Búsqueda

Grupo de Planificación y Aprendizaje (PLG)
Departamento de Informática
Escuela Politécnica Superior
Universidad Carlos III de Madrid

22 de diciembre de 2008

Busqueda heurística

Búsqueda

Grupo de Planificación y Aprendizaje (PLG)
Departamento de Informática
Escuela Politécnica Superior
Universidad Carlos III de Madrid

22 de diciembre de 2008

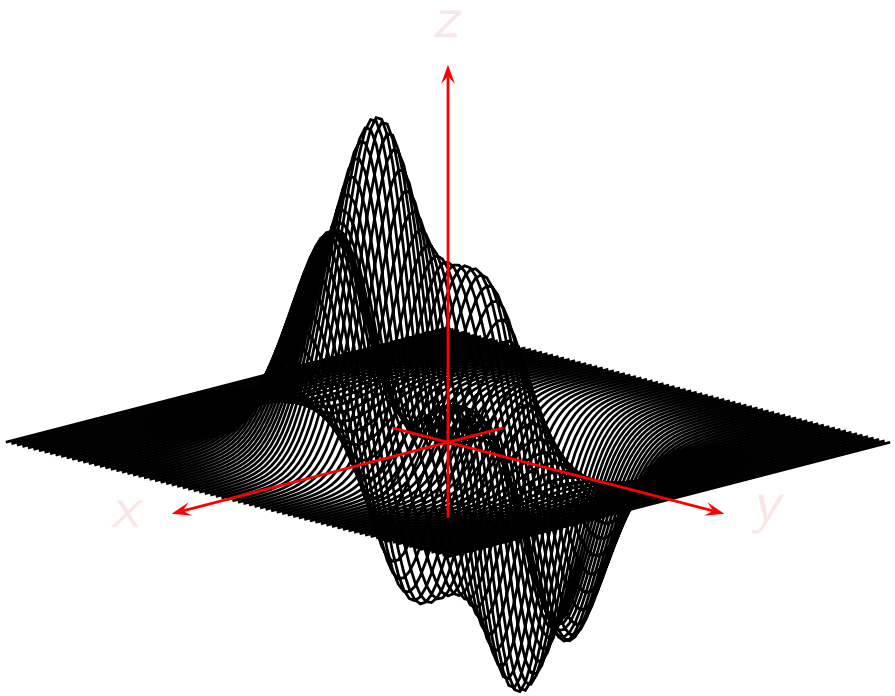
En Esta Sección:

Heurísticas

- Si no se tiene conocimiento $\longrightarrow$ búsqueda sin información
- Si se tiene conocimiento perfecto $\longrightarrow$ algoritmo exacto
- En la mayor parte de los problemas que resuelven los humanos, se está en posiciones intermedias
- Heurística: (del griego "*heurisko*" ($\varepsilon\acute{\upsilon}\rho\iota\sigma\kappa\omega$): "**yo encuentro**") conocimiento parcial sobre un problema/dominio que permite resolver problemas eficientemente en ese problema/dominio
- Representación de las heurísticas
  - Metarreglas
  - Funciones $h(n, t)$
- Las funciones heurísticas se **descubren** resolviendo modelos *simplificados* del problema real

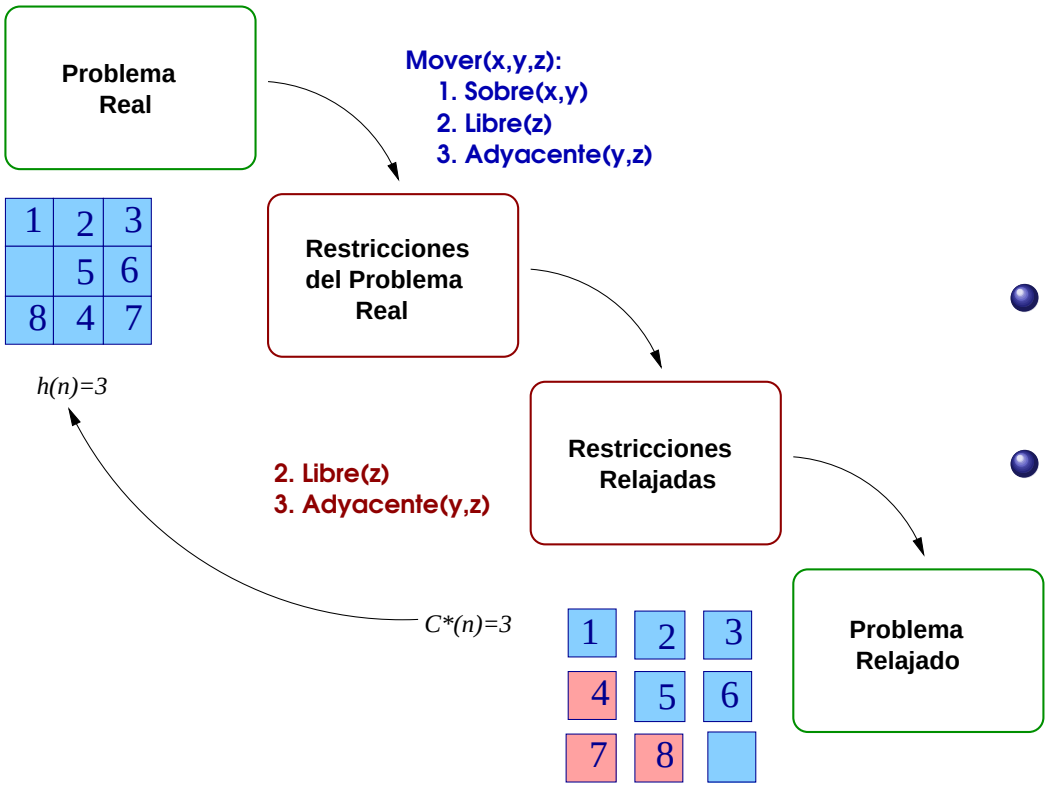
Relajación en programación entera

- Encontrar el par de números enteros (x, y) que maximiza/minimiza una función
- La relajación consiste en iniciar una búsqueda preliminar en el dominio continuo de una o más variables



Relajación de restricciones

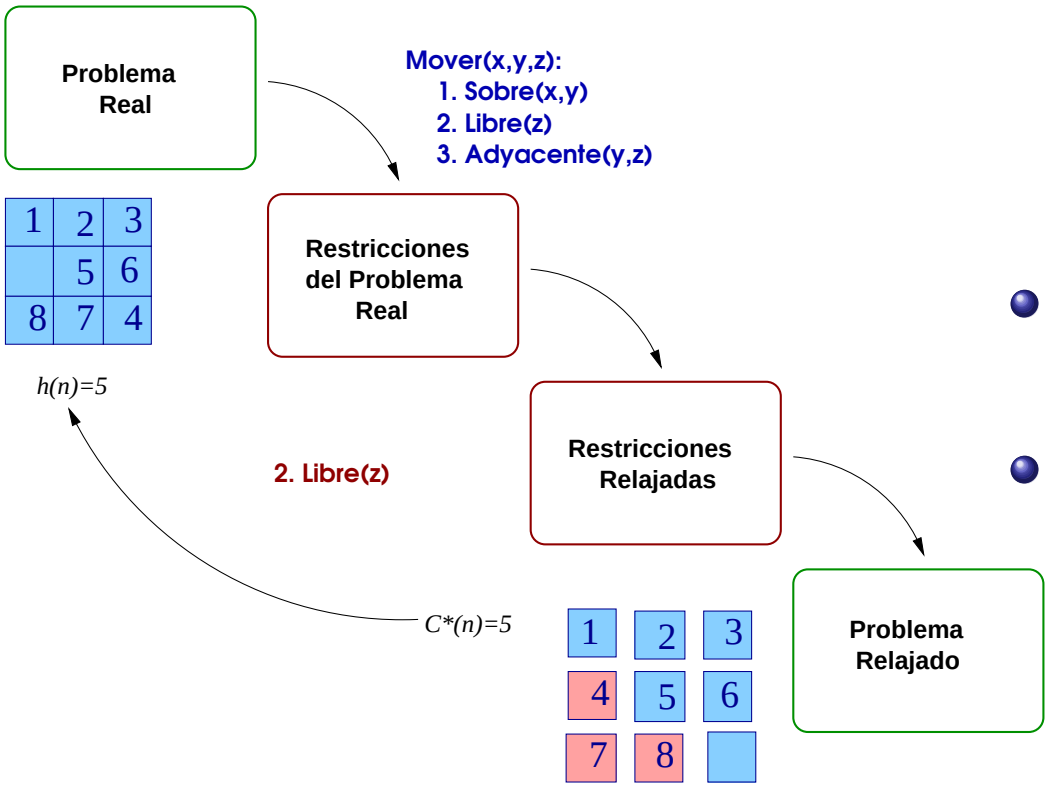
- Típicamente, las funciones heurísticas se obtienen por **relajación de las restricciones** del problema original



- $h(\cdot)$ es la solución **óptima** del problema relajado
- $h_1(\cdot)$: heurística del número de posiciones mal dispuestas

Relajación de restricciones

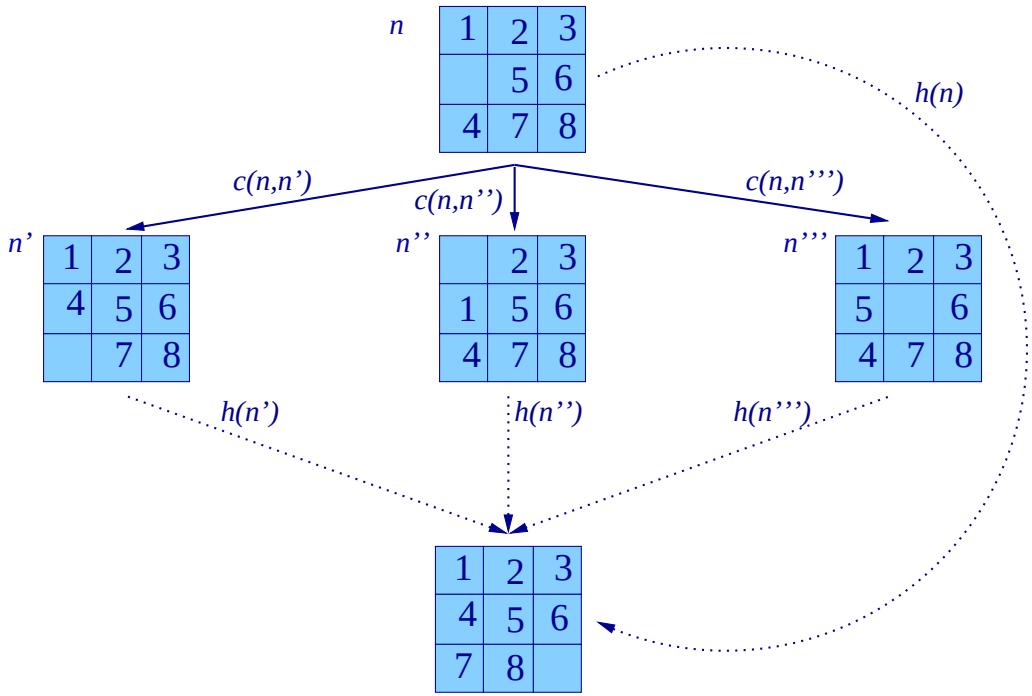
- Típicamente, las funciones heurísticas se obtienen por **relajación de las restricciones** del problema original



- $h(\cdot)$ es la solución **óptima** del problema relajado
- $h_2(\cdot)$: heurística de Manhattan

Admisibilidad

- Las funciones $h(\cdot)$ obtenidas por relajación deben ser necesariamente *monótonas*:



$$h(n) \leq c(n, n') + h(n')$$

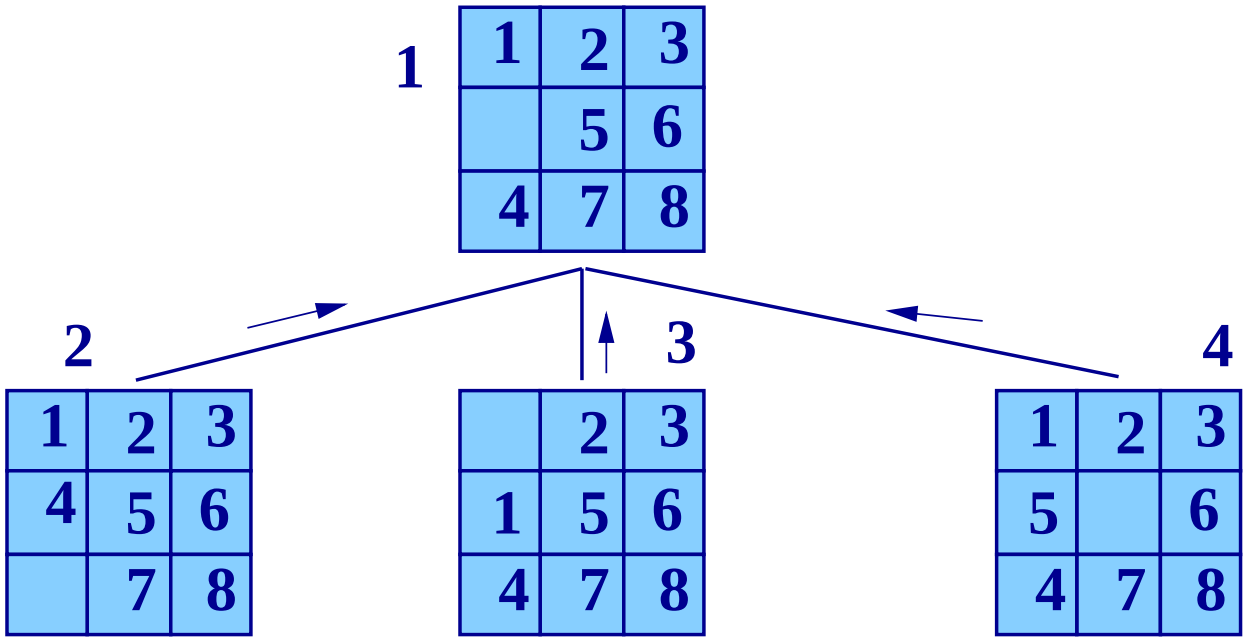
- Y, por lo tanto, *admisibles*:

$$h(n) \leq h^*(n)$$

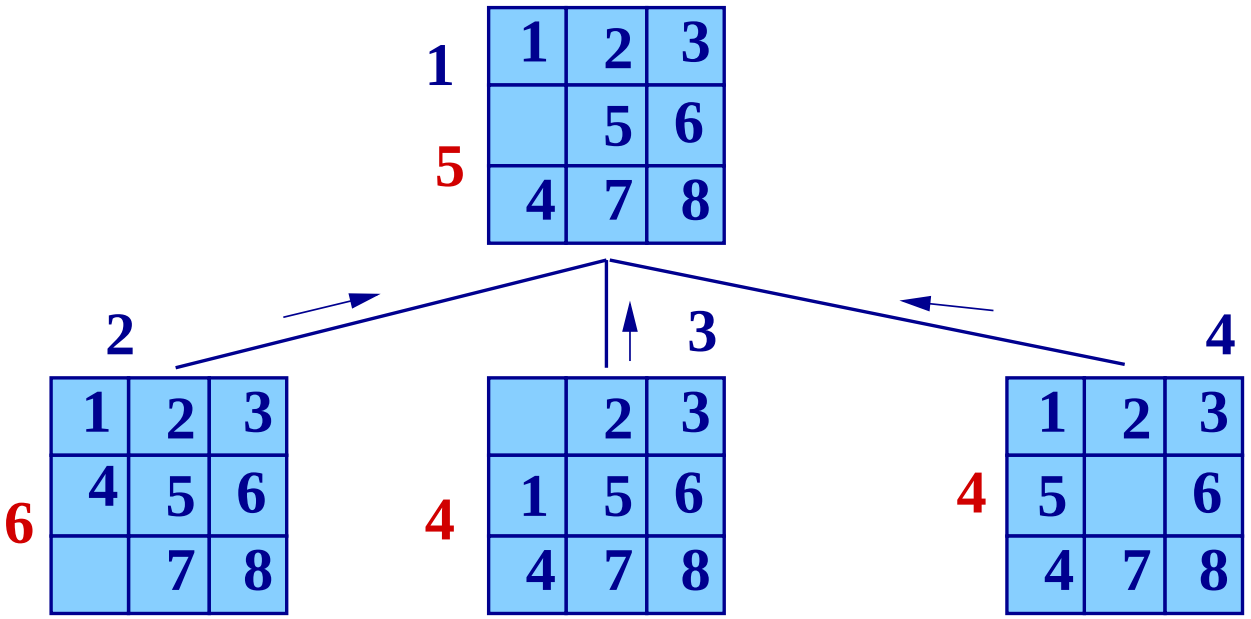
Otras relajaciones

- La *relajación* no es la única forma de simplificar problemas
- ¡Tampoco es cierto que cualquier *relajación* sea más fácil de resolver que el problema original!
- Otras alternativas son:
  - **Añadir restricciones** al problema original $\longrightarrow$ heurísticas no admisibles. Se usa para eliminar alternativas anticipadas que exceden el coste $h(\cdot)$
  - Por **estimación probabilística** de los descendientes más prometedores
  - Por **razonamiento por analogía** o **metafórico**

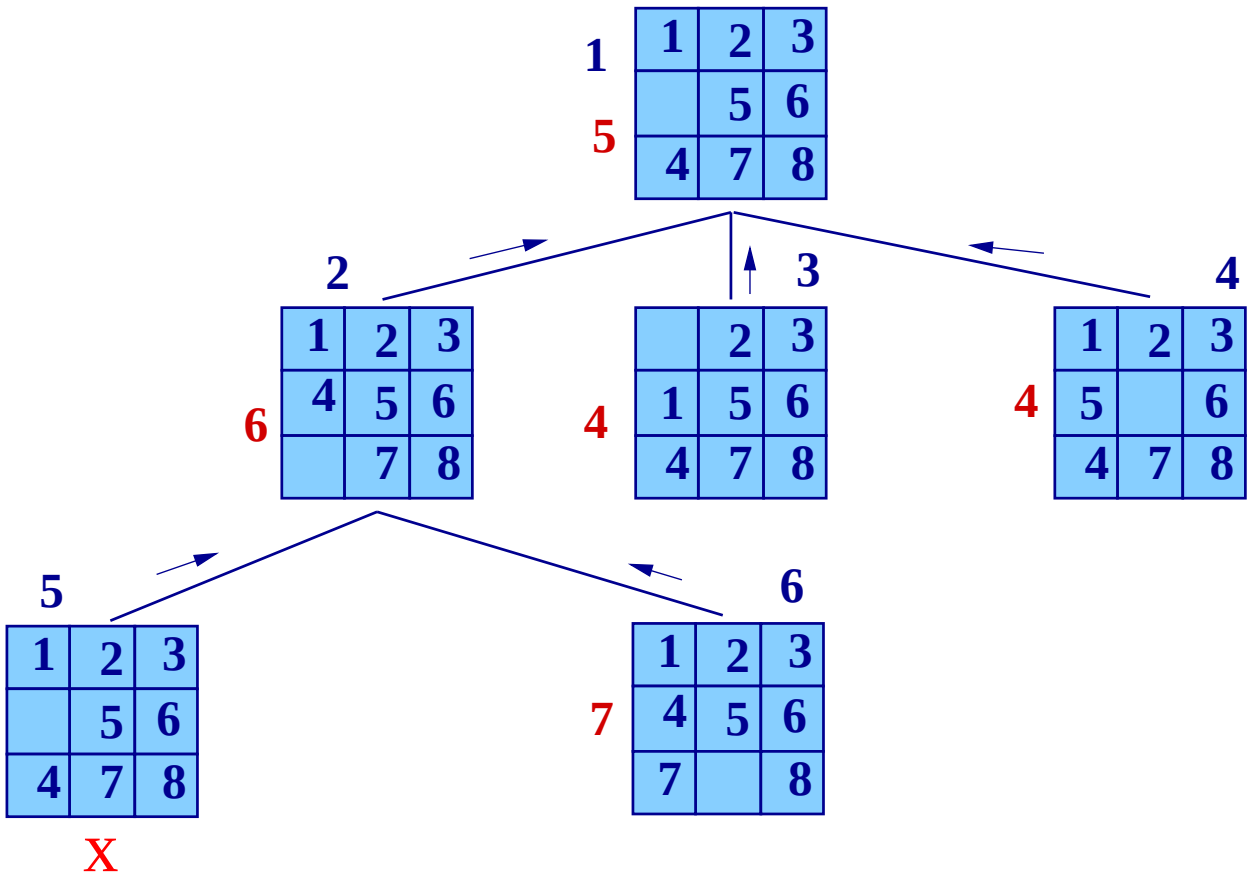
8-Puzle – Búsqueda en escalada



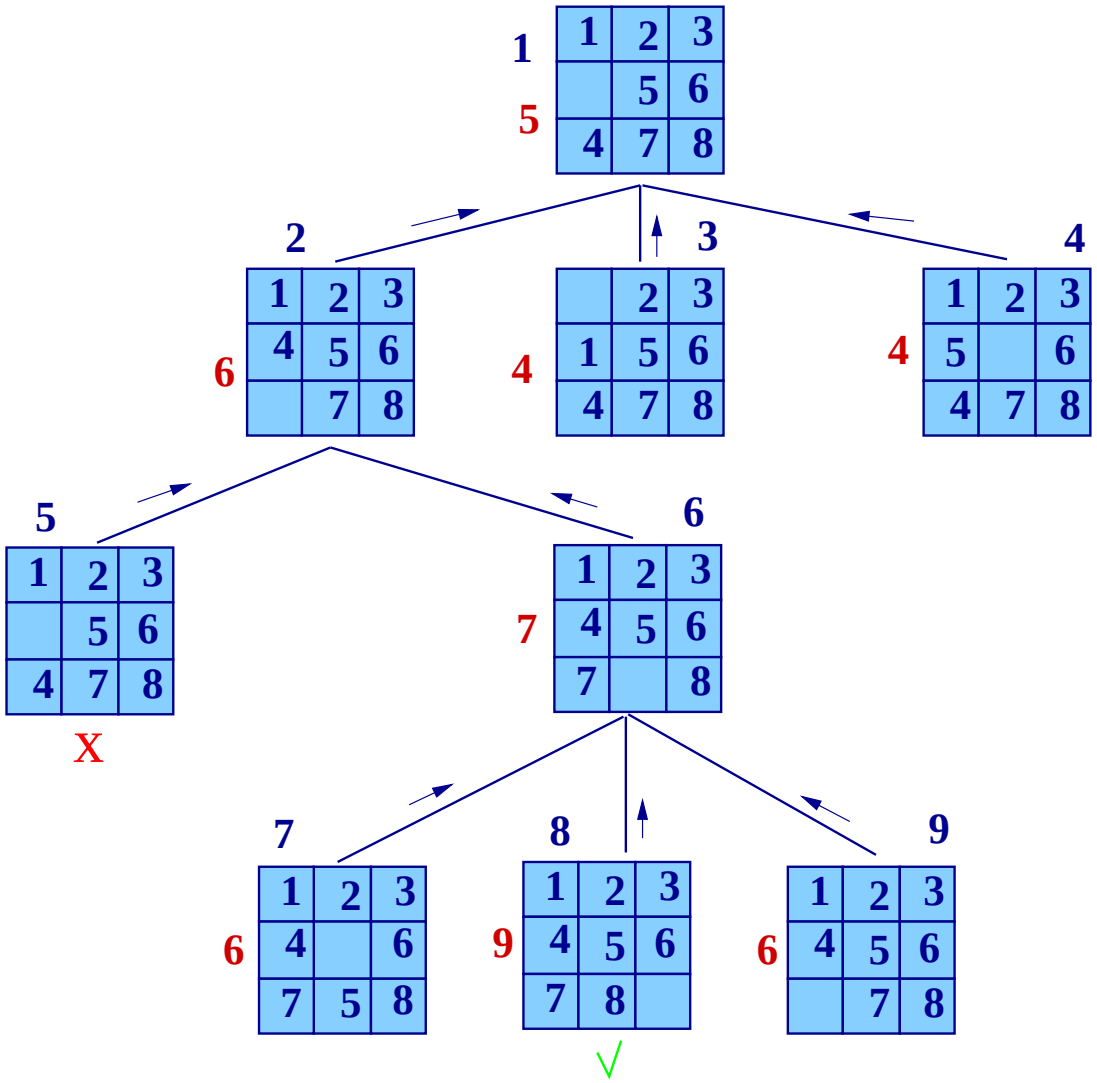
8-Puzle – Búsqueda en escalada



8-Puzle – Búsqueda en escalada



8-Puzle – Búsqueda en escalada



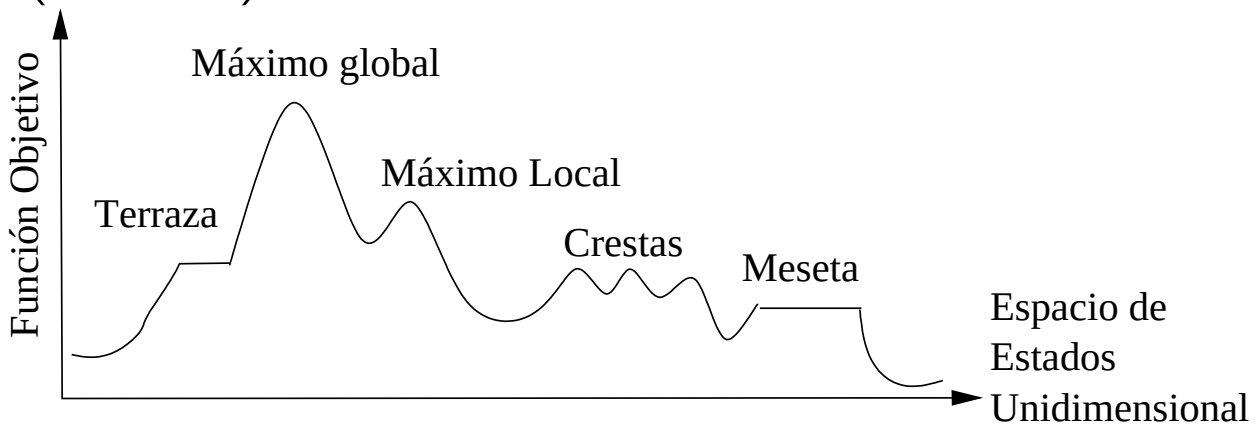
Búsqueda en escalada

Procedimiento escalada (Estado-inicial Estado-final)

N=Estado-inicial; EXITO=Falso
Hasta que ABIERTA esté vacía O EXITO
    Generar los sucesores de N
    SI algún sucesor es Estado-final
    ENTONCES EXITO=Verdadero
    SI NO, Evaluar cada nodo con la función de evaluación, $f(n)$
        N=mejor sucesor
Si EXITO
Entonces Solución=camino desde nodo del Estado-inicial
                  al nodo N por los punteros
Si no, Solución=fracaso

Características

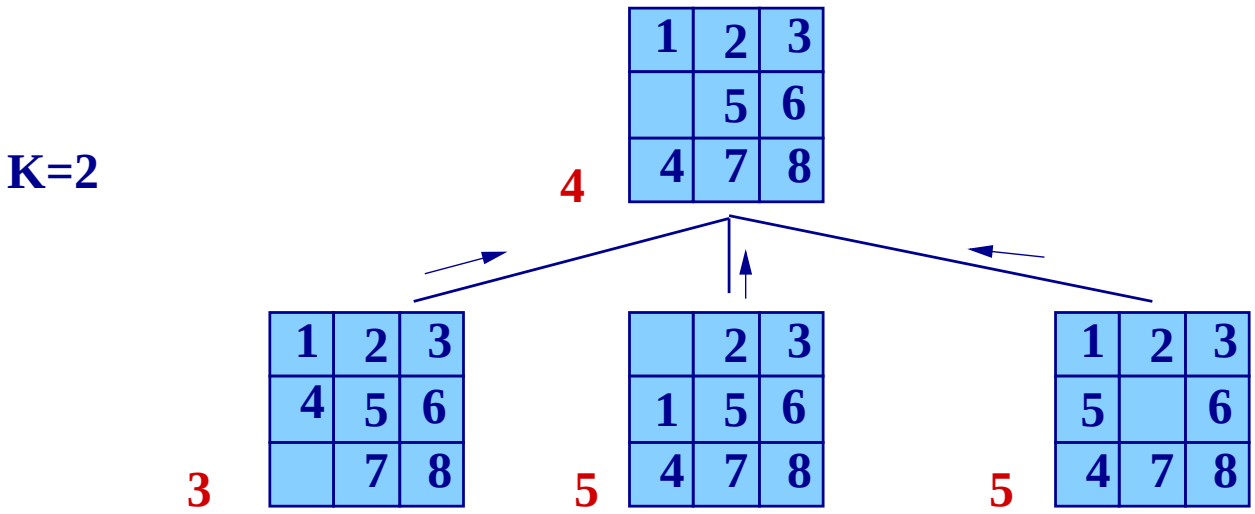
- Problemas de los métodos *avariciosos*
  - **Máximos (o mínimos) locales**: pico que es más alto que cada uno de sus estados vecinos, pero más bajo que el máximo global
  - **Mesetas**: zona del espacio de estados con función de evaluación plana
  - **Crestas**: zona del espacio de estados con varios máximos (mínimos) locales



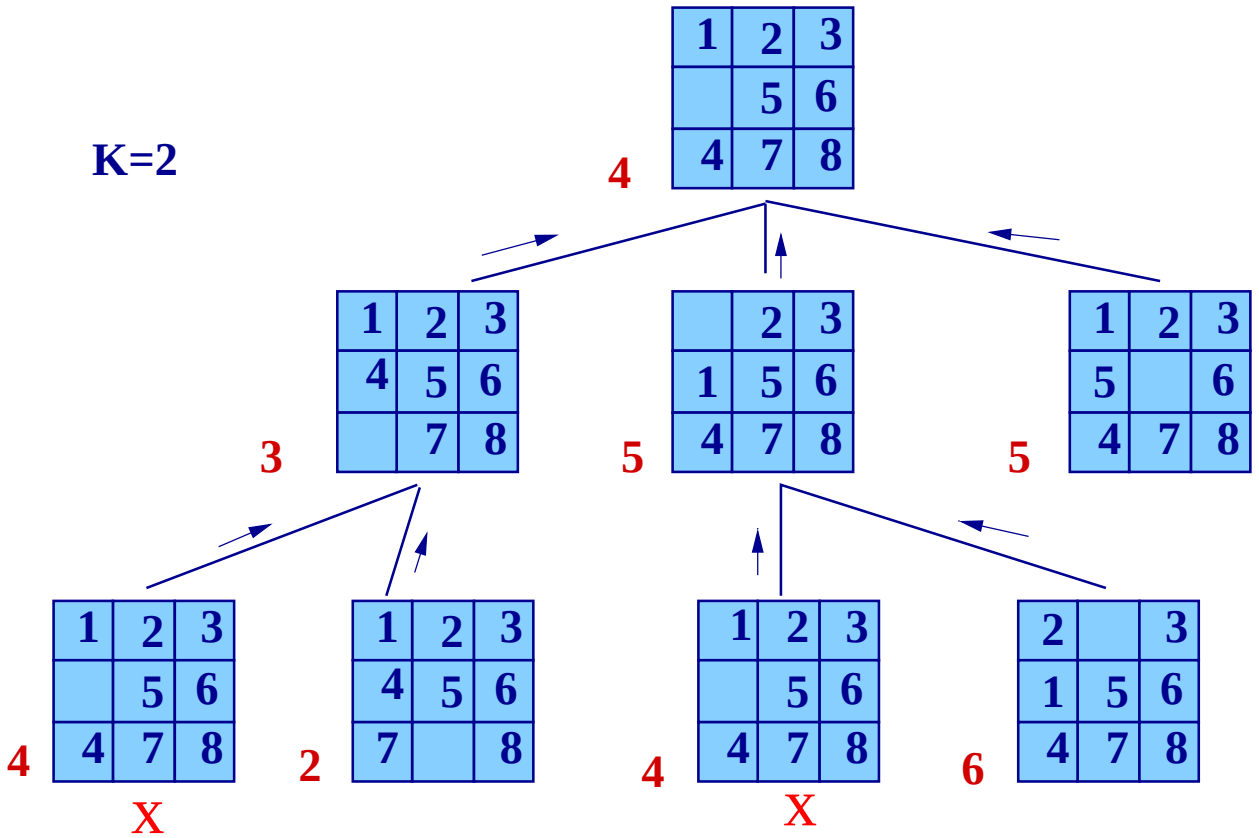
Características

- Soluciones
  - Retroceso
  - Dar más de un paso
  - Reinicio aleatorio
- Método local
  - Completitud: no tiene porqué encontrar la solución
  - Admisibilidad: no siendo completo, aún menos será admisible
  - Eficiencia: rápido y útil si la función es monótona (de)creciente

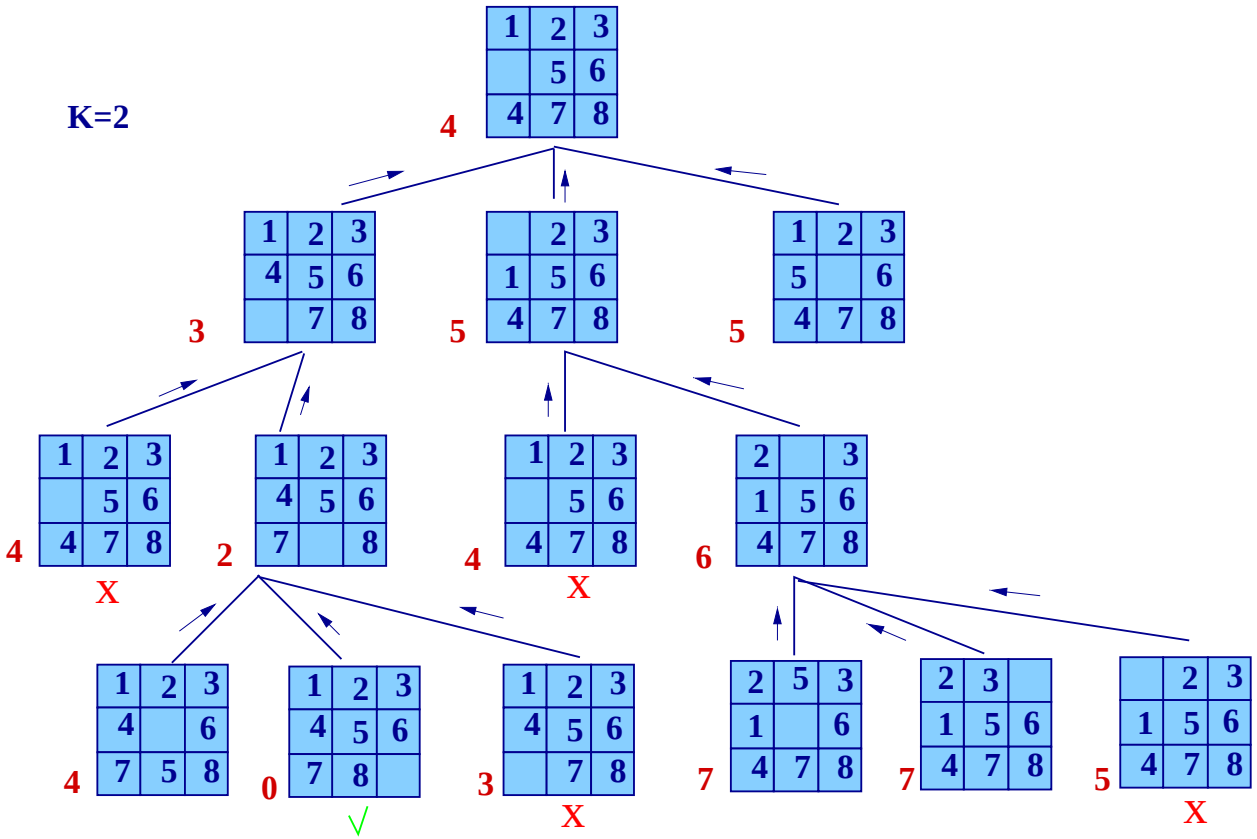
8 Puzle – Búsqueda en haz



8 Puzle – Búsqueda en haz



8 Puzle – Búsqueda en haz



Búsqueda en haz

Procedimiento "Beam Search" (Estado-inicial Estado-final K)

ABIERTA=(Estado-inicial); EXITO=Falso
Hasta que ABIERTA esté vacía O EXITO
    ABIERTA=Todos los sucesores de los nodos de ABIERTA
    SI algún nodo de ABIERTA es Estado-final
    ENTONCES EXITO=Verdadero
    SI NO, Evaluar cada nodo con la función de evaluación $f(n)$
        ABIERTA=K mejores nodos de ABIERTA
Si EXITO
Entonces Solución=camino desde nodo del Estado-inicial
        al nodo N por los punteros
Si no, Solución=fracaso

Características

- La búsqueda en haz es una generalización de la escalada: $HC = BS(k = 1)$
- Abriendo la ventana de sucesores eligibles, mejoran las posibilidades de:
  - Escapar de los *plateaus* o mesetas formadas por la función heurística
  - Encontrar caminos más cortos hasta alguna meta
- Sin embargo, no es cierto que el algoritmo encuentra soluciones mejores con valores de k mayores, aunque lo normal es que sea así

Búsqueda de el mejor primero

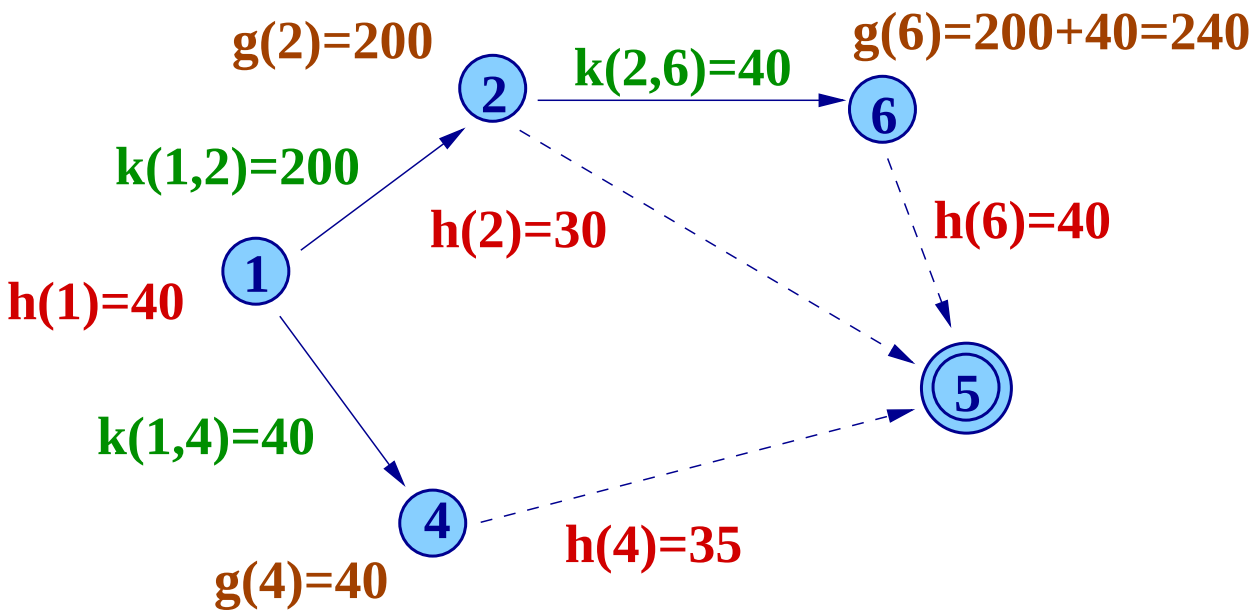
Procedimiento Mejor-primero (Estado-inicial Estado-final)

Crear grafo de búsqueda G, con el nodo inicial, I (Estado-inicial)
ABIERTA=I, CERRADA=Vacío, EXITO=Falso
Hasta que ABIERTA esté vacía O EXITO
    Quitar el primer nodo de ABIERTA, N y meterlo en CERRADA
    SI N es Estado-final ENTONCES EXITO=Verdadero
    SI NO Expandir N, generando el conjunto S de sucesores de N,
                que no son antecesores de N en el grafo
        Generar un nodo en G por cada s de S
        Establecer un puntero a N desde aquellos s de S que no estuvieran ya en G
        Añadirlos a ABIERTA
        Para cada s de S que estuviera ya en ABIERTA o CERRADA
                decidir si redirigir o no sus punteros hacia N
        Para cada s de S que estuviera ya en CERRADA
                decidir si redirigir o no los punteros de los nodos en sus subárboles
        Reordenar ABIERTA según $f(n)$
Si EXITO Entonces Solución=camino desde I a N a través de los punteros de G
Si no Solución=Fracaso

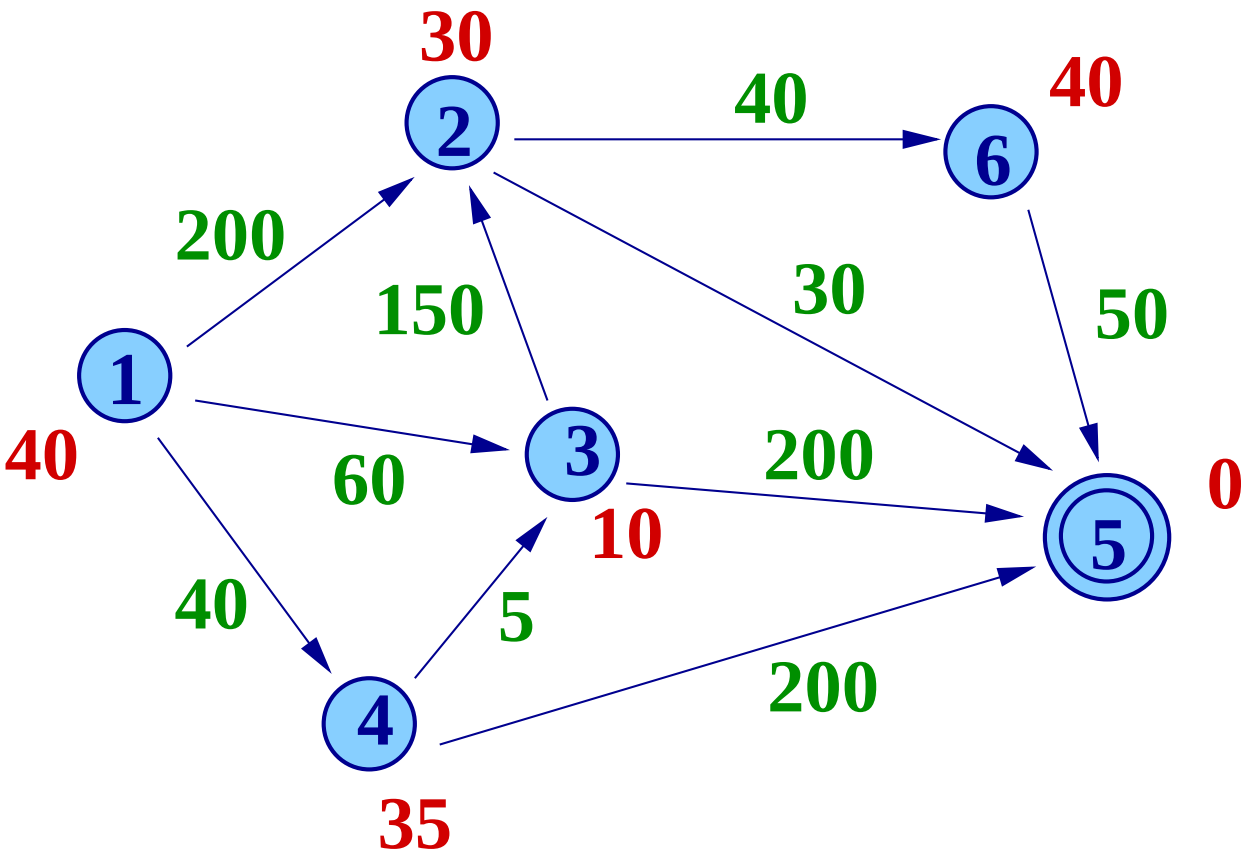
A* (Hart, Nilsson y Raphael, 1968)

- Función de ordenación de nodos: $f(n) = g(n) + h(n)$
  - $f(n)$: función de evaluación
  - $g(n)$: función de coste de ir desde el nodo inicial al nodo n
  - $h(n)$: función heurística que mide la distancia estimada desde n a algún nodo meta
- $g(n)$ se calcula como la suma de los costes de los arcos recorridos, $k(n_i, n_j)$
- Los valores reales sólo se pueden conocer al final de la búsqueda
  - $f^*(n)$: coste real para ir desde el nodo inicial a algún nodo meta a través de n
  - $g^*(n)$: coste real para ir desde el nodo inicial al nodo n
  - $h^*(n)$: coste real para ir desde el nodo n a algún nodo meta

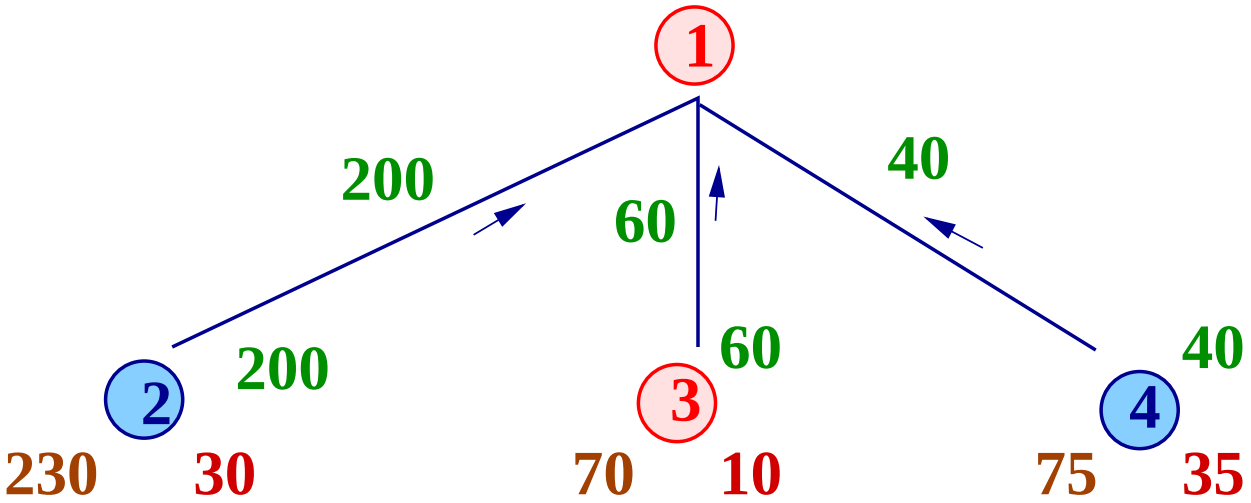
Grafo no dirigido – A* (definiciones)



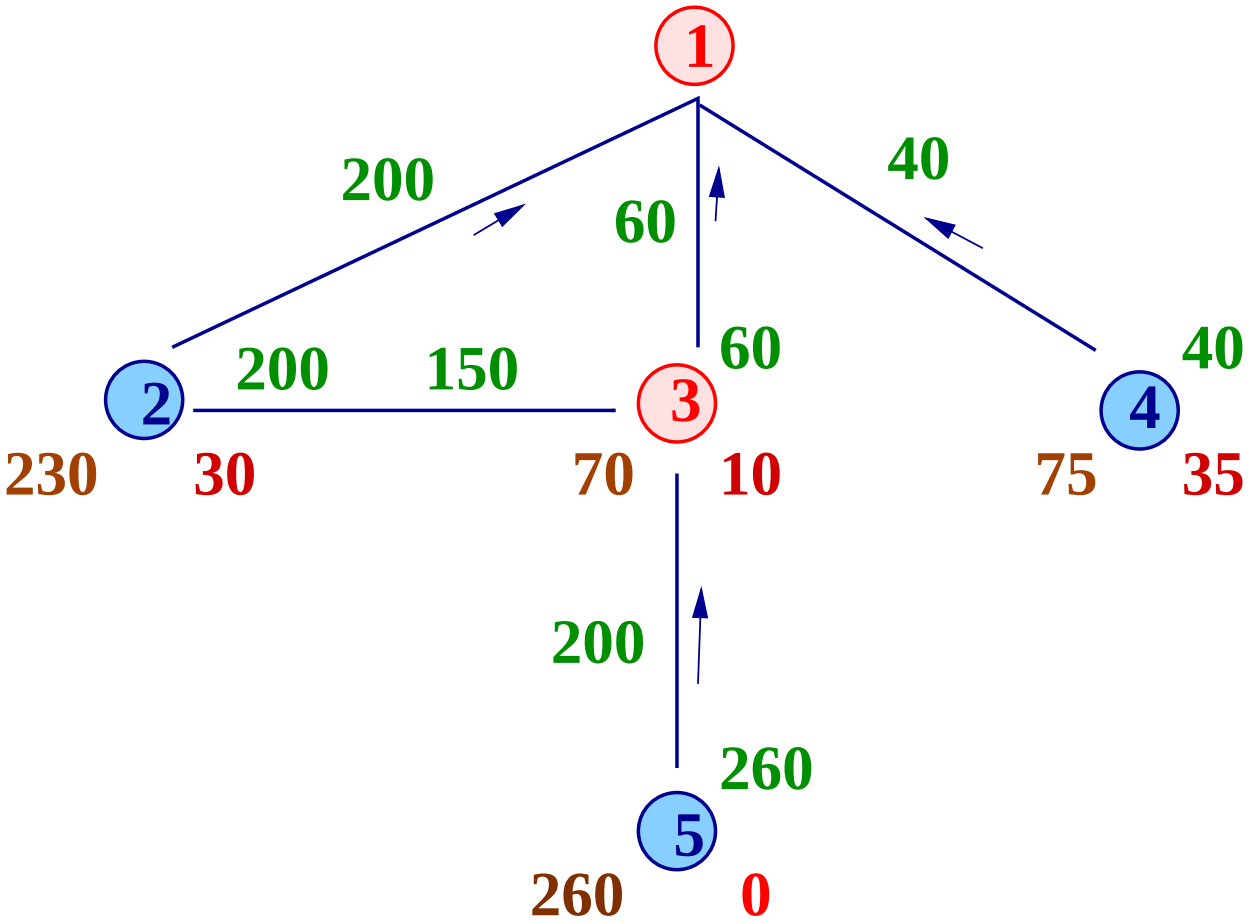
Grafo no dirigido – A*



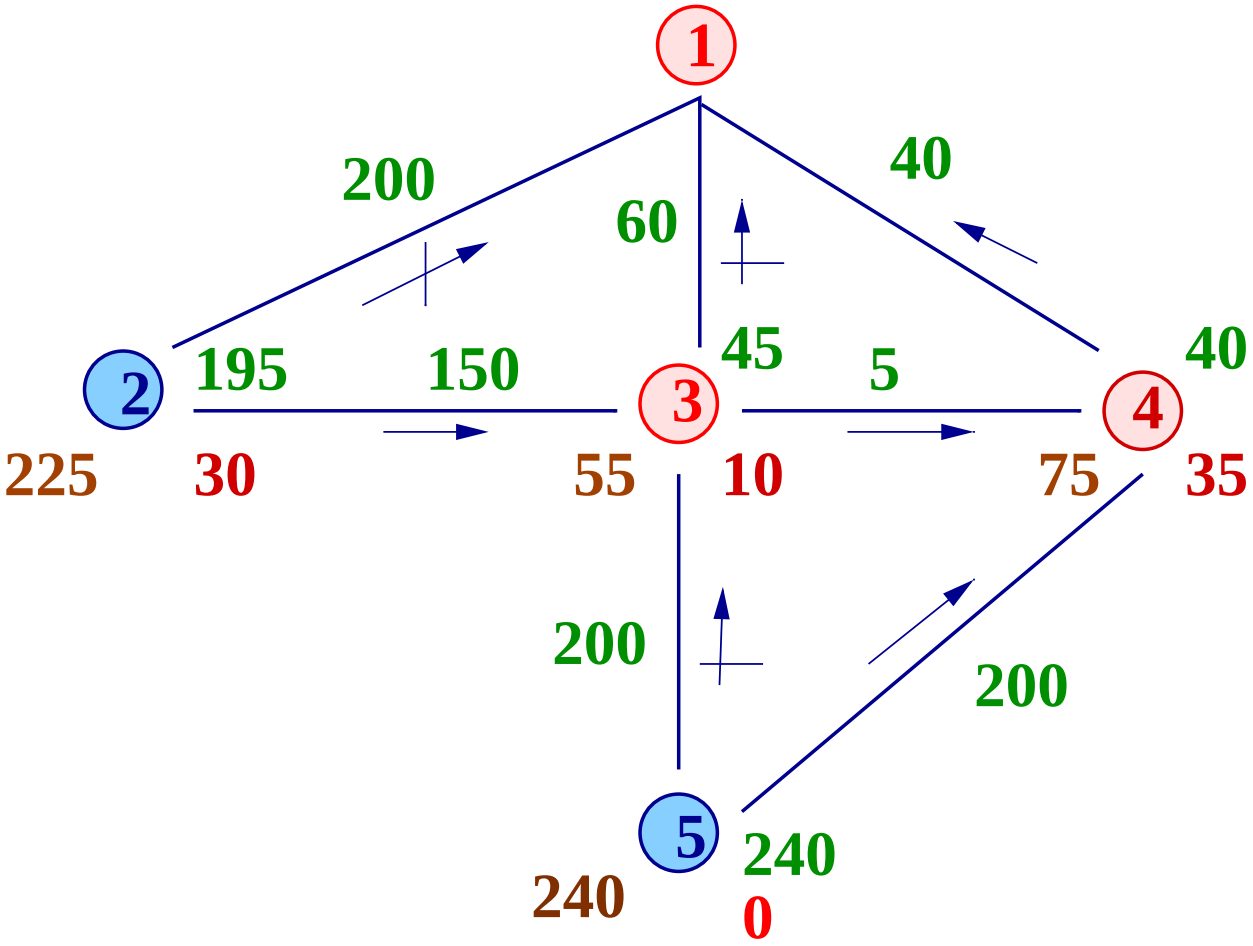
Grafo no dirigido – A*



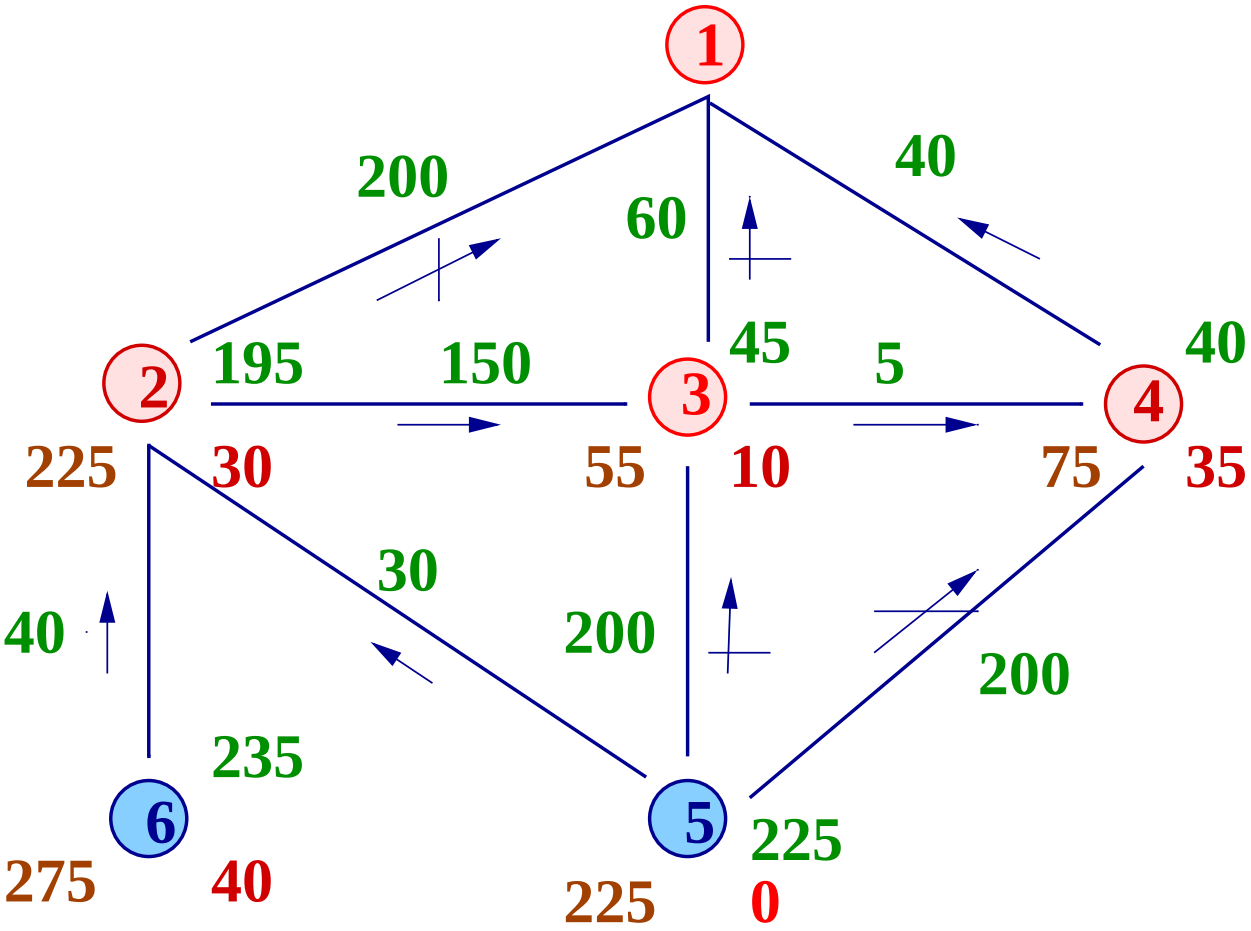
Grafo no dirigido – A*



Grafo no dirigido – A*



Grafo no dirigido – A*



Características

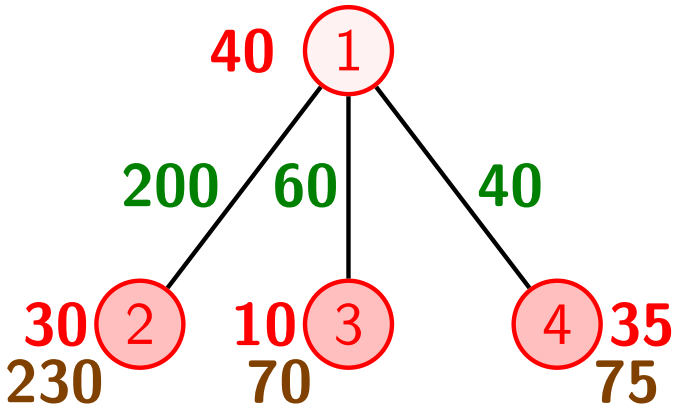
- Completitud: si existe solución, la encuentra
- Admisibilidad: si hay una solución, encuentra la óptima si:
  - el número de sucesores es finito para cada nodo,
  - $k(n_i, n_j) \geq \epsilon > 0$ en cada arco, y
  - La función heurística $h(\cdot)$ es admisible, $h(n) \leq h^*(n) \quad \forall n$
- Si $h_1(n) <= h_2(n) \forall n$, $h_2(n)$ está más informada que $h_1(n)$ y servirá para expandir menos nodos
  - Ejemplo: distancia de Manhattan está más informada que número de casillas mal colocadas (problema de Manhattan es menos relajado que el del número de casillas)
- Extremos:
  - $h(n) = 0$ para cada nodo: no se tiene información (Dijkstra)
  - $h(n) = h^*(n)$ para cada nodo: se tiene información perfecta
- No tiene sentido dedicar más coste computacional a calcular una buena $h(n)$ que a realizar la búsqueda equivalente: equilibrio

Resumen de técnicas de el mejor primero

- No informadas:
  - Búsqueda en amplitud: $f(n) = \text{profundidad}(n)$
  - Dijkstra: $f(n) = g(n)$
- Informadas (heurísticas):
  - Escalada y búsqueda en haz: $f(n) = h(n)$
  - A*, IDA*: $f(n) = g(n) + h(n)$
  - Ponderadas: $f(n) = g(n) + \omega h(n), \omega > 1$
    - Es completo, pero no es admisible
    - La solución óptima tiene un coste menor o igual que $(1 + \omega)$ veces la generada

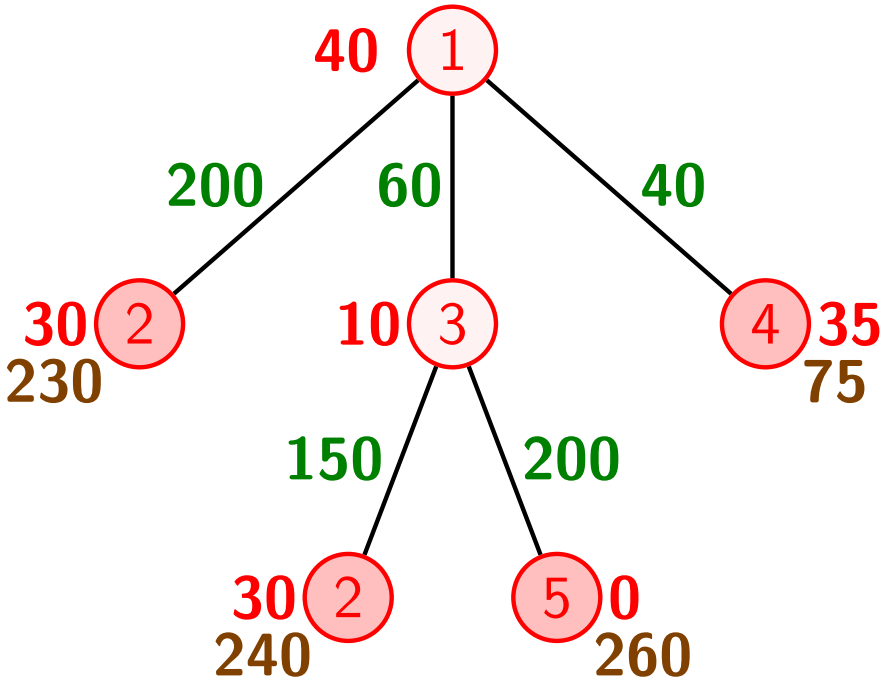
Grafo no dirigido – IDA* (Korf, 1985)

Iteración 1, $\eta_1 = h(1) = 40$



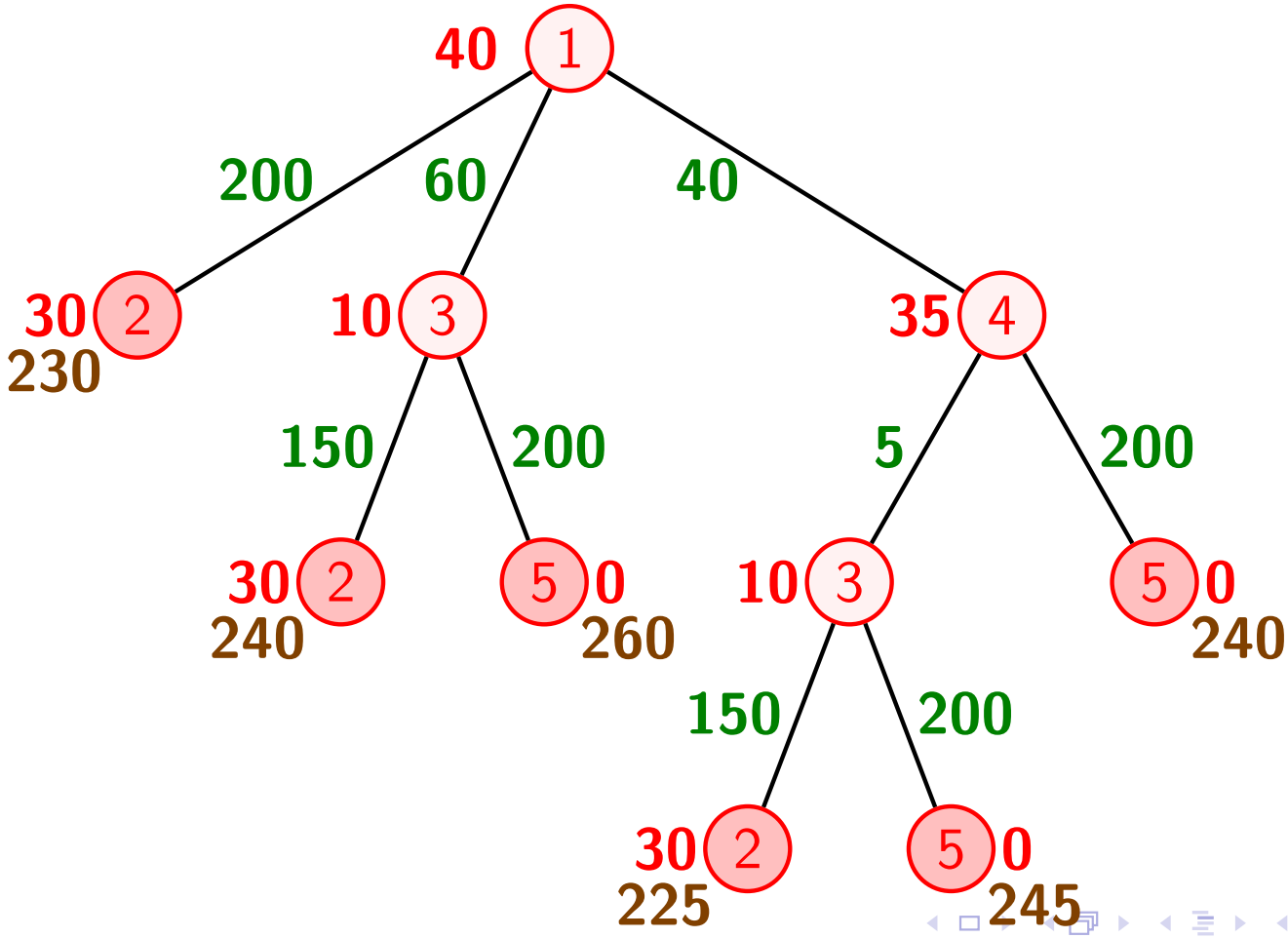
Grafo no dirigido – IDA*

Iteración 2, $\eta_2 = f(3) = 70$



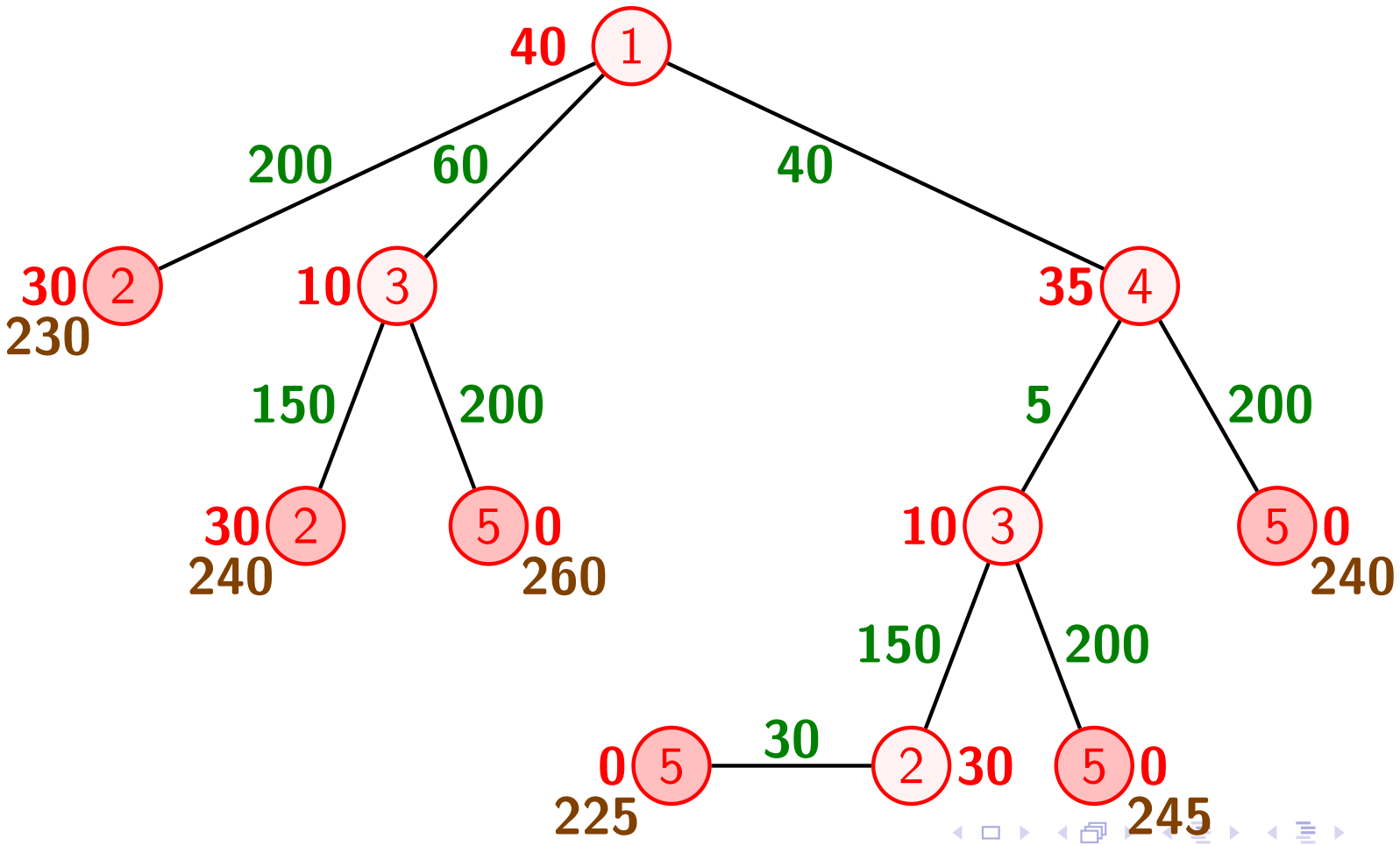
Grafo no dirigido – IDA*

Iteración 3, $\eta_3 = f(4) = 75$



Grafo no dirigido – IDA*

Iteración 4, $\eta_4 = f(2) = 225$



IDA*

Procedimiento IDA* (Estado-inicial Estado-final)

EXITO=Falso
$\eta = h(s)$
Mientras que EXITO=Falso
    EXITO=Profundidad (Estado-inicial,η)
    $\eta = \min_{i=1,n} \{f(i)\} = \min_{i=1,n} \{g(i) + h(i)\}$
Solución=camino desde nodo del Estado-inicial
        al Estado-final por los punteros

Profundidad (Estado-inicial,η)

Expande todos los nodos cuyo coste $f(n)$ no excede el valor de η

Características

- **Completitud**: El algoritmo IDA* es completo, esto es, encuenta una solución si existe alguna
- **Admisibilidad**: Además, el algoritmo IDA* es admisible y, por lo tanto, encontrará la solución óptima
- Mientras su complejidad de tiempo es también exponencial, su complejidad de espacio es lineal en la profundidad del árbol de búsqueda
- Aunque pudiera parecer lo contrario, el número de *re-expansiones* es sólo mayor en un pequeño factor que el número de expansiones de los algoritmos de el mejor primero
- Fue el primer algoritmo que resolvió óptimamente 100 casos generados aleatoriamente en el 15–Puzle

K-Best-First-Search (Felner, Kraus & Korf, 2003)

- KBFS (k) expande los mejores k nodos de la lista $\mathrm{ABIERTA}$ e inserta sus sucesores
- Es una generalización de Mejor-primero:
  - KBFS (1) = BFS (*Best-First-Search*)
  - KBFS (∞) = Primero en amplitud
- Ventajas:
  - Buscando simultáneamente en k descendientes se evitan los *dead-ends*
  - Para algunos valores de k devuelve la solución rápidamente
- Inconvenientes:
  - Es preciso determinar el valor de k

Búsqueda en tiempo real

- Los métodos anteriores (A^*, IDA^*, etc.) suelen consumir tiempo exponencial para encontrar una solución óptima
- La búsqueda en tiempo real asume que el tiempo para decidir qué acción ejecutar en cada momento está limitado
- Por tanto, podría no haber tiempo suficiente para obtener el camino óptimo "a priori"
- Es necesario comenzar a ejecutar acciones antes de disponer el plan de acción definitivo
- Ejemplo: RTA^*
  - ¿Cuándo retrocedo a un estado visitado anteriormente? cuando la estimación de resolver el problema desde ese estado más el coste de volver hasta ese estado es menor que el coste estimado de llegar a la meta desde donde estoy hacia adelante.

RTA* (Korf, 1988)

Algoritmo RTA* (Estado-inicial, Metas, Profundidad-máxima)

1. N=estado inicial
2. Generar el conjunto $S(N)$ de todos los sucesores de N
3. Si algún sucesor de N es la meta, entonces terminar
4. Estimar el valor de cada sucesor realizando una búsqueda desde él en profundidad con Profundidad-máxima

$$f(n) = \begin{cases} g(n) + h(n) & \text{si n es nodo hoja} \\ \text{mín}_{n' \in S(n)} f(n') & \text{en caso contrario} \end{cases}$$

5. N=$arg\ \text{mín}_{n' \in S(N)} f(n')$
6. Se ejecuta la acción correspondiente en el mundo real
7. Se guarda N en una tabla junto al valor del segundo mejor sucesor. De esta forma se evita tomar la misma acción en el mismo nodo dos veces. La siguiente vez que se visite el nodo, se devuelve el valor

Características

- **Tiempo real**: se puede ajustar la profundidad de búsqueda dependiendo del tiempo disponible para realizar la acción
- **Completitud**: RTA* encontrará una solución, si existe
- **Admisibilidad**: la solución no tiene por qué ser la óptima
- Este algoritmo se basa, en parte, en los algoritmos que se aplican a juegos