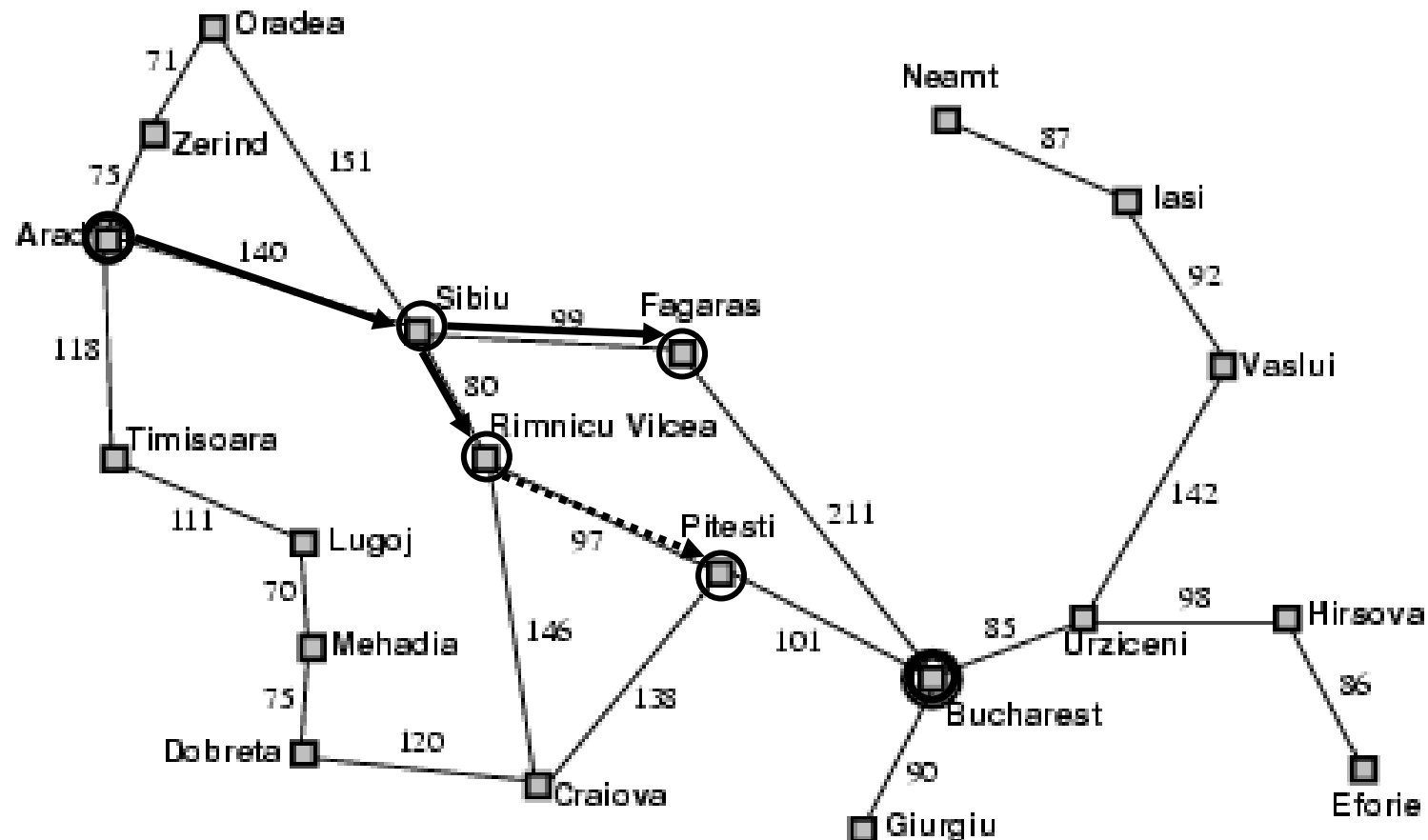
Búsqueda A*



Se expande **Rimnicu** ($f=(140+80)+193=413$) frente a **Fagaras** ($f=(140+99)+176=415$)



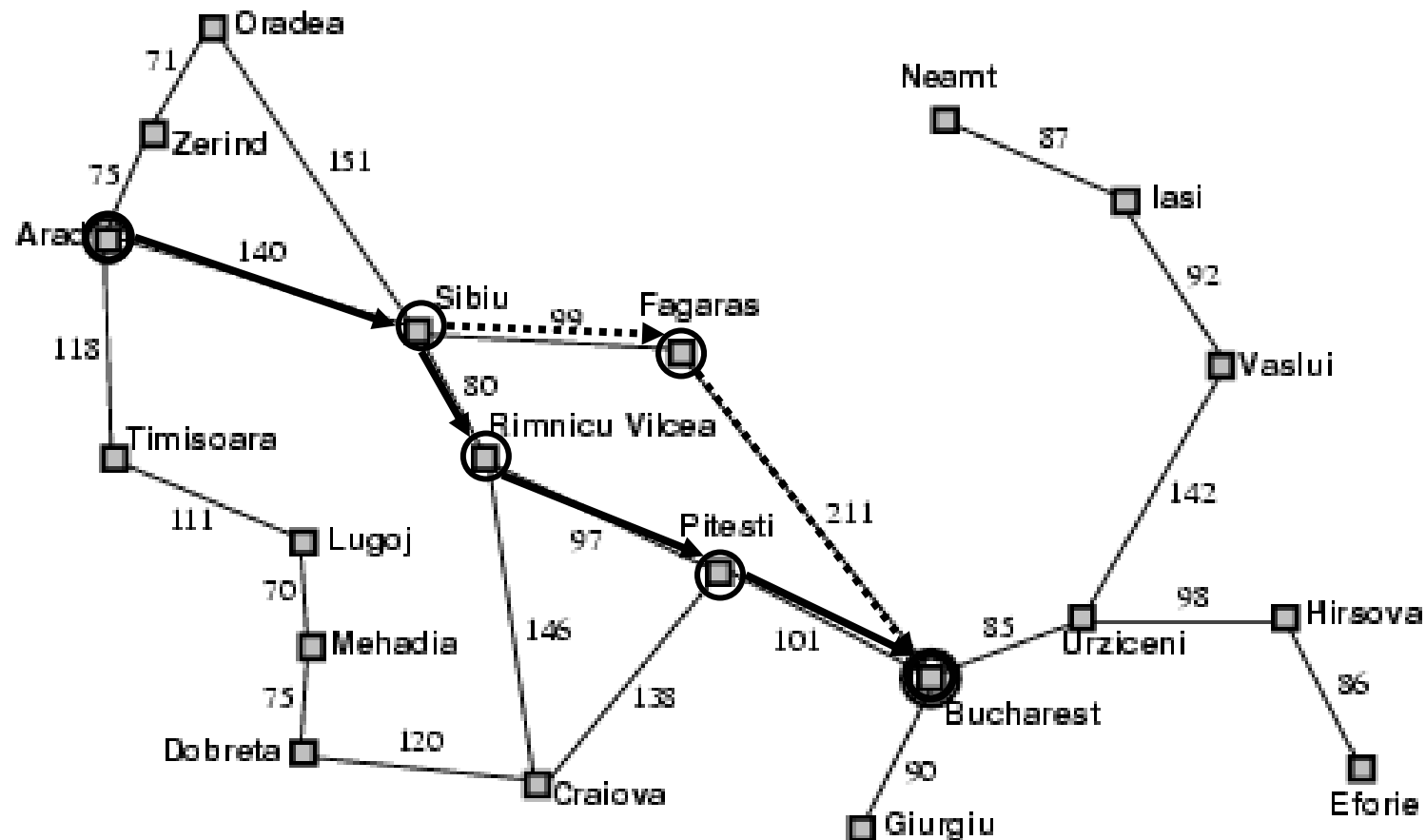
Búsqueda A*



Se expande **Fagaras** ($f=(140+99)+176=415$) frente a **Pitesti** ($f=(140+80+97)+100=417$)



Búsqueda A*



A través de **Fagaras** $f(\text{Bucharest}) = 140 + 99 + 211 = 450$

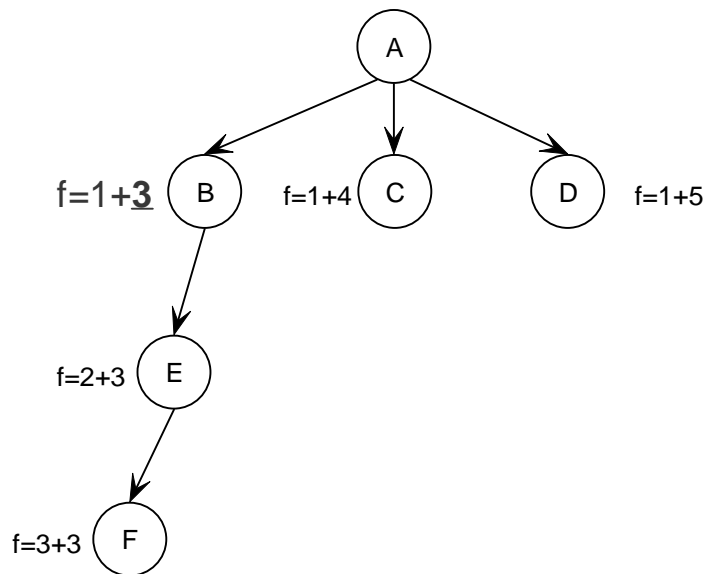
A través de **Pitesti** $f(\text{Bucharest}) = 140 + 80 + 97 + 101 = 418$



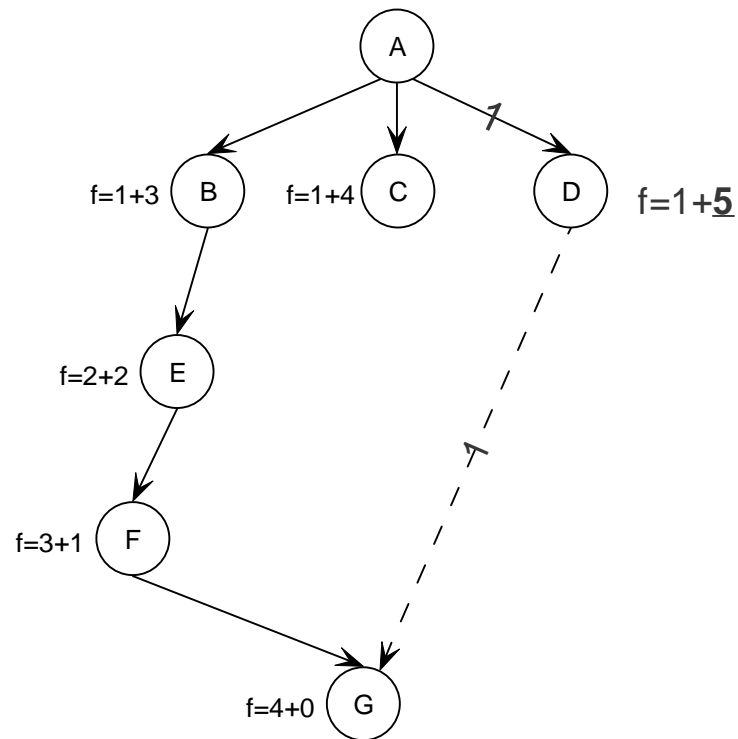
Búsqueda A*

- A* puede utilizarse para encontrar:
 - un camino de coste total mínimo
 - el camino más rápido posible $g=1$ (menor nº pasos)
- A* se comporta como
 - Anchura si:
 - g se incrementa en 1 y $h = 0$
 - los nodos con igual f se ordenan de menos a más reciente.
 - Profundidad si:
 - $g = 0$ y $h = 0$
 - Los nodos se ordenan de más a menos reciente

Búsqueda A*: Efecto de la sobreestimación



h subestima a h^*



h sobrestima a h^*

Búsqueda A*: Admisibilidad de una heurística



- Proposición:
 - Si h es admisible, es decir, $\forall n, h(n) \leq h^*(n)$, entonces la búsqueda A* es óptima.
- Corolario:
 - Si h raramente sobreestima a h^* más de δ , entonces al algoritmo A* raramente encontrará una solución cuyo coste sea más que δ mayor que el coste de la solución óptima.

Admisibilidad: Consecuencias

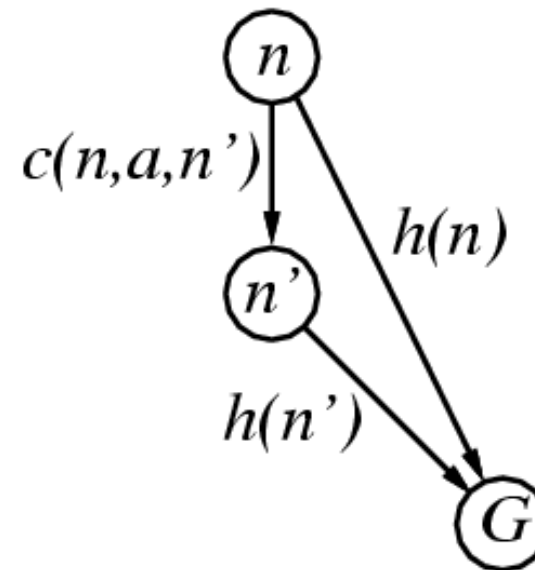


- La búsqueda basada en árbol no seleccionará soluciones subóptimas.
- La búsqueda basada en grafo es posible que seleccione soluciones subóptimas al descartar un camino óptimo a un nodo repetido si no ha sido el primero en ser generado.
- Soluciones:
 - Descartar el camino más costoso de los dos que llegan al mismo nodo.
 - Asegurar que el camino óptimo a cualquier estado repetido es siempre el primero que se encuentra → CONSISTENCIA de una heurística

Consistencia de una heurística



- Una función heurística $h(n)$ es consistente si, para cada nodo n y cada sucesor n' de n generado por cualquier acción a , el coste estimado de alcanzar la meta desde n no es mayor que la suma entre el coste de llegar a n' y el coste estimado de alcanzar la meta desde n'
- O sea, $h(n) \leq c(n, a, n') + h(n')$, o lo que es lo mismo, $h(n) - h(n') \leq c(n, a, n')$
- Entonces, el incremento de valor de la heurística es como mucho el coste real $c(n, a, n')$
- Se conoce como *desigualdad del triángulo*: cada lado no puede tener una longitud mayor que la suma de los otros dos.
- Toda heurística consistente es admisible



Consistencia: Consecuencias

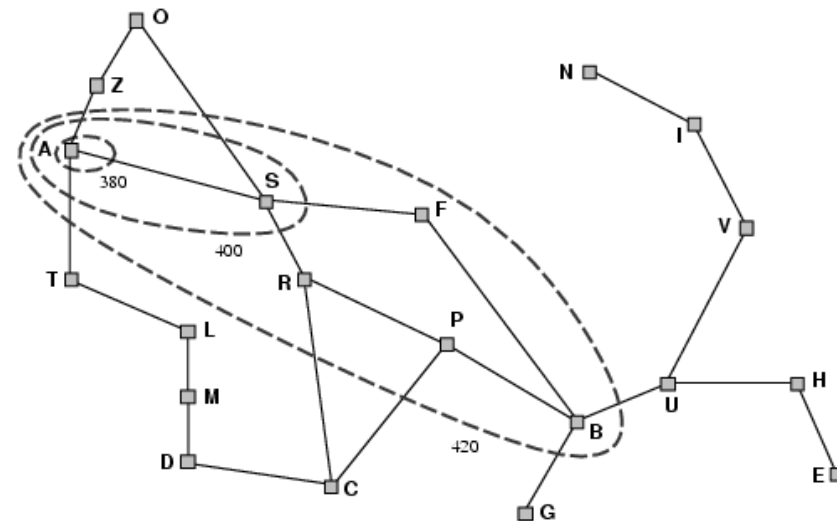


- Si h es consistente, entonces los valores de f a lo largo del camino son no decrecientes \rightarrow Monotonía.
- El algoritmo A^* basado en grafos es óptimo si la función h es consistente. Es decir, si selecciona un nodo n para su expansión, entonces $g(n) = g^*(n)$, siendo $g^*(n)$ la función que devuelve el coste óptimo.
- El primer nodo meta seleccionado para expansión deberá ser una solución óptima, ya que todos los nodos posteriores serán, al menos, tan costosos.



Monotonía de f

- Contornos en el espacio de estados para el problema del viajante.
- Mapa de Rumanía con Arad como estado inicial mostrando contornos en el espacio de estados.
- Los nodos dentro de un contorno dado tienen costes f menores o iguales que el valor del contorno ($f=380$, $f=400$, $f=420$)





Búsqueda A^*

- Óptima: si, A^* es óptimamente eficiente
 - Ningún otro algoritmo óptimo garantiza expandir menos nodos
- Completa:
 - A^* expande nodos en orden de f creciente, hasta realizar una última banda de expansión que le lleve a un estado meta.
 - Siempre que todos los costes exceden algún n° finito ε y si b es finito.
- Complejidad temporal: Exponencial
- Complejidad espacial: b^d , Exponencial en la longitud de la solución

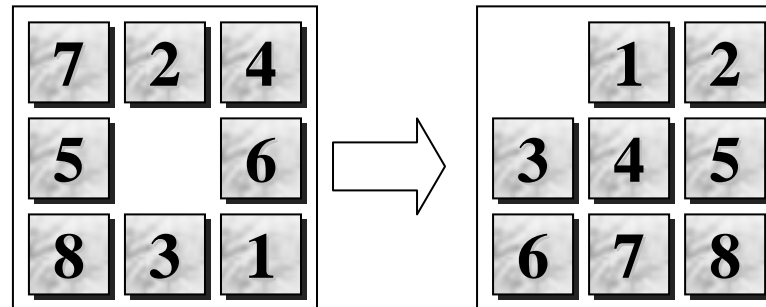


Búsqueda A^*

- El tiempo computacional de A^* no es su desventaja principal.
 - Mantiene todos los nodos generados en memoria
 - Se queda sin espacio antes de quedarse sin tiempo.
 - A^* no es práctico para problemas muy grandes.
- Otros algoritmos de menor coste espacial y temporal son:
 - Búsqueda heurística con memoria acotada.



Generación de heurísticas



- Operadores: Mover una pieza $\uparrow \downarrow \leftarrow \rightarrow$
- Coste de una solución = número de movimientos necesarios para llegar del estado de partida al estado meta
- Solución típica:
 - 22 pasos,
 - factor de ramificación=3,
 - búsqueda exhaustiva: 3^{22} , aprox. $3,1 \times 10^{10}$ estados
 - Con control de nodos repetidos--- 181. 440 estados distintos
 - Para el puzzle 15 serían 10^{13}
 - Aconsejable buscar una función heurística

Diseño de heurísticas admisibles



- Heurísticas basadas en la relajación o abstracción del problema:
 - La solución a un problema se construye aplicando una serie de operadores elementales sujetos a una serie de restricciones.
 - La idea es conseguir una abstracción de un problema transformándolo en otro más sencillo eliminando ciertas restricciones del problema original ("relajándolo").
 - La función de coste de la ruta en el problema relajado es una heurística admisible y consistente en el problema original
 - Cuantas más restricciones tengamos en cuenta, más precisa será la heurística
 - La heurística así diseñada será admisible y consistente

8-puzzle: Relajación del problema

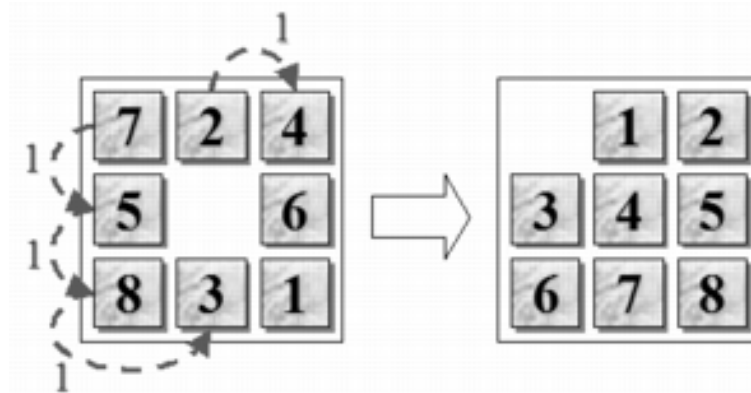


- Operador: Mover una pieza de A a B
- Restricciones:
 - si A es adyacente (vertical u horizontal) a B y
 - B está vacío.
- Problemas relajados:
 - a) Se puede mover una pieza de A a B, si A es adyacente a B (solapamiento)
 - b) Se puede mover una pieza de A a B, si B está vacío (teletransporte)
 - c) Se puede mover una pieza de A a B (solapamiento y teletransporte)



Distancia Manhattan

- Se puede mover una pieza de A a B, si A es adyacente a B



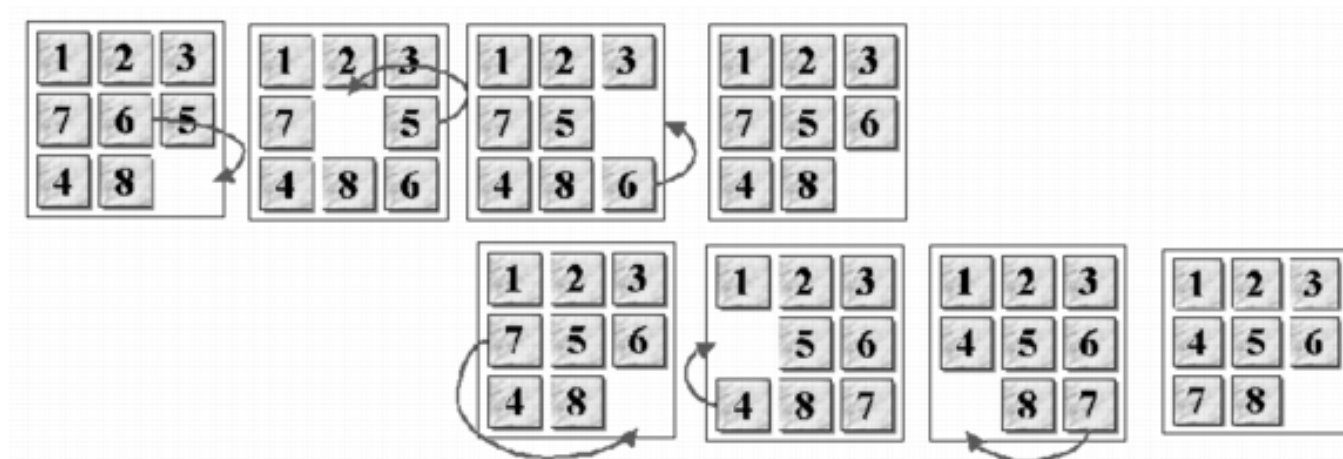
$$H_1 = \text{Movimientos}(\boxed{7}) + \text{Movimientos}(\boxed{2}) + \dots = 3 + 1 + \dots = 18$$

Coste de la solución óptima = 26



Distancia Gaschnig

- Se puede mover una pieza de A a B, si B está vacío



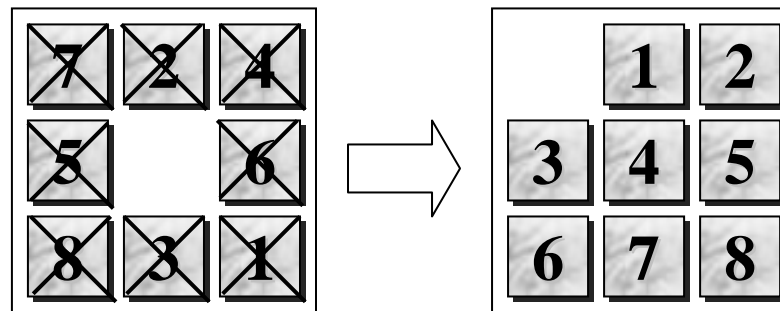
$$H_2 = 6$$


Coste de la solución óptima = 18



Piezas mal colocadas

- Se puede mover una pieza de A a B



 Mal colocada

$$H_3 = 8$$

Coste de la solución óptima = 26

¿Cuál es la mejor heurística?



- Heurística Manhattan (h_1) domina al resto
 - Para todo nodo n se cumple que $h_1(n) \geq h_2(n)$ y $h_1(n) \geq h_3(n)$. h_1 está *más informada*.
 - A^* con h_1 expandirá menos nodos en promedio
 - $d=14$ $A^*(h_1)=113$ nodos $A^*(h_2)= 539$ nodos
 - $d=24$ $A^*(h_1)=1641$ nodos $A^*(h_2)= 39135$ nodos
- Si ninguna heurística domina al resto,
$$h(x) = \max\{h_1(x), h_2(x), \dots, h_n(x)\}$$
- Objetivo: Diseñar heurísticas cercanas a h^* , admisibles y computables eficientemente

Búsqueda conducida por agendas



- Listas ordenada de tareas que puede realizar un sistema
- Cada tarea lleva asociada:
 - una lista de razones o motivos por los que dicha tarea es interesante (justificaciones)
 - un valor que representa el peso total de la evidencia que sugiere que la tarea es útil
- No todas las justificaciones de una misma tarea tienen por qué pesar lo mismo
- La búsqueda conducida por agendas es un método de búsqueda por el mejor nodo en el que:
 - Se debe elegir la mejor tarea de la agenda
 - Se debe ejecutar la tarea seleccionada asignando para ello recursos. Si se crean subtareas se insertan en su lugar correspondiente

Búsqueda conducida por agendas



Función Búsqueda–agenda

hacer mientras no se alcance un estado objetivo o la agenda este vacía

Identificar la mejor tarea de la agenda y ejecutarla

Si se han generado nuevas tareas (subtareas) **para cada** una de ellas

hacer

Si no estaba ya en la agenda **añadir** la nueva tarea

Si estaba ya en la agenda

hacer

Si no tiene la misma justificación **añadir** la nueva justificación

Si se ha añadido una tarea o una justificación

calcular el peso asignado combinando la evidencia de todas sus justificaciones

reorganizar la agenda



Búsqueda local

- Búsqueda no informada y búsqueda por el mejor:
 - Exploran sistemáticamente el espacio de búsqueda conservando uno o más caminos en memoria
 - Registran qué alternativas han sido exploradas en cada punto de ese camino y cuáles no
 - Cuando encuentran una meta, el camino a la meta constituye una solución al problema.
- En muchos problemas, el camino es irrelevante:
 - 8 reinas
 - Diseño de circuitos integrados
 - Programación del trabajo en una empresa
 - Optimización de redes de telecomunicaciones
 -



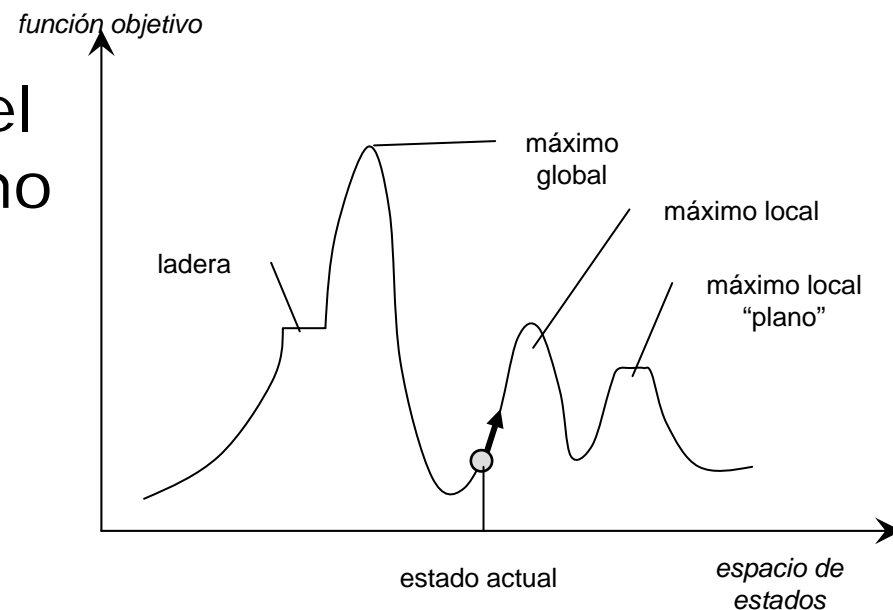
Búsqueda local

- Usan un único estado actual (en lugar de múltiples caminos) y se mueven sólo a sus vecinos (el camino a la meta es irrelevante).
- Los caminos no se retienen.
- No son sistemáticos, pero tienen dos ventajas:
 - Poca memoria, por lo general una cantidad constante.
 - Encuentran soluciones razonables en espacios de estados grandes o infinitos (continuos), aptos para problemas de optimización.

Algoritmo de escalada ("Hill-climbing")



- Cada punto representa un estado y un valor de la función objetivo
- Se trata de encontrar el punto más alto, máximo global*

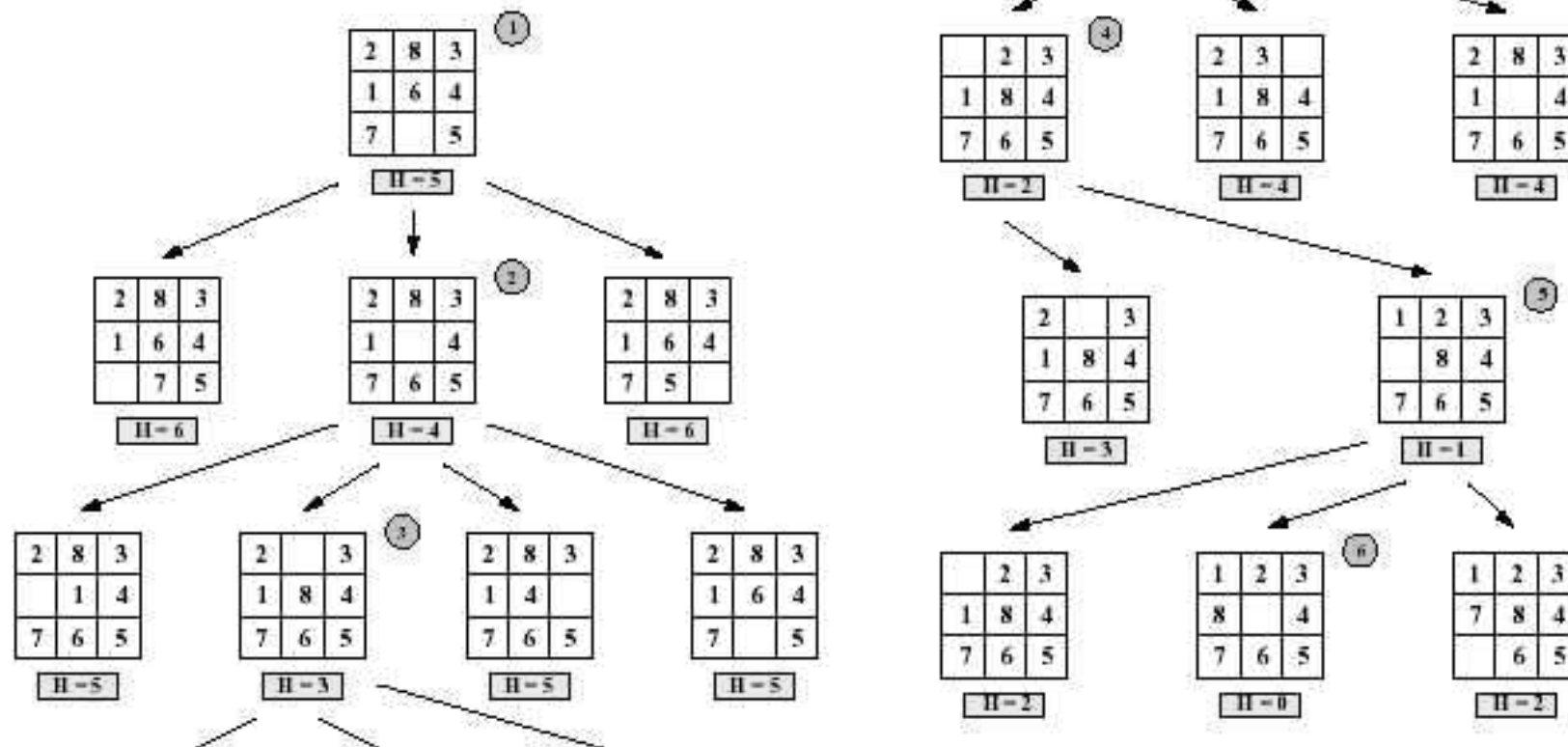


* A efectos prácticos, consideraremos que el algoritmo intenta maximizar una función objetivo; en el caso de función heurística, en realidad intenta minimizar la función heurística



122

Algoritmo de escalada: Ejemplo



h = Suma de las distancias Manhattan

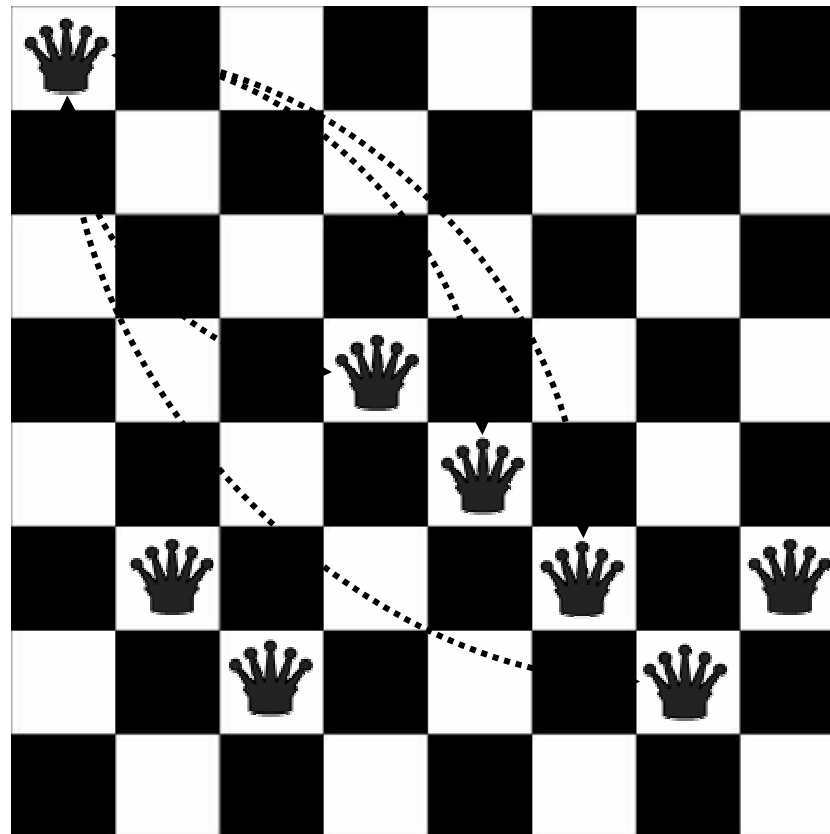
El problema de las 8 reinas



- Operador: mover una reina dentro de la columna
- Heurística = número de pares de reinas que se atacan
- En esta situación, $h=12$

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♠	13	16	13	16
♠	14	17	15	♠	14	16	16
17	♠	16	18	15	♠	15	♠
18	14	♠	15	15	14	♠	16
14	14	13	17	12	14	12	18

El problema de las 8 reinas

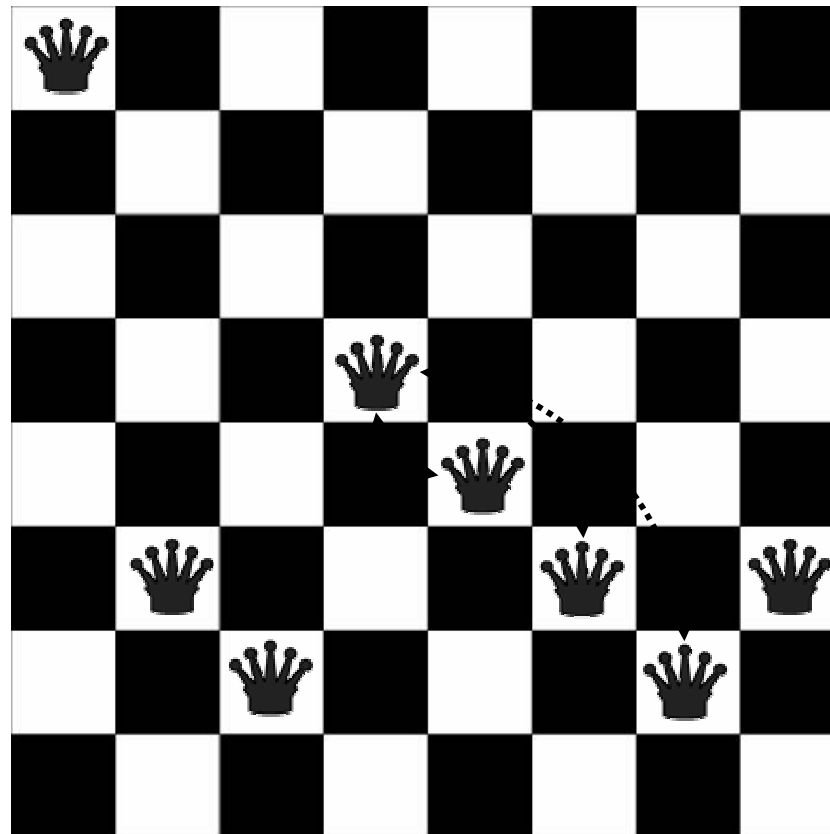


En esta situación, $h=18$

$h = \text{número de pares de reinas que se atacan} = 4 + \dots$

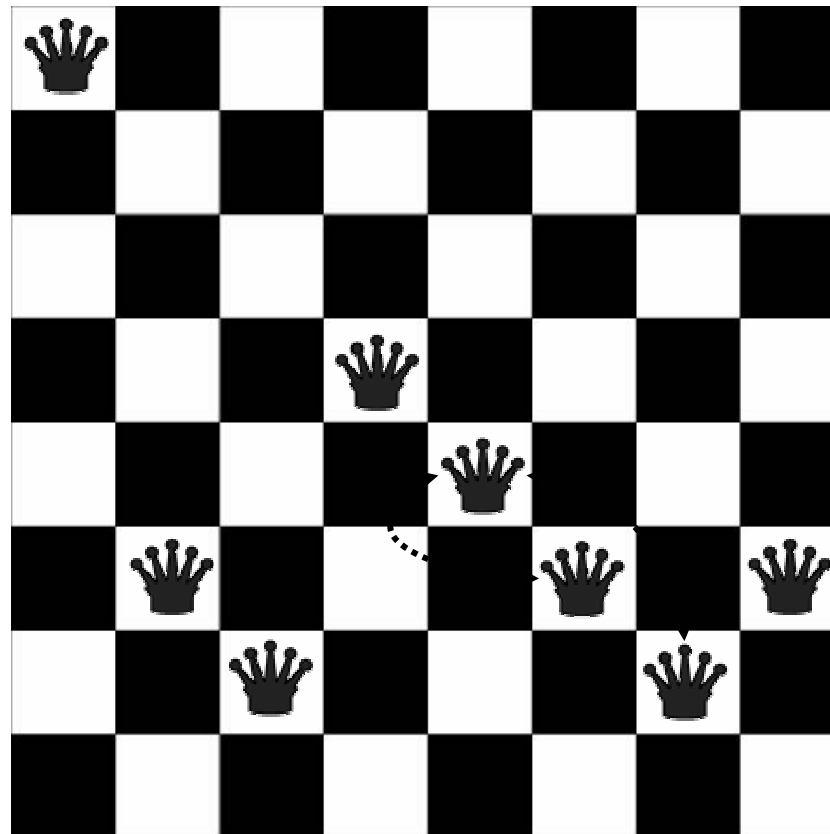
INTELIGENCIA ARTIFICIAL

El problema de las 8 reinas



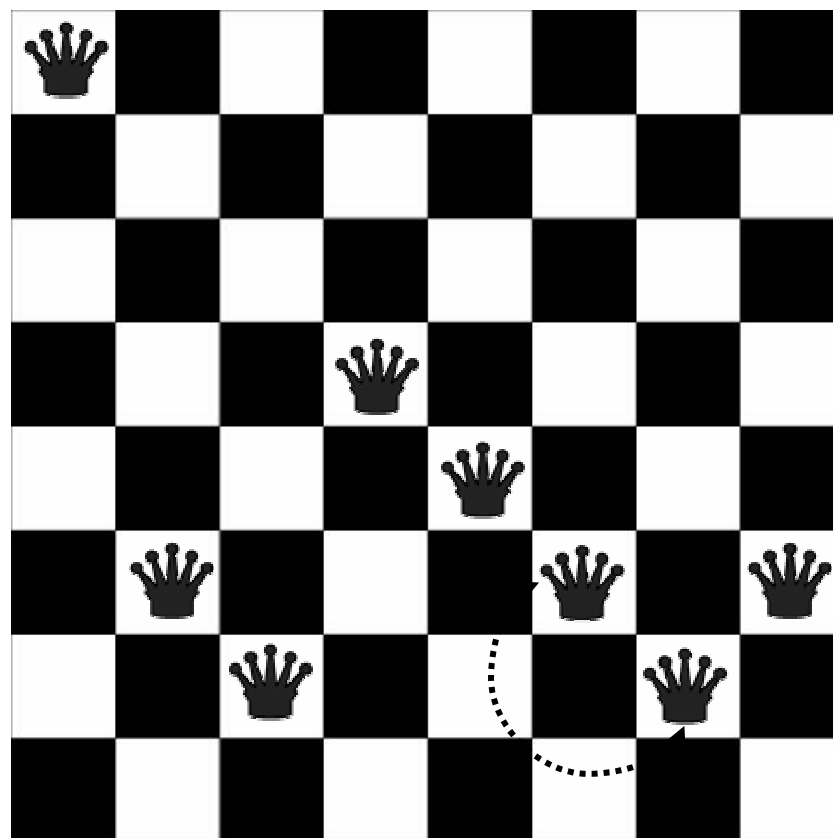
$h = \text{número de pares de reinas que se atacan} = 4 + 3 + \dots$

El problema de las 8 reinas



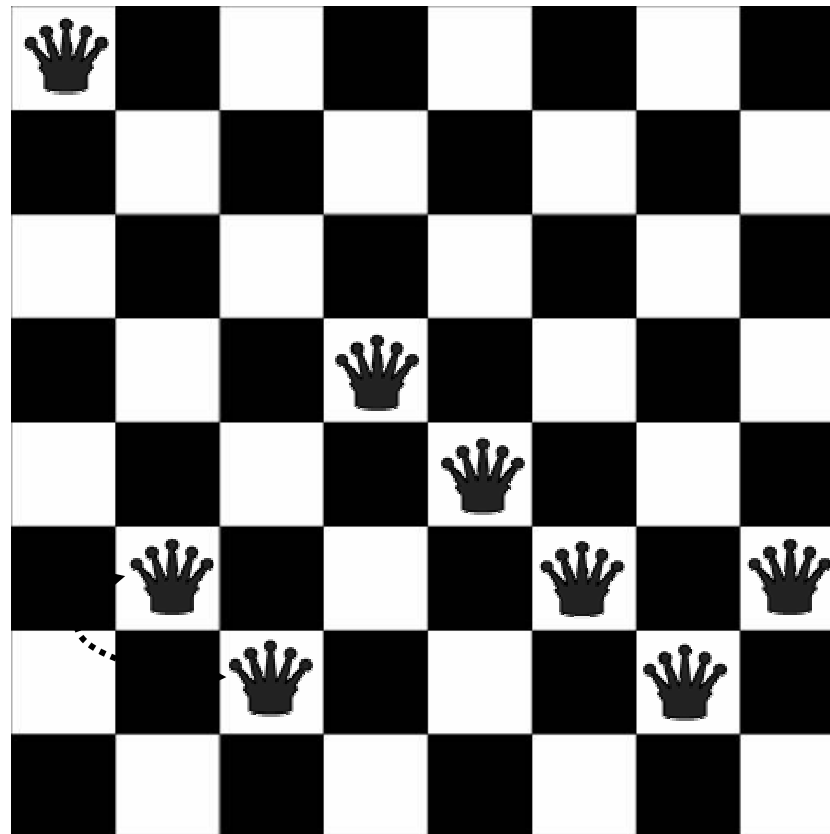
$h = \text{número de pares de reinas que se atacan} = 7 + 2 + \dots$

El problema de las 8 reinas



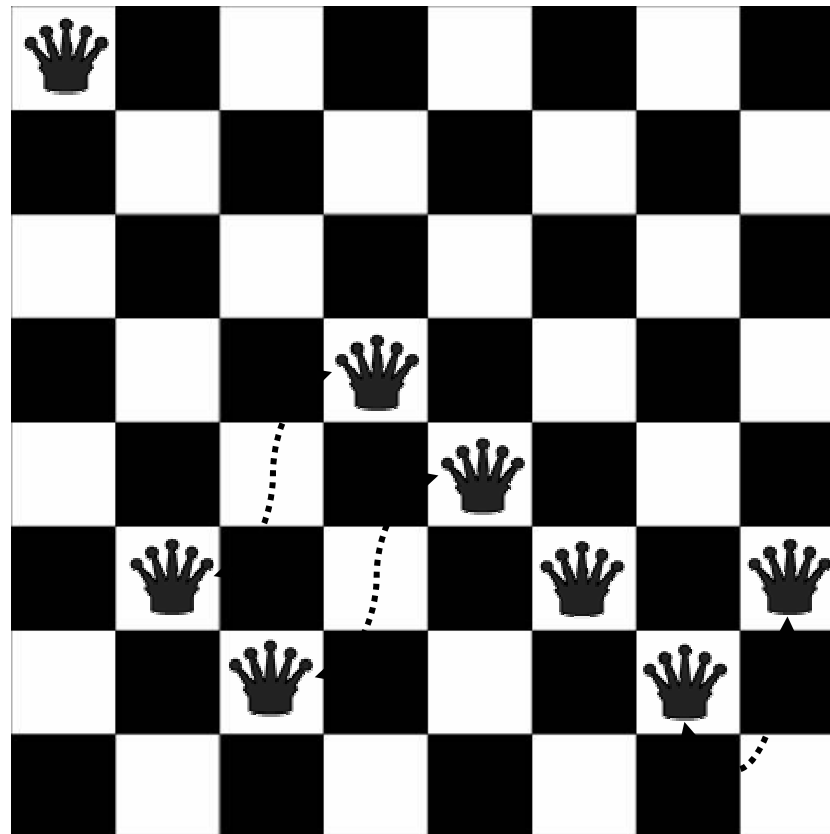
$h = \text{número de pares de reinas que se atacan} = 9 + 1 + \dots$

El problema de las 8 reinas



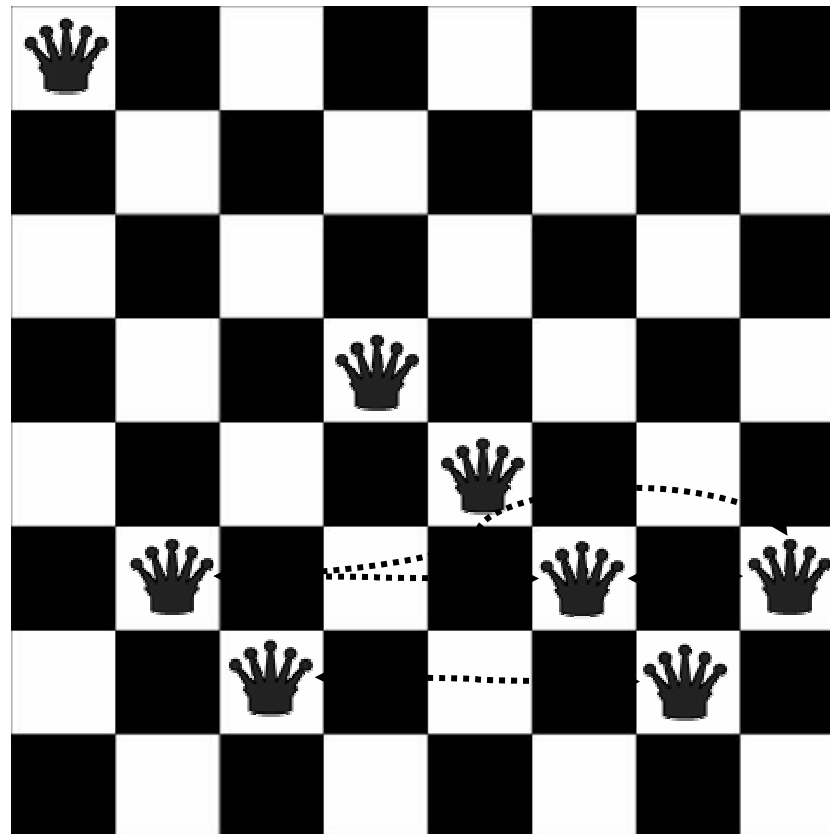
$h = \text{número de pares de reinas que se atacan} = 10 + 1 + \dots$

El problema de las 8 reinas



$h = \text{número de pares de reinas que se atacan} = 11 + 3 + \dots$

El problema de las 8 reinas



$h = \text{número de pares de reinas que se atacan} = 14 + 4 = 18$

Algoritmo de escalada: Inconvenientes

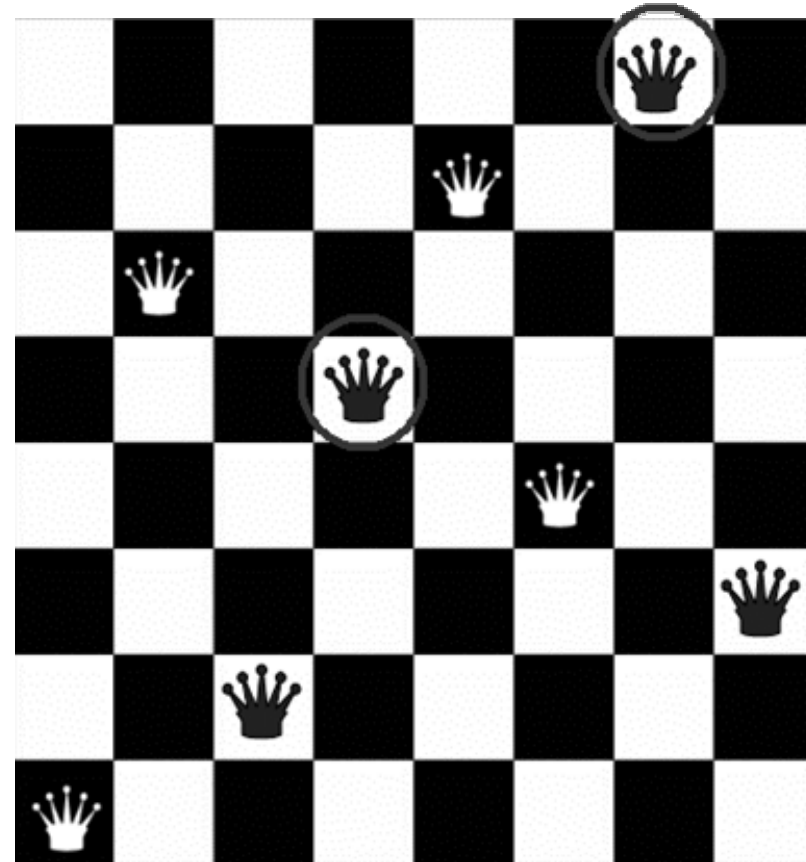


- Máximo Local
 - Estado puntual mejor que cualquiera de sus vecinos, pero peor que otros estados más alejados (máximo global).
Cuando aparecen cerca de la solución final, se llaman estribaciones
- Meseta
 - Región del espacio de estados en la que todos los estados individuales tienen el mismo valor de la función heurística y, por lo tanto, no es posible determinar la mejor dirección para continuar
- Cresta
 - Región del espacio de estados con estados que tienen mejores valores de la función heurística que los de regiones colindantes, pero a los que no podemos llegar mediante transiciones simples



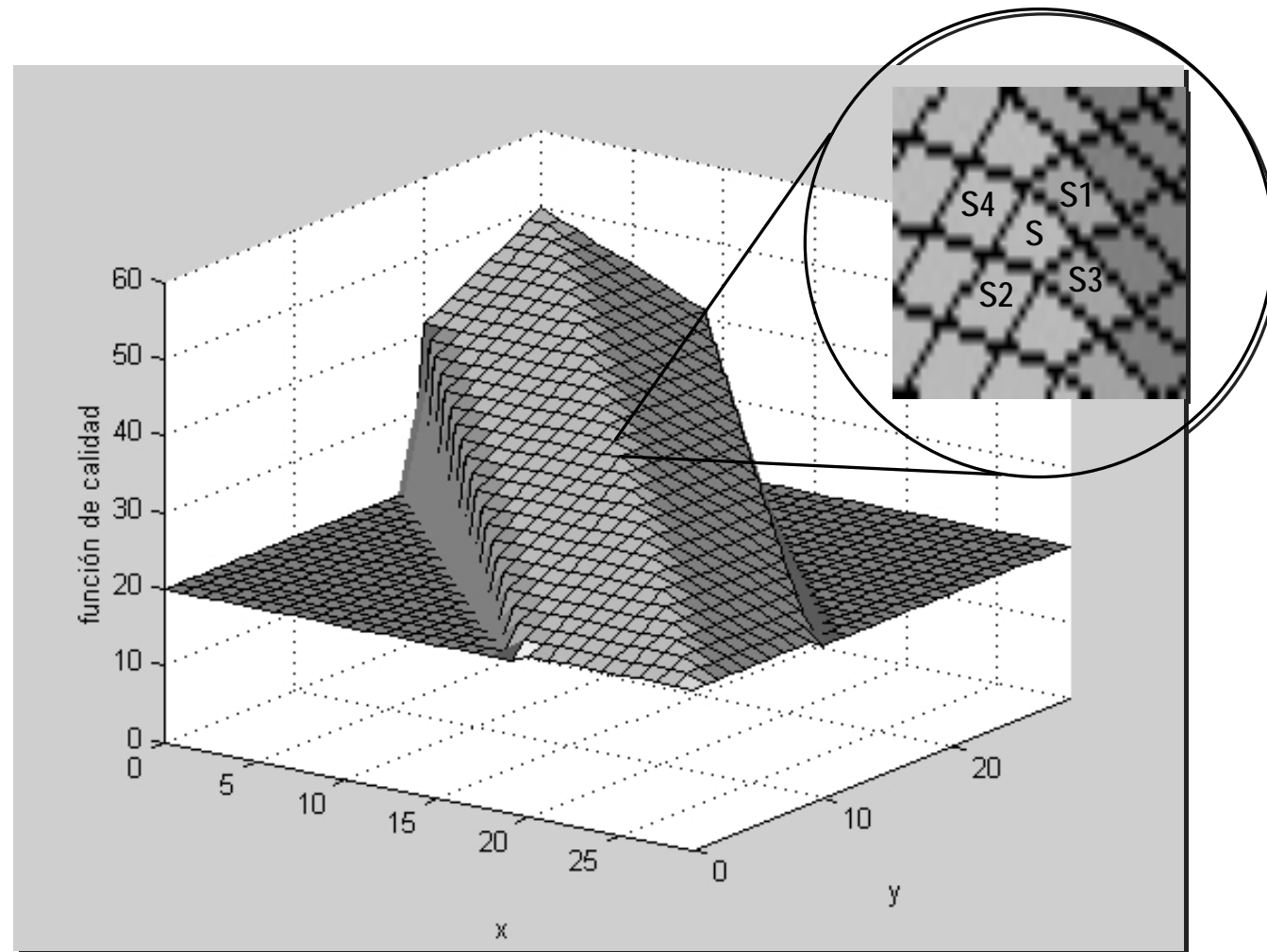
Máximo local

- Operador: mover una reina dentro de la columna
- Heurística = número de pares de reinas que se atacan
- En esta situación, $h=1$, y cualquier movimiento empeora este valor



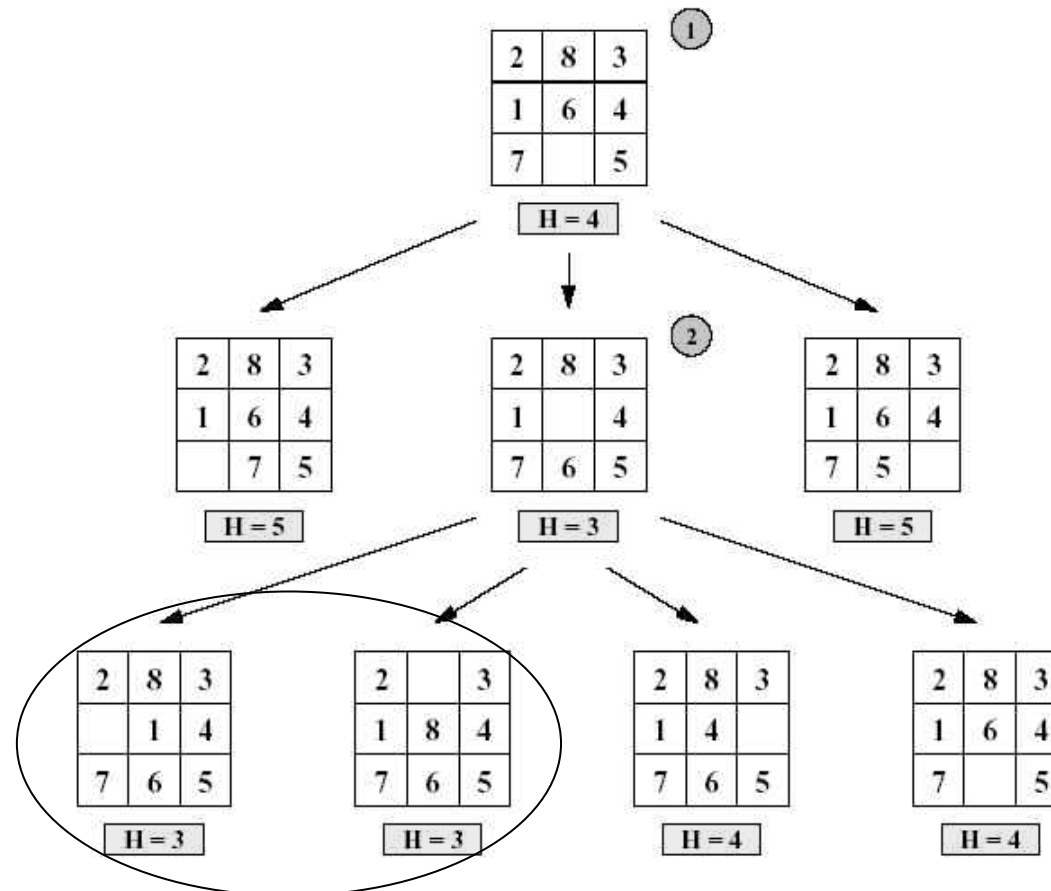


Crestas





Mesetas



h = número de piezas mal colocadas

Algoritmo de escalada: Soluciones



- Para los Máximos Locales
 - Regresar a un nodo previo e intentar una dirección diferente (backtracking)
- Para las Mesetas
 - Realizar un gran salto en el espacio de búsqueda y tratar de alcanzar una región diferente del espacio de estados
- Para las Crestas
 - Aplicar más de un operador antes de realizar la prueba de meta (equivale a moverse en varias direcciones)