Inteligencia Artificial

Sesión 3

"Búsqueda Informada"

Web:

Ingeniería de Sistemas de Información,
Semestre académico: 2019 -II
Profesor: Oswaldo Vélez Langs, PhD
velezlangs@gmail.com

Tema 3: Juegos unipersonales con información

Resumen:

2. Juegos unipersonales
   - 2.1. Representación básica
   - 2.2. Juegos con información completa
   - 2.3. Recursos limitados en juegos con información completa
     - Heurísticas
     - Búsqueda voraz
     - Búsqueda A*
     - Búsqueda con subobjetivos
     - Búsqueda por ascenso de colinas
     - Búsqueda con horizonte
   - 2.4. Juegos con información incompleta

Problema

En determinados problemas es imposible/impracticable buscar el plan de acciones completo.

Las razones pueden ser:

- Limitación de recursos computacionales:
  - el espacio de búsqueda es demasiado grande para buscar hasta la solución (no pueden emplear las técnicas anteriores)
  - Ejemplo: Ajedrez, 24-Puzzle, ...
- Información incompleta o entornos dinámicos:
  - no se conocen todos los estados posibles del problema o no se pueden predecir los posibles efectos de acciones
  - Ejemplo: laberinto, laberinto con obstáculos en movimiento
- Agentes reactivos que actúan de forma rápida o cuyo objetivo es continuo:
  - es imposible encontrar un plan de acciones hasta el final, por falta de tiempo o porque el estado objetivo es continuo
  - solo hace falta una acción "buena" (no un plan de acción)
  - Ejemplo: conducir un coche, agentes de control, ...

Solución: Información heurística

1. Acotar el espacio de búsqueda:

- Reducir anchura del árbol expandido: eliminar algunas ramas porque *probablemente no reflejan la solución óptima*
- *¿Qué ramas no necesitan ser explorados?*
- Reducir profundidad del árbol expandido: expandir el árbol sólo hasta un determinado nivel y elegir el nodo más prometedor
- *¿Cuál es el nodo más prometedor ?*

2. Guiar la búsqueda:

- explorar *ramas prometedoras* del árbol antes que otras/ expandir determinados nodos antes que otras
- *¿Qué nodos hay que expandir primero?*

Se requiere información adicional: información heurística

- Heurística (griego: heuriskein): "encontrar", "descubrir"
- compila conocimiento "empírico" sobre un problema / un entorno
- suele reflejar una *estimación* o *aproximación* de la solución (**esencialmente imperfecta**)

Heurísticas

Aplicación "fuerte":

- la aplicación de la heurística reduce el problema
- se deja de analizar soluciones posibles pero improbables
- Objetivo:
  - encontrar soluciones "buenas" (no necesariamente óptimas)
  - minimizar el índice competitivo: $\frac{\textit{Coste del camino real del agente}}{\textit{Coste del camino óptimo}}$
- mejorar el rendimiento de forma substancial / optimalidad y completitud no garantizados

Aplicación "débil":

- método de búsqueda + información heurística
- la información heurística no reduce el problema sino **guía** la búsqueda
- Información heurística puede mejorar el rendimiento medio de un método, pero no asegura una mejora en el peor caso

Funciones heurísticas

Funciones heurísticas en la búsqueda en el espacio de estados:

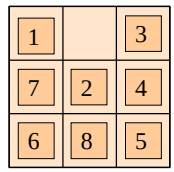
- estiman la cercanía de un estado a un estado meta
- Posibles funciones heurísticas:
  - "distancia": coste estimado del camino de un estado al estado meta
    - $h: N \rightarrow \aleph$ mide el coste *real* desde el nodo *n* hasta el nodo meta más cercano
    - $h^*: N \rightarrow \aleph$ es una función *heurística* que estima el valor de $h(n)$
  - similitud de un estado con el estado meta (mide la "calidad" o "utilidad" de un estado)
  - Posibilidad de un estado de pertenecer al camino óptimo

Ejemplos de funciones heurísticas:

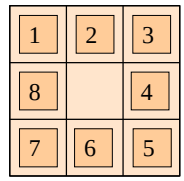
- Laberinto: distancia en *línea recta* hasta la salida
- Ajedrez: número de piezas de un jugador
- Encontrar una ruta de Madrid a Bilbao en un mapa de capitales de provincias: Sevilla no; Burgos si; Soria quizas; …

Encontrar Funciones Heurísticas: Diseño

Estado inicial



Estado meta



- **Problemas relajados**:
  - menos restricciones para cada operador
  - h^*: distancia h exacta en el problema relajado
- **8 Puzzle**: una pieza puede moverse de A a B...
  a) siempre
  b) si B está vació
  c) si A es adyacente a B
- **Funciones heurísticas**:
  a) número de *piezas descolocadas*
     - $h_a^*(s_0) = 4$
  b) suma de *saltos* necesarios
     - $h_b^*(s_0) = 5$
  c) suma de las *distancias de Manhattan*
     - $h_c^*(s_0) = 1+1+1+2=5$
- **Basadas en el coste del camino restante:**
  - valores pequeños reflejan estados buenos

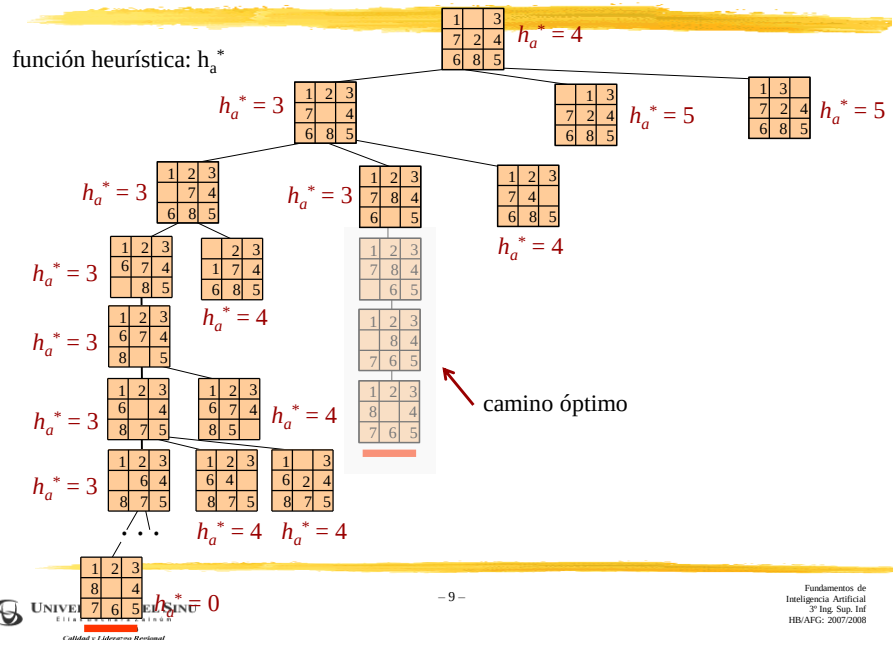
Búsqueda voraz

Búsqueda voraz:

- Inglés: greedy search
- Idea:
  - emplear una función heurística h^* para guiar la búsqueda (heurística débil)
  - minimizar el coste estimado *para llegar a la meta*
- Estrategia:
  - Entre las hojas del árbol de búsqueda, seleccionar el nodo que minimice $h^*(n)$
- Algoritmo:
  - mantener la lista *abierta* ordenada por valores crecientes de h^*
  - insertar nuevos nodos en *abierta* según sus valores h^*

{búsqueda voraz}
abierta ← s_0
Repetir
  Si *vacía?*(abierta) **entonces**
    devolver(negativo)
  nodo ← primero(abierta)
  Si *meta?*(nodo) **entonces**
    devolver(nodo)
  sucesores ← *expandir*(nodo)
  Para cada n∈sucesores **hacer**
    n.padre ← nodo
    ordInsertar(n,abierta,h^*)
  Fin {repetir}

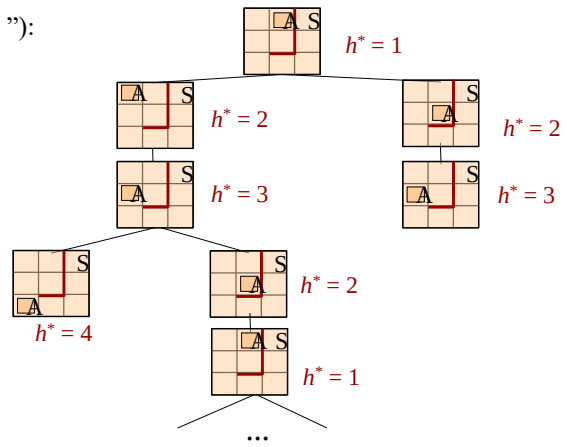
Búsqueda voraz (evitando ciclos simples): Ejemplo 1



Búsqueda voraz (evitando ciclos simples): Ejemplo 2

Ejemplo ("*mínimo local* de h^* "):

- Nodo inicial:
- Nodo meta:
- *h*:* distancia de Manhattan



Búsqueda voraz: complejidad

Depende de la función heurística:

- *d* profundidad de la mejor solución, *b* factor de ramificación efectivo
- si *h*(n)=h(n):*
  - la búsqueda "va directamente a la solución"
  - caso irreal, ya que significa que se conozca la solución
  - complejidad en espacio: O(d·b)
  - complejidad en tiempo: O(d)
- otros casos:
  - complejidad en espacio y en tiempo puede ser mayor que en la búsqueda por amplitud (si la función heurística es mala)
- *OJO:* hay que contar la complejidad de calcular *h**

Búsqueda voraz: análisis

Análisis:

- en general, la búsqueda voraz sufre los mismos problemas que la búsqueda en profundidad
  - no es óptima (ejemplo 1)
  - no es completa (ejemplo 2)
- sin embargo, suele encontrar una solución *aceptable* de forma *rápida*

Comentarios:

- para asegurar la *completitud* habría que evitar todos los estados repetidos
- el método es *óptimo:*
  - si la función heurística devuelve siempre el coste exacto del camino: para todo *n*: *h*(n)=h(n)*
  - en aquellos espacios de estados, en los que el coste de un nodo *n* es independiente del camino por el que se ha llega hasta él (p.e.: n-Reinas)

Ejercicio 2.8

Búsqueda voraz: 8-Puzzle

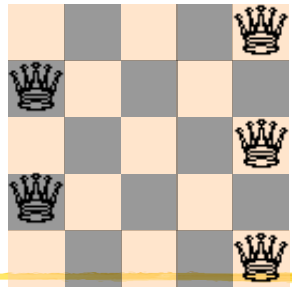
a) Aplica la búsqueda voraz al problema presentado en el ejemplo 1. Utiliza en este caso la función heurística h_c*.

b) Compara la solución encontrada y el árbol expandido con la del ejemplo 1. ¿Qué función heurística es mejor? ¿Por qué?

Ejercicio 2.9

Problema de las 5 reinas:

- 5 reinas en un tablero 5x5
- *estados*: casillas de las 5 reinas
- *meta?*: ninguna reina amenazada
- *op.*: mover una reina a otra casilla **de su *misma* fila**
- *coste*: el coste de cada op. es cero
- *estado inicial*:



Nótese:

- dado que el coste de cada operador es 0, el camino por el cual se llega a un nodo no importa, siempre que al final se encuentre un nodo meta (ninguna reina esta amenazada)

a) encuentre una heurística h^* para el problema de las 5 reinas

b) resuelve el problema aplicando la búsqueda voraz con dicha heurística h^*

Búsqueda A*

Idea:

- emplear información heurística para guiar la búsqueda (heurística débil)
- minimizar el coste estimado *total* de un camino en el árbol de búsqueda
- combinar
  - el coste para llegar al nodo *n* (se conoce exactamente: *g*), y
  - el coste aproximado para llegar a un nodo meta desde el nodo *n* (estimado por la función heurística h^*)

Función heurística de A*:

- $f(n) = g(n)+h(n)$: coste real del plan (camino) de mínimo coste que pasa por *n*
- $f^*(n) = g(n)+h^*(n)$: estimación de f

Estrategia A* :

- entre las hojas del árbol de búsqueda, elegir el nodo de valor f^* mínimo

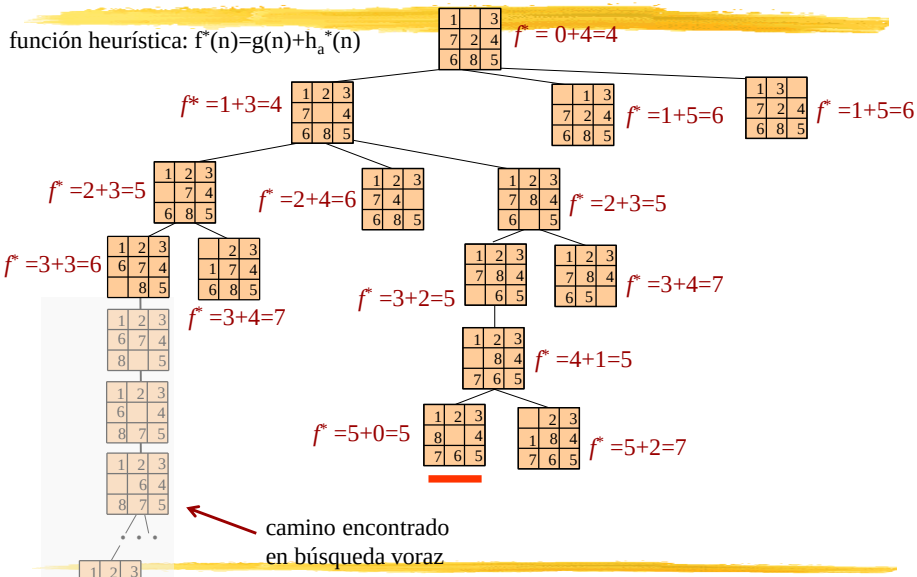
El Algoritmo A*

Algoritmo A* :

- se basa en la búsqueda general
- almacenar el valor *g* de cada nodo expandido
- mantener la lista *abierta* ordenada por valores crecientes de f^*
- insertar nuevos nodos en *abierta* según sus valores f^*

```
{búsqueda A*}
abierta ← s0
Repetir
    Si vacía?(abierta) entonces
        devolver(negativo)
    nodo ← primero(abierta)
    Si meta?(nodo) entonces
        devolver(nodo)
    sucesores ← expandir(nodo)
    Para cada n∈sucesores hacer
        n.padre ← nodo
        ordInsertar(n,abierta, f*)
    Fin {repetir}
```

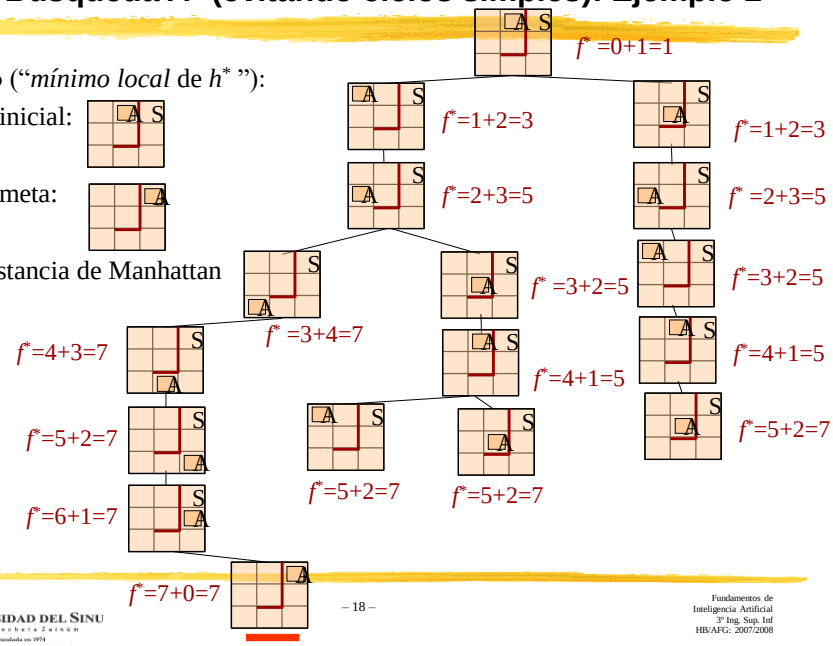
Búsqueda A* (evitando ciclos simples): Ejemplo 1



Búsqueda A* (evitando ciclos simples): Ejemplo 2

Ejemplo ("*mínimo local* de h^* "):

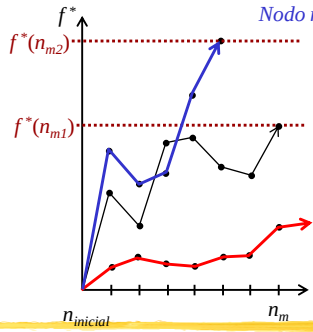
- Nodo inicial:
- Nodo meta:
- *h*:* distancia de Manhattan



Optimalidad y completitud de A*

A* es optimo y completo si:

- h^* es *optimista:* para todos los nodos n_i: $h^*(n_i) \leq h(n_i)$
- para todos los nodos n_i: $h^*(n_i) \geq 0$
- El coste de todas las acciones es mayor que 0 (para todos los nodos hijos n_i de cualquier nodo n_j: $c(n_j, n_i) > 0$)



Nodo meta no óptimo: $g(n_{m2}) > g(n_{m1})$

Mejor nodo meta:
$f^*(n_{m1}) = h^*(n_m) + g(n_{m1}) = g(n_{m1}) = f(n_{m1})$

Camino infinito:
Tiene que cruzar $f^*(n_{m1})$ *an algún punto*

Calidad de las Funciones Heurísticas

Definición:

Sean h_1^* y h_2^* dos funciones heurísticas optimistas.

h_1^* es *más informada* que h_2^*, si para todo nodo n se cumple que

$$h_1^*(n) \geq h_2^*(n)$$

Ejemplo:

- en el 8-puzzle, h_c^* es más informada que h_a^*
  - las piezas bien colocadas no cuenta en h_a^* ni en h_c^*
  - la distancia Manhattan de cada pieza descolocada es al menos 1
  - en consecuencia, en toda posible configuración n del 8-puzzle la suma de las distancias es igual o mayor que la suma de piezas descolocadas
  - para todas las configuraciones n se cumple $h_c^*(n) \geq h_a^*(n)$

Complejidad de A*

El número de nodos expandidos por A* depende de la *calidad* de h^*:

- Para dos funciones heurísticas h_1^* y h_2^*:
  - Si h_1^* es más informada que h_2^*, entonces A*(h_1^*) expande el mismo número o menos nodos que A*(h_2^*)

- La complejidad es más baja si h^* es más informada:
  - si $h^*(n) = h(n)$ para todos los nodos n:
    - información completa: complejidad lineal (sin contar la complejidad de computar h^*!)
    - calcular $h^*(n)$ suele equivaler a resolver el problema completo
  - si $h^*(n) = 0$ para todos los nodos n:
    - A* degenera a la búsqueda por amplitud

Resultados experimentales

Comparación experimental:

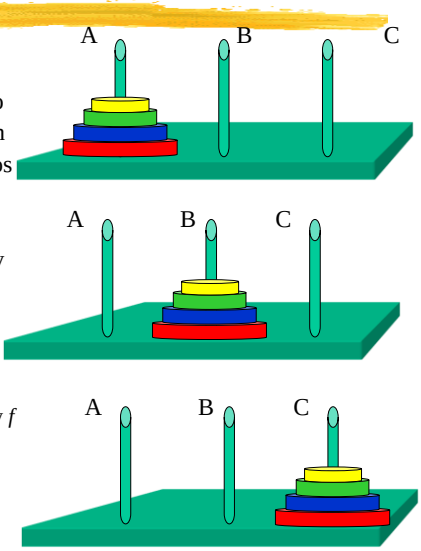
- número de nodos expandidos en el problema del 8-puzzle
- varias profundidades *d* de la solución
- media sobre 100 instancias del problema

d	prof. iterativa	A*(h_a)	A*(h_c)
2	10	6	6
4	112	13	12
6	680	20	18
8	6.384	39	25
10	47.127	93	39
12	3.644.035	227	73
14	—	539	113
16	—	1.301	211
18	—	3.056	363
20	—	7.276	676
22	—	18.094	1.219
24	—	39.135	1.641

Ejercicio 2.10

Algoritmo A^* :

Suponga el juego de las torres de Hanoi con el estado inicial que se presenta al lado. El objetivo consiste en pasar la torre bien a la aguja B o bien a la aguja C. Se filtren todos los estados repetidos y equivalentes (dos estados se consideran *equivalentes*, si están los mismos discos en la aguja *A* mientras que los discos de las agujas *B* y *C* están intercambiadas).



a) Defina una función heurística para este problema.

b) Desarrolle el árbol de búsqueda que genera el algoritmo A*, incluyendo los valores de g, h^*, y f^* de cada nodo. Indique el orden en que se expanden los nodos.

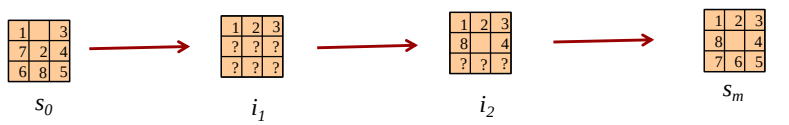
c) ¿Su función heurística es optimista? Argumente su respuesta.

Búsqueda por Subobjetivos

Idea:

- reducir la anchura del árbol al subdividir el problema en varios problemas más pequeños
- determinar una secuencia de estados intermedios $i_1,i_2,...,i_n$ que "muy posiblemente" están en el camino óptimo
- realizar búsquedas con un método base (amplitud, prof., prof. iterativa, ...) desde el estado inicial s_0 a i_1, de i_1 a i_2,…, e de i_n al estado meta s_m
- se aplica la **heurística** para elegir los estados intermedios

Ejemplo 8-Puzzle: obtener filas correctas



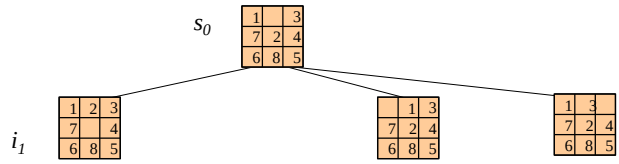
s_0 i_1 i_2 s_m

estado inicial — estados intermedios — estado meta

Búsqueda por subobjetivos (evitando ciclos simples)

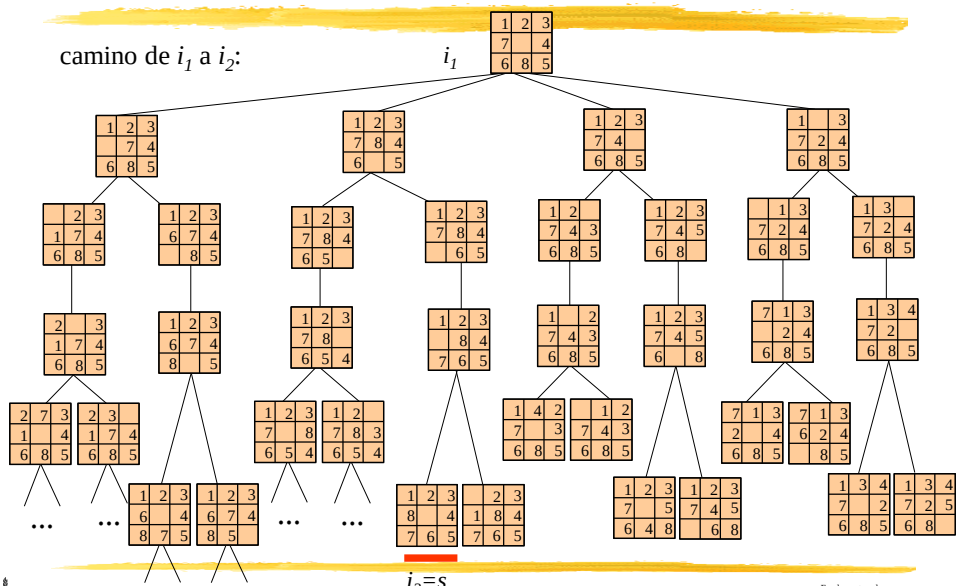
Método base: búsqueda en amplitud

camino de s_0 a i_1:



Búsqueda por subobjetivos (evitando ciclos simples)

camino de i_1 a i_2:



Búsqueda por subojetivos: complejidad

- Complejidad en espacio: proporcional al número de nodos abiertos
- Complejidad en tiempo: proporcional al número de nodos expandidos
- En general se obtiene una mejora de la complejidad respecto a los métodos de búsqueda no informados que se usan como base
  - basado en: la suma de varias búsquedas pequeñas es más eficiente que una búsqueda global
  - depende de la selección inicial de los subojetivos
  - condición necesaria para una mejora: los caminos entre los pares de subojetivos están "más cortos" que la solución global

Búsqueda por subojetivos: Ejemplo complejidad

General: método base: búsqueda en amplitud; factor de ramificación efectivo: *b;* profundidad de la mejor solución: *d; x* subojetivos intermedios; profundidad de los caminos mínimos entre los subobjetivos: *d'*

Ejemplo: *b=3; d=20; x=4; d'=5* (el camino encontrado tiene una longitud de 25)

		General		Ejemplo	
		Bús. prof.	Bús. subob.	Bús. prof.	Bús. subob.
mejor caso:	- tiempo:	$\frac{b^d-1}{b-1}$	$(x+1)*\frac{b^{d'}-1}{b-1}$	1.743.392.200	605
	- espacio:	$\frac{b^{d+1}-1}{b-1}$	$(x+1)*\frac{b^{d'+1}-1}{b-1}$	5.230.176.601	1820
peor caso:	- tiempo:	$\frac{b^{d+1}-1}{b-1}-1$	$(x+1)*\left(\frac{b^{d'+1}-1}{b-1}-1\right)$	5.230.176.600	1815
	- espacio:	$\frac{b^{d+2}-1}{b-1}-b$	$(x+1)*\left(\frac{b^{d'+2}-1}{b-1}-b\right)$	15.690.529.801	5450

Búsqueda por subojetivos: análisis

Análisis:

- Es completo si:
  - El método base es completo, y
  - Los subobjetivos están en (por lo menos) un camino que lleva del estado inicial a la meta
- Es óptimo si:
  - El método base es óptimo, y
  - Los subobjetivos están en este orden en el camíno mínimo desde el estado inicial a la meta

Comentarios:

- Es posible utilizar búsquedas heurísticas como método base (p.e. búsqueda voraz)
- Es posible modificar el método para realizar búsquedas jerárquicas:
  - Aplicar la búsqueda a distintos niveles jerárquicas
  - Por ejemplo: buscar un camino (en coche/barco) desde Madrid a Buenos Aires

Ejercicio 2.11

Problema de las 5 reinas:

Resuelve el problema de las 5 reinas (del ejercicio 2.9) con la búsqueda por subobjetivos

a) Utilizando como estado inicial el estado descrito en el ejercicio 2.9, define una serie de estados intermedios.

b) Realiza los árboles de la búsqueda por subobjetivos, utilizando como método base la búsqueda por profundidad limitada. Determina un limite de profundidad adecuado.

c) Analizar los árboles obtenidos y el árbol del ejercicio 2.9, para comparar la complejidad de los dos métodos.

Búsqueda por ascenso de colinas (hill climbing)

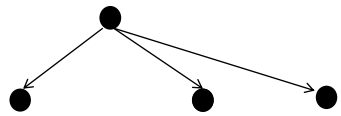
- Idea subyacente:
  - **Reducir profundidad del árbol expandido** ("on line search")
  - Realizar siempre la acción más prometedora
  - Expandir el árbol sólo un nivel y realizar la acción hacía el "mejor" nodo
  - Repetir el ciclo percepción/acción de forma continua

Repetir:
1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
4. realizar acción

Fin repetir

Búsqueda por ascenso de colinas (hill climbing)

- Idea subyacente:
  - **Reducir profundidad del árbol expandido** ("on line search")
  - Realizar siempre la acción más prometedora
  - Expandir el árbol sólo un nivel y realizar la acción hacía el "mejor" nodo
  - Repetir el ciclo percepción/acción de forma continua

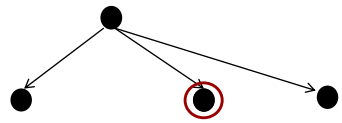


Repetir:
1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
4. realizar acción

Fin repetir

Búsqueda por ascenso de colinas (hill climbing)

- Idea subyacente:
  - **Reducir profundidad del árbol expandido** ("on line search")
  - Realizar siempre la acción más prometedora
  - Expandir el árbol sólo un nivel y realizar la acción hacía el "mejor" nodo
  - Repetir el ciclo percepción/acción de forma continua

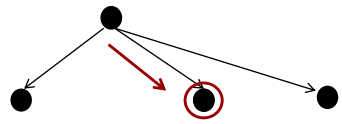


Repetir:
1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
4. realizar acción

Fin repetir

Búsqueda por ascenso de colinas (hill climbing)

- Idea subyacente:
  - **Reducir profundidad del árbol expandido** ("on line search")
  - Realizar siempre la acción más prometedora
  - Expandir el árbol sólo un nivel y realizar la acción hacía el "mejor" nodo
  - Repetir el ciclo percepción/acción de forma continua

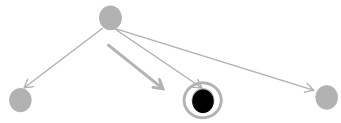


Repetir:
1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
4. realizar acción

Fin repetir

Búsqueda por ascenso de colinas (hill climbing)

- Idea subyacente:
  - **Reducir profundidad del árbol expandido** ("on line search")
  - Realizar siempre la acción más prometedora
  - Expandir el árbol sólo un nivel y realizar la acción hacía el "mejor" nodo
  - Repetir el ciclo percepción/acción de forma continua

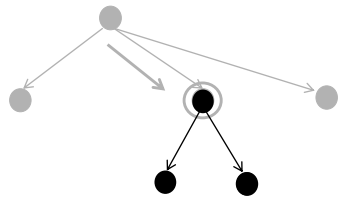


Repetir:
1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
4. realizar acción

Fin repetir

Búsqueda por ascenso de colinas (hill climbing)

- Idea subyacente:
  - **Reducir profundidad del árbol expandido** ("on line search")
  - Realizar siempre la acción más prometedora
  - Expandir el árbol sólo un nivel y realizar la acción hacía el "mejor" nodo
  - Repetir el ciclo percepción/acción de forma continua

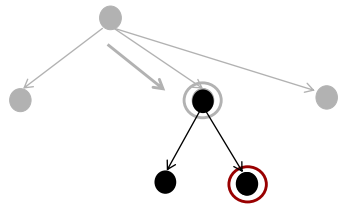


Repetir:
1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
4. realizar acción

Fin repetir

Búsqueda por ascenso de colinas (hill climbing)

- Idea subyacente:
  - **Reducir profundidad del árbol expandido** ("on line search")
  - Realizar siempre la acción más prometedora
  - Expandir el árbol sólo un nivel y realizar la acción hacía el "mejor" nodo
  - Repetir el ciclo percepción/acción de forma continua

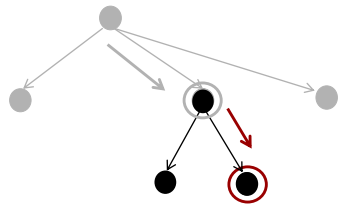


Repetir:
1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
4. realizar acción

Fin repetir

Búsqueda por ascenso de colinas (hill climbing)

- Idea subyacente:
  - **Reducir profundidad del árbol expandido** ("on line search")
  - Realizar siempre la acción más prometedora
  - Expandir el árbol sólo un nivel y realizar la acción hacía el "mejor" nodo
  - Repetir el ciclo percepción/acción de forma continua

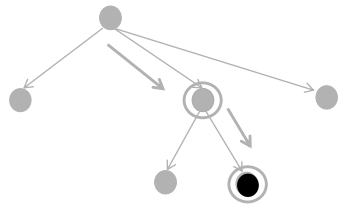


Repetir:
1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
4. realizar acción

Fin repetir

Búsqueda por ascenso de colinas (hill climbing)

- Idea subyacente:
  - **Reducir profundidad del árbol expandido** ("on line search")
  - Realizar siempre la acción más prometedora
  - Expandir el árbol sólo un nivel y realizar la acción hacía el "mejor" nodo
  - Repetir el ciclo percepción/acción de forma continua

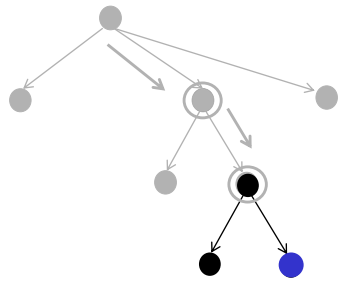


Repetir:
1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
4. realizar acción

Fin repetir

Búsqueda por ascenso de colinas (hill climbing)

- Idea subyacente:
  - **Reducir profundidad del árbol expandido** ("on line search")
  - Realizar siempre la acción más prometedora
  - Expandir el árbol sólo un nivel y realizar la acción hacía el "mejor" nodo
  - Repetir el ciclo percepción/acción de forma continua

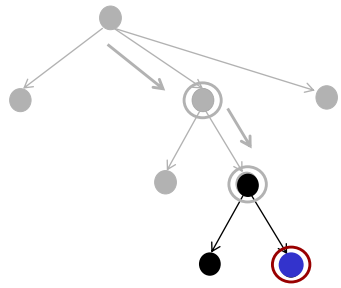


Repetir:
1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
4. realizar acción

Fin repetir

Búsqueda por ascenso de colinas (hill climbing)

- Idea subyacente:
  - **Reducir profundidad del árbol expandido** ("on line search")
  - Realizar siempre la acción más prometedora
  - Expandir el árbol sólo un nivel y realizar la acción hacía el "mejor" nodo
  - Repetir el ciclo percepción/acción de forma continua

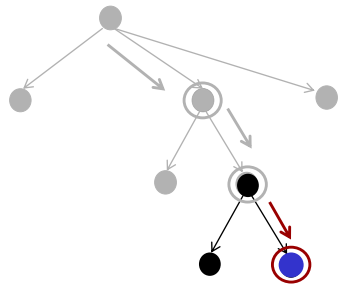


Repetir:
1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
4. realizar acción

Fin repetir

Búsqueda por ascenso de colinas (hill climbing)

- Idea subyacente:
  - **Reducir profundidad del árbol expandido** ("on line search")
  - Realizar siempre la acción más prometedora
  - Expandir el árbol sólo un nivel y realizar la acción hacía el "mejor" nodo
  - Repetir el ciclo percepción/acción de forma continua

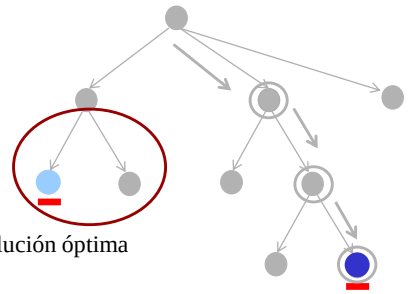


Repetir:
1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
4. realizar acción

Fin repetir

Búsqueda por ascenso de colinas (hill climbing)

- Idea subyacente:
  - **Reducir profundidad del árbol expandido** ("on line search")
  - Realizar siempre la acción más prometedora
  - Expandir el árbol sólo un nivel y realizar la acción hacía el "mejor" nodo
  - Repetir el ciclo percepción/acción de forma continua



Solución óptima

Repetir:
1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
4. realizar acción

Fin repetir

Algoritmo búsqueda por ascenso de colinas

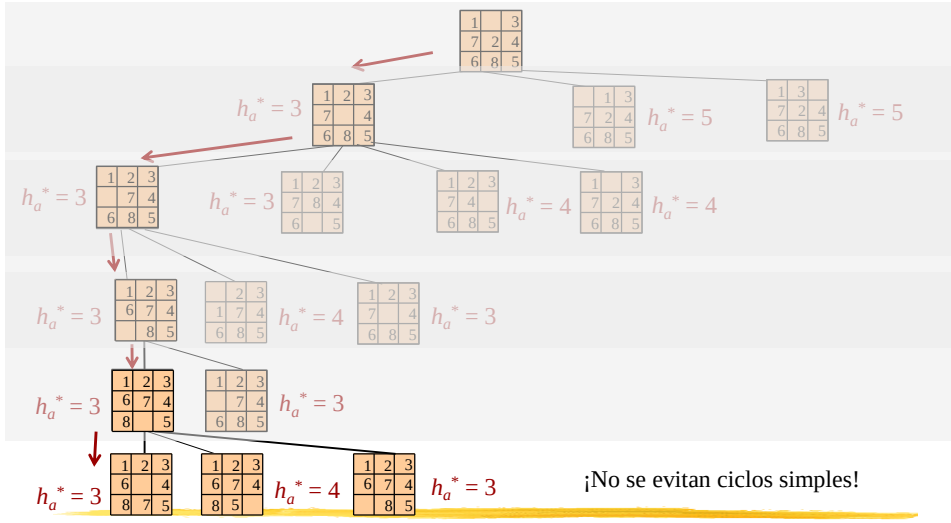
Algoritmo:

- *h**: función heurística que estima la distancia al nodo meta más cercano
- *percibir(entorno):* devuelve el estado actual del problema
- *evaluar(n,h*):* calcula el valor de la función heurística *h** para el nodo *n*
  - debería comprobar si n es nodo meta
  - en este caso debería devolver el mínimo valor posible
- *acción(n,m):*devuelve la acción que lleva de *n* a *m*
- *ejecutar(a, entorno):* efectúa la acción *a* en el entorno

{búsqueda ascenso de colinas}
Repetir
  nodo ← *percibir(entorno)*
  Si *meta?*(nodo) **entonces**
    devolver(positivo)
  sucesores ← *expandir*(nodo)
  Si *vacia?*(sucesores) **entonces**
    devolver(negativo)
  mejor ← $\arg\min_{n \in sucesores} [evaluar(n, h^*)]$
  a ←*acción(n,mejor)*
  ejecutar(a, entorno)
  Fin {repetir}

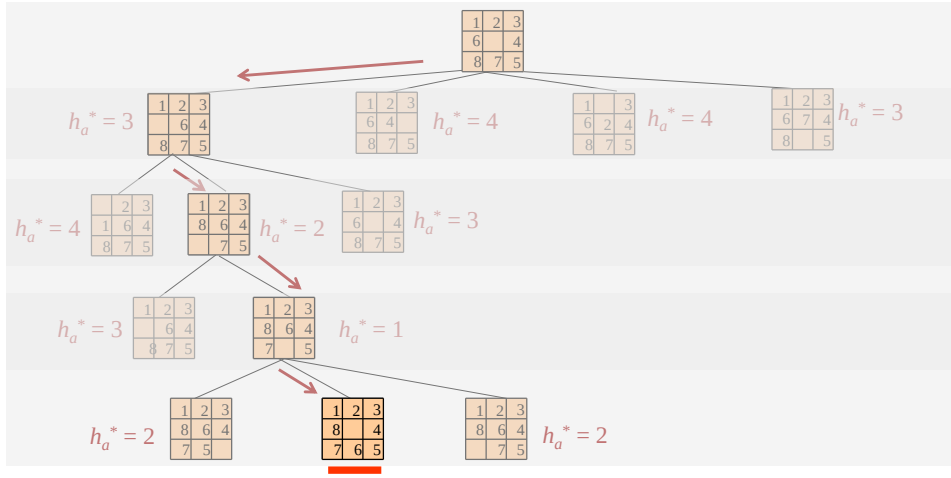
Búsqueda por ascenso de colinas: Ejemplo

Función heurística: h_a^* (piezas descolocadas)



Búsqueda por ascenso de colinas: Ejemplo (cont.)

Función heurística: h_a^* (piezas descolocadas)



Búsqueda por ascenso de colinas: complejidad

- Proporcional al número de nodos abiertos/expandidos
- *d* profundidad de la mejor solución, *b* factor de ramificación efectivo

- Complejidad en espacio:
  - *O(b)* solo se mantiene en memoria los hijos de un nodo
- Complejidad en tiempo:
  - depende de la función heurística
  - hay que contar la complejidad de calcular *h**
  - si *h*(n)=h(n):*
    - la búsqueda "va directamente a la solución"
    - caso irreal, ya que significa que se conozca la solución
    - complejidad en tiempo: O(d)
  - otros casos:
    - puede ser mayor que en la búsqueda por amplitud (si la función heurística es mala)

Búsqueda por ascenso de colinas: análisis

Análisis:

- Suele obtener buenos resultados en algunos casos, especialmente:
  - si no hay ciclos
  - la función heurística es muy buena
- En general, no se puede asegurar optimalidad ni completitud
- No se puede evitar ciclos simples
- Si h* es ideal (h*=h), entonces es óptimo y completo

- Se puede conseguir completitud:
  - Si el conjunto de posibles estados es finito
  - Si se guardan todos los estados a los que se ha ido en el pasado:
    - cada vez que se llega a un estado repetido, se elige una acción que no se ha elegido anteriormente
    - pero, para eso hay que guardar todo el árbol y la ganancia en complejidad se pierde

Búsqueda por ascenso de colinas: comentarios

Comentarios:

- Aplicable en entornos inaccesibles o dinámicos
  - Ejemplos: Laberinos, Robot moviéndose en un entorno real, …
- Útil en tareas de optimización, donde se quiere mejorar el estado continuamente
  - Ejemplo: Controlar los parámetros de procesos industriales

Posibles modificaciones:

- Se podría evitar ciclos simples guardando el último estado conocido
- Buscar un camino completo:
  - Realizar el algoritmo "off line"
  - Solo simular las acciones pero no efectuarlas en el entorno
  - No intercalar la búsqueda con el ciclo percepción/acción
  - No es posible en entornos inaccesibles o dinámicos

Ejercicio 2.12

Búsqueda por ascenso de colinas

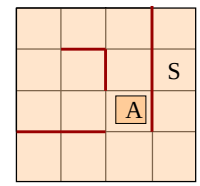
Al lado se muestra el plano de una habitación. El agente (A) tiene que encontrar un camino a la salida (S). En cada acción, el agente puede moverse a un cuadro adyacente (no diagonal), si no existe un obstáculo que lo impida. El agente conoce todo el plano de antemano.

a) Resuelve el problema mediante la búsqueda por ascenso de colinas (evitando ciclos simples).

b) Resuelve el problema mediante la búsqueda voraz (evitando ciclos simples).

c) Suponiendo que se evitan ciclos simples, ¿Qué diferencias hay entre estos dos métodos?

d) ¿Qué ocurre (en ambos métodos) si no se evitan los ciclos simples?



Búsqueda con horizonte

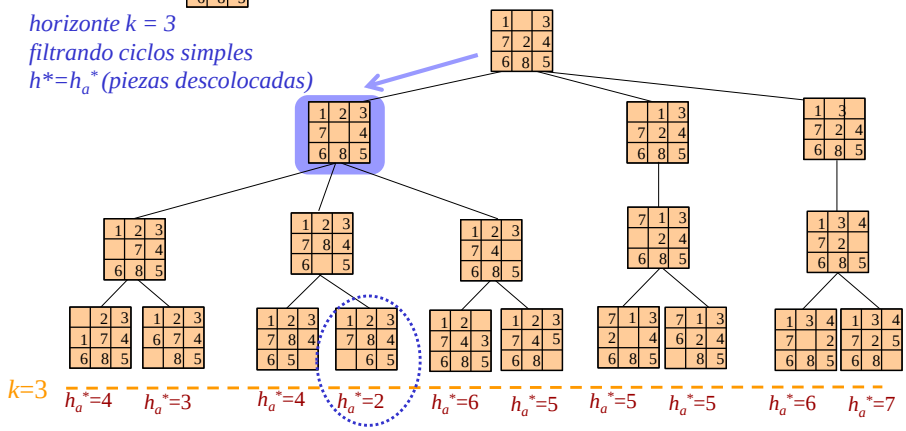
- Idea subyacente:
  - la primera acción de un plan que lleva a un nodo con una evaluación heurística óptima, tiene una buena *probabilidad* de pertenecer al camino óptimo
  - **Reducir profundidad del árbol expandido** ("on line search")
  - Expandir el árbol hasta un nivel *k* (horizonte) y realizar la acción hacía el "mejor" nodo en el nivel *k*
  - Repetir el ciclo percepción/acción de forma continua
- Algoritmo general:
  - Percibir el estado actual *s*
  - Aplicar un método de *búsqueda no informado* hasta el nivel *k* y nodos metas.
  - Sea H el conjunto de nodos hoja hasta el nivel k:
    - Utilizar *h** para determinar el "mejor" nodo hoja $n^* \in H$: $n^* \leftarrow \arg\min_{n \in H} [h^*(n)]$
  - Ejecutar la primera acción a^* en el camino que lleva a n^*
  - Repetir hasta que el agente se encuentra en un estado meta

Búsqueda con horizonte: Ejemplo

método base: búsqueda en profundidad limitada

estado actual =

horizonte k = 3
filtrando ciclos simples
h=h_a^* (piezas descolocadas)*

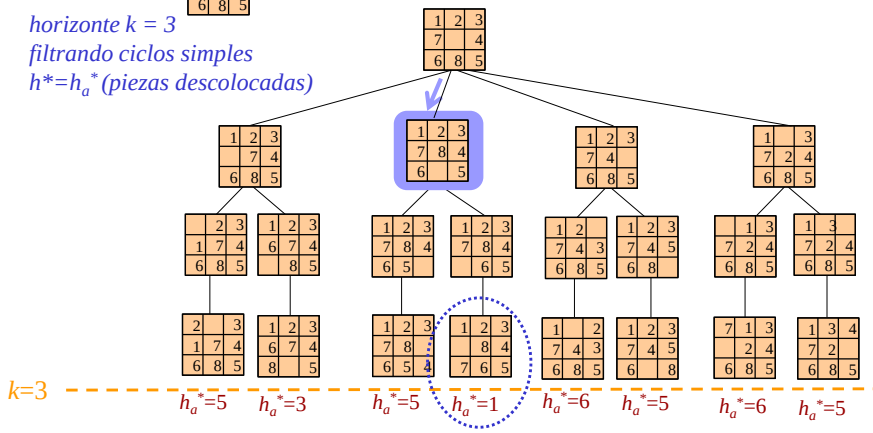


Búsqueda con horizonte: Ejemplo

método base: búsqueda en profundidad limitada

estado actual =

horizonte k = 3
filtrando ciclos simples
h=h_a^* (piezas descolocadas)*

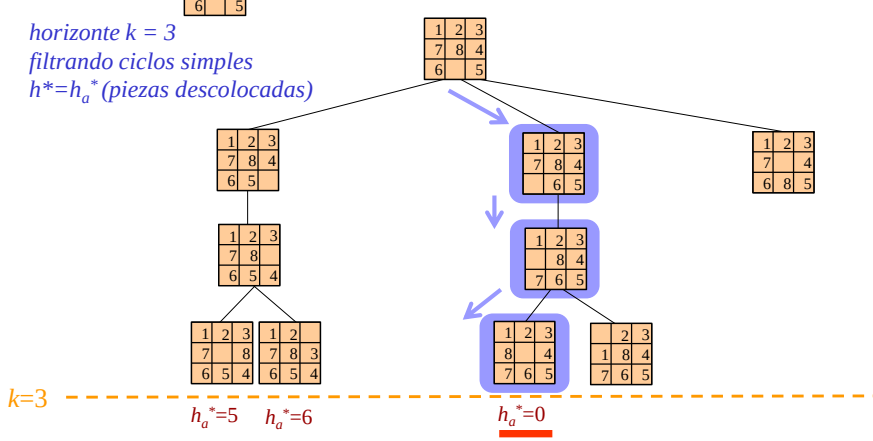


Búsqueda con horizonte: Ejemplo

método base: búsqueda en profundidad limitada

estado actual =

horizonte k = 3
filtrando ciclos simples
h=h_a^* (piezas descolocadas)*



Búsqueda con horizonte: complejidad

- Proporcional al número de nodos abiertos/expandidos
- *b* factor de ramificación efectivo, *k* limite de profundidad
- Método base: búsqueda en profundidad limitada

- Complejidad en espacio:
  - $(k*b)+1 \in O(b*k)$ (solo se mantiene el camino explorado y sus vecinos en la memoria)
- Complejidad en tiempo:
  - el encontrar un nodo meta y su profundidad depende de la función heurística
  - hay que contar la complejidad de calcular *h**

Búsqueda con horizonte: complejidad

Complejidad en tiempo:

- si se encuentra una solución con profundidad $d \leq k$:
  - igual que en la búsqueda en profundidad limitada:
  - mejor caso: $d \in O(d)$
  - peor caso: $1+b+...+b^{k-1} \in O(b^k)$
- si se encuentra una solución con profundidad $d > k$:
  - primero hay que realizar $d-k$ búsquedas en profundidad limitadas completos
  - en el siguiente paso se encuentra la solución en la profundidad k
  - mejor caso: $(d-k)(1+b+b^2+\ldots+b^{k-1})+k = (d-k)\frac{(b^k-1)}{b-1}+k \quad \in O(d \cdot b^k)$
  - peor caso: $(d-k+1)(1+b+b^2+\ldots+b^{k-1}) = (d-k+1)\frac{(b^k-1)}{b-1} \quad \in O(d \cdot b^k)$

Búsqueda con horizonte : análisis

Análisis:

- En general, no se puede asegurar optimalidad ni completitud
  - depende de la función heurística
- Si h* es ideal (h*=h), entonces es óptimo y completo
- La búsqueda por ascenso de colinas es un caso especial de la búsqueda con horizonte (horizonte = 1)
- Aumentar el horizonte *k*:
  - permite evitar ciclos de longitud *k* o menor
  - se trata mejor el caso en los que varias acciones tienen el mismo valor heurístico (ver ejemplo)

Búsqueda online y optimización

La búsqueda online (horizonte/ascenso de colinas) es aplicable a problemas de optimización.

Problema de optimización:

- El objetivo es encontrar el mejor estado según una *función objetivo*
- No necesariamente se busca un estado exacto, sino uno que se acerca al máximo al objetivo
- El camino para llegar al estado buscado puede ser irrelevante (coste 0)
- Ejemplos: juegos lógicos y de configuración, n-Reinas

Funciones heurísticas para problemas de optimización:

- Funciones que estimen el coste del camino hasta el nodo meta más próximo (todos los que hemos visto hasta ahora)
- Funciones que miden la calidad de un nodo respecto a la función objetivo
- Muchas funciones heurísticas estiman las dos cosas a la vez
- A veces es más fácil definir una heurística que estime la "calidad" de los nodos

Búsqueda online y optimización: El juego de las cifras

El juego de las *cifras:*

- Combinar una serie de CIFRAS mediante las operaciones de suma y multiplicación de tal forma que el resultado se acerca lo máximo a un número dado (EXACTO).
- Ejemplo:

```
CIFRAS:     6     2     5     25
EXACTO:     420
```

- El objetivo consiste en encontrar una secuencia de *operaciones*, que aplicadas sobre (algunas de) las cifras de entrada, y sobre los resultados intermedios, permiten obtener un número lo más próximo a EXACTO. Cada cifra se puede utilizar sólo una vez.

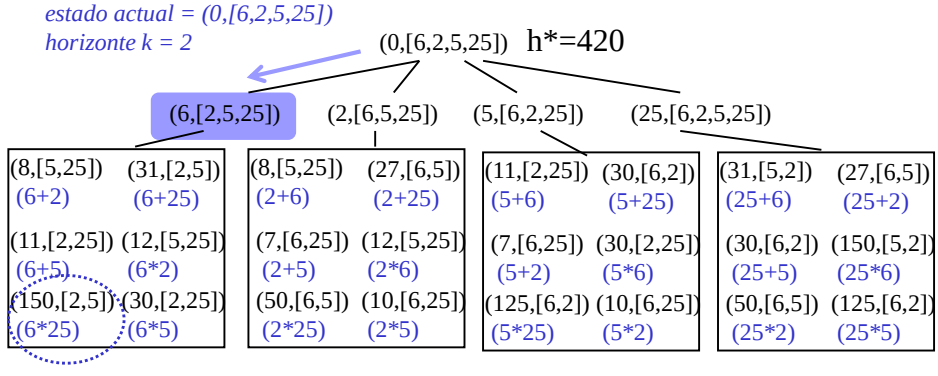
Búsqueda online y optimización: El juego de las cifras

- **Formalización del problema:**
  - Representación (eficiente) de estados:
    (<último resultado>,<lista de los números disponibles todavía>)
  - Estado inicial: (0,[6,2,5,25])
  - Estado meta: (x,y) tal que x es lo más cerca posible de 420
  - Operadores: aplicar la suma/multiplicación
  - Coste de un operador: *0*
  - Tipo de solución: estado
- Se puede resolver el problema con un método de búsqueda no informado usando como estado meta (420,y), pero ¿Que pasa si no hay ninguna solución exacta?
- Mejor utilizar la búsqueda online:
  - Función heurística: h*((x,y))=|420-x|
    - mide la similitud del estado al objetivo, **no mide el coste del camino restante**
  - Modificar los algoritmos:
    - se termina cuando el estado elegido como más prometedor tiene un valor de h* mayor que el estado actual

Búsqueda con horizonte: El juego de las cifras

método base: búsqueda en profundidad limitada

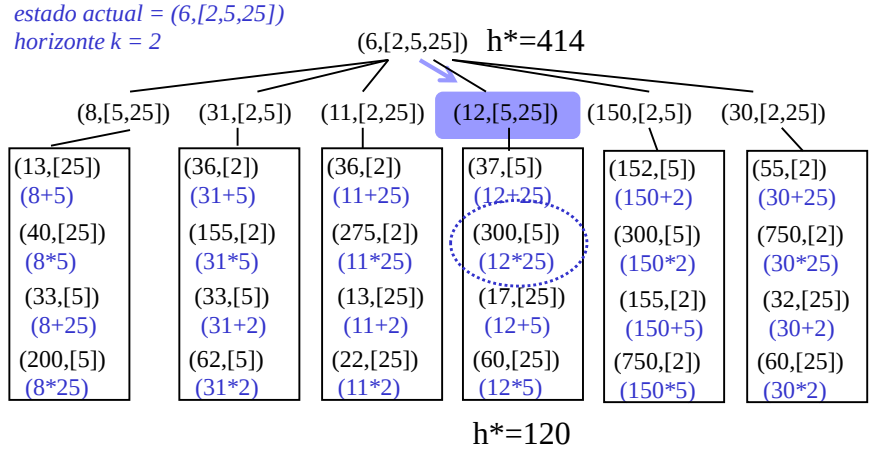
estado actual = (0,[6,2,5,25])
horizonte k = 2



Búsqueda con horizonte: El juego de las cifras

método base: búsqueda en profundidad limitada

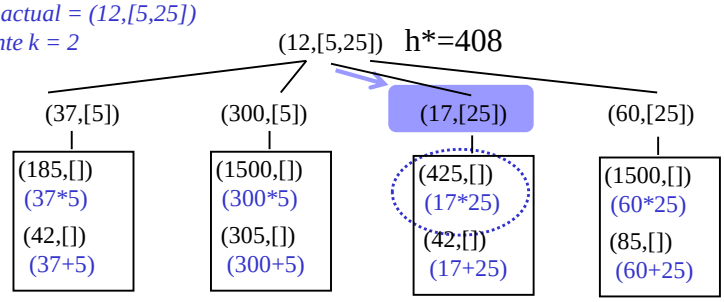
estado actual = (6,[2,5,25])
horizonte k = 2



Búsqueda con horizonte: El juego de las cifras

método base: búsqueda en profundidad limitada

estado actual = (12,[5,25])
horizonte k = 2



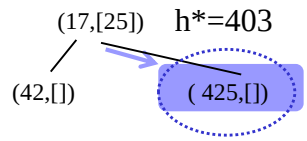
h*=5

k=2

Búsqueda con horizonte: El juego de las cifras

método base: búsqueda en profundidad limitada

estado actual = (17,[25])
horizonte k = 2



h*=5

k=2

Búsqueda con horizonte: El juego de las cifras

método base: búsqueda en profundidad limitada

estado actual = (425,[])
horizonte k = 2

(425,[])

k=2

Ejercicio 2.13

Búsqueda online para optimización: Juego de las cifras

- La búsqueda online para optimización (parando cuando no se obtiene ninguna mejora) **es muy útil en problemas de optimización con un número infinito de estados**.

- Aplica la búsqueda por ascenso de colinas a la siguiente instanciación del juego de las cifras:

  CIFRAS: 2, 3, 5, 7, 8

  EXACTO: 163

- En este caso, se usan los operadores suma, resta, multiplicación y división. Solo se puede aplicar un operador si el resultado es entero positivo. Cada cifra se puede utilizar varias veces.

Tema 2: Juegos unipersonales

Resumen:

2. Juegos unipersonales
   - 2.1. Representación básica
   - 2.2. Juegos con información completa
   - 2.3. Recursos limitados en juegos con información completa
   - 2.4. Juegos con información incompleta

     Búsqueda en tiempo real
     Búsqueda A* con aprendizaje en tiempo real

Características de problemas con información incompleta

- Entorno:
  - secuencial (acciones se efectúan de forma secuencial)
  - discreto (número finito de acciones en cada estado)
  - determinista (resultados de las acciones son estados definidos)
  - accesible (se puede percibir el estado actual y comprobar si es estado meta)

- Agente:
  - Es capaz de percibir el estado en el que se encuentra
  - Conoce las acciones que puede aplicar en el estado actual
    - Pero no conoce los estados sucesores de un estado hasta que no ha probado las acciones correspondientes (no puede prever los estados resultantes de las aciones)
  - Tiene un objetivo (estado meta)
  - Suposiciones:
    - Las acciones son deterministas (el resultado de una acción, aunque no es previsible, está claramente definido y no cambia)
    - El agente puede reconocer siempre un estado que ha visitado anteriormente

Conocimientos mínimos a priori de un agente

- Conocimientos mínimos a priori de un agente de búsqueda en el espacio de estados:
  - s_0 — Estado *inicial*
  - *acciones*: $s \mapsto \{a_1, ..., a_n\}$ — Devuelve una lista de acciones permitidas en el estado *s*
  - *meta?*: s $\mapsto$ verdad | falso — Compara el estado *s* con los estados meta y devuelve *verdad* si *s* es un estado meta
  - c: $(s_i, a, s_j) \mapsto v$, $v \in \aleph$ — *Coste* del operador *a* para ir de s_i a s_j solo se puede aplicar si se conoce s_j
  - $c(s_{i_1} s_{i_2} \ldots s_{i_n}) = \sum_{k=1}^{n-1} c(s_{i_k}, a_k, s_{i_{k+1}})$ — *Coste* de un plan

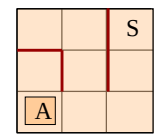
Ejemplo base: Laberinto

Problema:

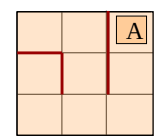
- **Estados**:
  - posición del agente en el laberinto
- **Acciones:**
  - el agente (A) se puede mover a una posición adyacente si no hay un obstáculo
- **Coste:**
  - La aplicación de cada operador vale *una* unidad
- **Solución:**
  - se busca un camino (más corto) a la salida (S)
- **Suposición:**
  - el agente no conoce el laberinto
  - es capaz de recordar estados visitados anteriormente

Instanciación:

Estado inicial



Estado meta



Búsqueda (en profundidad) en tiempo real

Idea:

- Realizar una búsqueda en profundidad creando un mapa del problema
- Se crea sucesivamente el grafo del problema de búsqueda guardando el grafo en memoria
- Si se llega a un estado que ya se ha llegado antes, se explora otros caminos no explorados
- Si se llega a un estado en el que se han explorado todas las acciones se vuelve (físicamente) atrás

Búsqueda en tiempo real

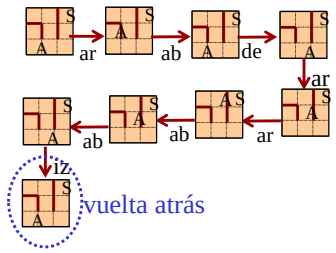
Algoritmo:

- G: es el grafo etiquetado y dirigido del problema que se genera al explorar el espacio; inicialmente vacío
- ha: es una lista de los estados recorridos, necesario para realizar una vuelta atrás
- añade(s,G): añade el nodo *s* al grafo *G* y añade sus acciones como arcos "abiertos"
- conecta(s_a,a,s,G): conecta el arco con etiqueta *a* del nodo s_a al nodo *s*
- s_a y *a*: estado anterior y última acción
- arc_ab[s]: lista de arcos "abiertos" de *s*
- *Primero(l):* devuelve el primer elemento de una lista y lo quita de la lista
- *percibir(entorno)* y *realizar(a):* percibe el estado actual y realizar la acción a

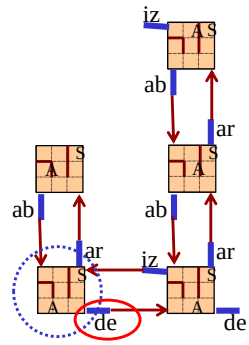
```
{búsqueda en profundidad en TR}
s_a=null
Repetir
  s ← percibir(entorno)
  Si meta?(s) entonces devolver(parar)
  Si nuevo?(s) entonces añade(s,G)
  Si s_a no es nulo entonces
     conecta(s_a,a,s,G)
     añade s_a al principio de ha
  Si vacio?(arc_ab[s]) entonces
     Si vacio?(ha) entonces devolver(parar)
     En caso contrario
        a ← acción que va de s a Primero(ha)
        s_a ←nulo
  En caso contrario a ← Primero(arc_ab[s])
     s_a ←s
  entorno ← realizar(a)
Fin {repetir}
```

Búsqueda en tiempo real: Ejemplo

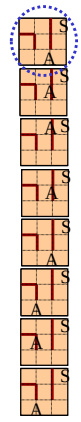
Movimientos del agente



grafo G



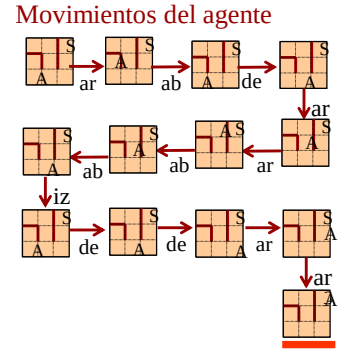
ha



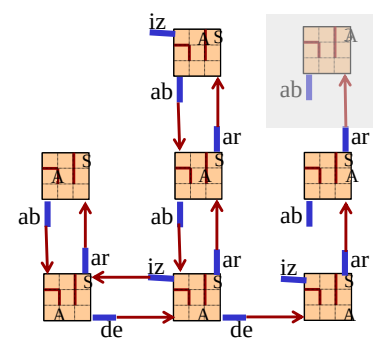
Orden de realizar los movimientos si hay varias posibilidades:
- arriba, abajo, izquierda, derecha

Búsqueda en tiempo real: Ejemplo

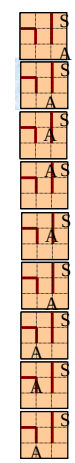
Movimientos del agente



grafo G



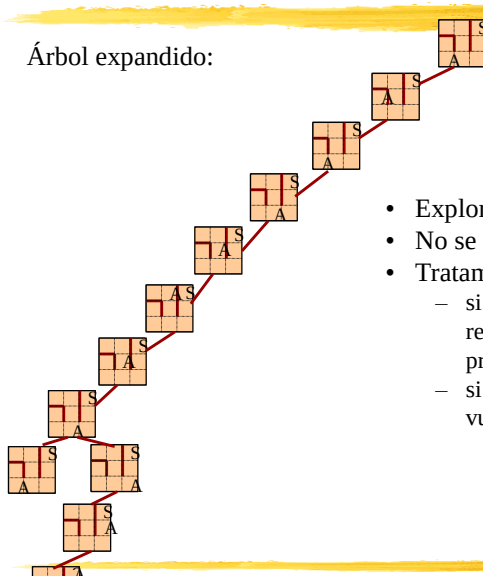
ha



Orden de realizar los movimientos si hay varias posibilidades:
- arriba, abajo, izquierda, derecha

Búsqueda en tiempo real: Ejemplo

Árbol expandido:

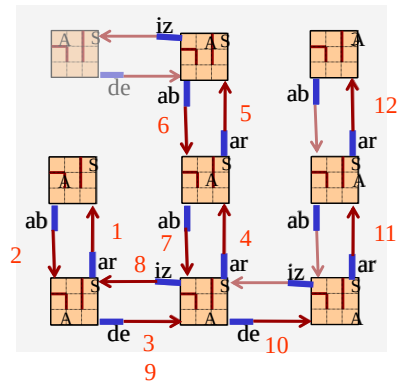


- Exploración por profundidad
- No se evitan ciclos
- Tratamiento de estados repetidos:
  - si hay un estado repetido se realiza la acción que no se ha probado todavía
  - si no hay acciones sin probar vuelve atrás

Búsqueda en tiempo real: Ejemplo

Camino del agente el el laberinto:

- los números en rojo indican el camino del agente



Búsqueda en tiempo real: Analisis

Complejidad:

- *n* número de estados posibles; *b* factor de ramificación
- en espacio: espacio requerido para el grafo G y la lista *ha*
  - *peor caso: (n-1)(b+1)+(n-1)b=2nb+n - 2b -1 ∈O(nb)*
- en tiempo: número de movimientos del agente
  - *peor caso: O(nb)*
- en el peor caso b=n y la complejidad en tiempo y espacio podría ser $O(n^2)$
- $O(n^2)$ puede ser un problema, especialmente respecto al espacio

Búsqueda en tiempo real: Analisis

Completitud:

- es completo: en espacios finitos y si de todos los nodos a que se puede llegar existe un camino al nodo meta

Optimalidad:

- No es óptimo
- Índice competitivo del agente (coste camino real/coste camino óptimo)
  - puede ser infinito si hay acciones irreversibles o el espacio es infinito
  - puede ser arbitrariamente largo si todas las acciones son reversibles y el espacio es limitado
- Ningún algoritmo puede evitar callejones sin salida en todos los espacios de estados

Ejercicio 2.14

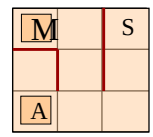
Búsqueda en profundidad en tiempo real:

Considera el laberinto dibujado abajo. A es el agente cuyo objetivo es llegar a S. M es un monstruo que come el agente si llega a el.

a) Realiza la búsqueda por profundidad en tiempo real. Si en un estado hay varios acciones sin explorar, elige las acciones en el siguiente orden: **izquierda, arriba, derecha, abajo**. Dibuja el grafo que genera el agente e indica sus movimientos.

b) Realiza la búsqueda por profundidad en tiempo real. Si en un estado hay varios acciones sin explorar, elige las acciones en el siguiente orden: **derecha, arriba, izquierda, abajo**. Dibuja el grafo que genera el agente e indica sus movimientos.

c) Suponiendo un coste de cada movimiento de 1, calcula el índice competitivo del agente para ambos casos y compáralos.



Búsqueda A* con aprendizaje en tiempo real

Problema:

- La búsqueda en profundidad en tiempo real es esencialmente ciega:
  - El agente intenta caminos al azar y evita caminos ya explorados
- Se podría utilizar funciones heurísticas:
  - para explorar las acciones más prometedoras primero
  - Pero, ¿Qué heurística se puede usar si no se conoce mucho del problema?

Idea: generar información heurística "sobre la marcha"

- Crear el mapa del problema y *aprender* al mismo tiempo los valores de h* para cada estado (y durante varias búsquedas sucesivas)
- Inicialmente, se inicializa $h^*(s) = 0$ para todos los nodos s
- Después de realizar una acción en el estado s_a se actualiza $h^*(s_a)$ al coste mínimo para ir de s_a al estado meta, según lo que se sabe en este momento
- Estando en un estado s, se elige aquella acción permitida b que lleva a un estado s' tal que $h^*(s')+c(s,b,s')$ es mínimo

Búsqueda A* con aprendizaje en tiempo real

Algoritmo:

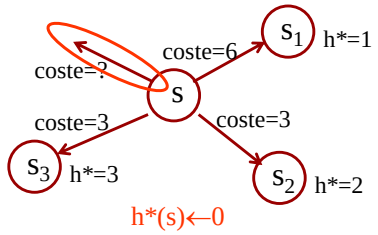
- G: es el grafo etiquetado y dirigido del problema que se genera al explorar el espacio; inicialmente vacío; contiene un valor *h** para cada nodo
- añade(s,G): añade el nodo *s* al grafo *G* y añade sus acciones como arcos "abiertos"; inicializa *h*(s)* a 0
- conecta(s_a,a,s,G): conecta el arco con etiqueta *a* del nodo s_a al nodo *s*
- s_a y *a*: estado anterior y última acción; son variables globales
- estado[s,b]: estado al que se llega del estado *s* con la acción *b* (puede ser no definido)
- funcion costo (s,s',b):
  {calcula el coste estimado para ir de s al nodo meta más cercano a través de s' }
  Si nodefinido?(s') **entonces devolver** 0
  en otro caso devolver c(s,b,s')+H[s']

{búsqueda con aprendizaje en TR}
s_a=null
Repetir
 s ← *percibir(entorno)*
 Si nuevo?(s) **entonces** añade(s,G)
 Si s_a no es nulo **entonces**
  conecta(s_a,a,s,G)
  $h^*[s_a] \leftarrow \min_{b \in acciones(s_a)} (costo(s_a, estado[s_a,b], b))$
 Si meta?(s) **entonces devolver(**parar)
 a ← acción b de *acciones(s)* que minimiza costo(s,estado[s,b],b)
 entorno ← *realizar(a)*
 s_a ←s
Fin {repetir}

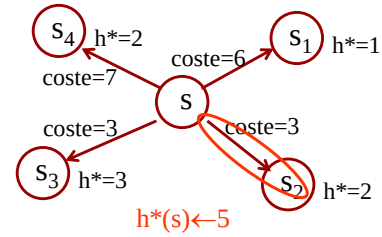
Coste estimado mínimo para ir de un estado a la meta

Caso 1: (s tiene acciones sin explorar) Caso 2: (s ha explorado todas las acciones)

Actualizar h* de s:

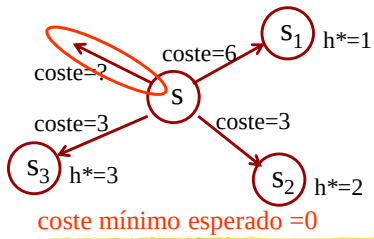


h*(s)←0

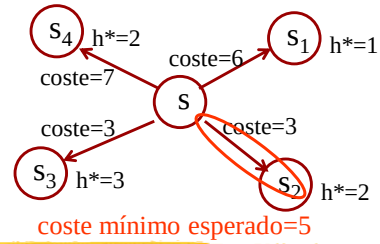


h*(s)←5

Elegir acción desde s:



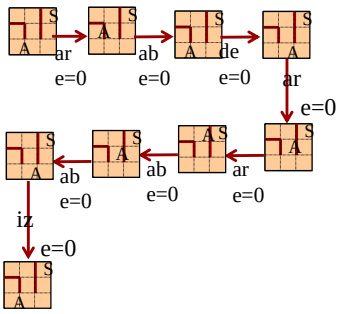
coste mínimo esperado =0



coste mínimo esperado=5

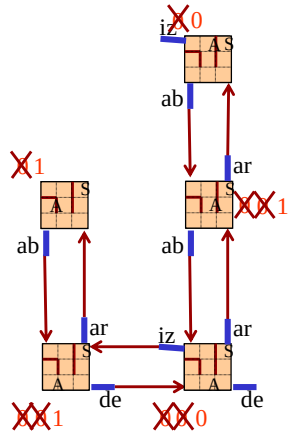
Búsqueda A* con aprendizaje en tiempo real: Ejemplo

Movimientos del agente



- Orden de realizar los movimientos si hay varias posibilidades:
  - arriba, abajo, izquierda, derecha
- e es el costo estimado de la acción para ir a la meta

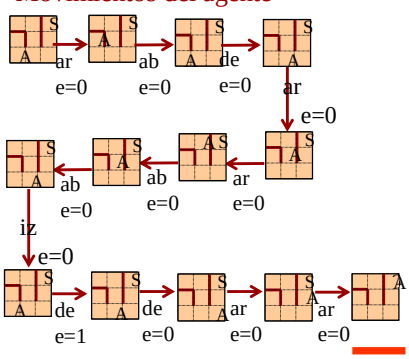
grafo G



- en rojo los valores de h*
- el coste de cada acción es 1

Búsqueda A* con aprendizaje en tiempo real: Ejemplo

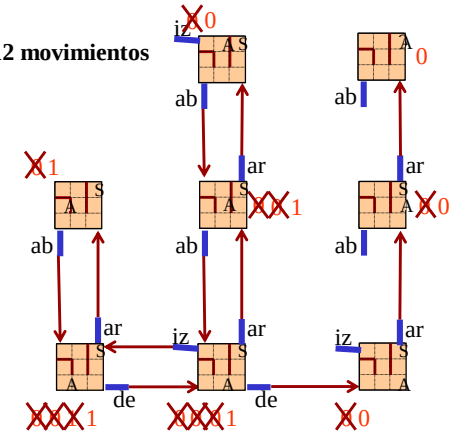
Movimientos del agente



- Orden de realizar los movimientos si hay varias posibilidades:
  - arriba, abajo, izquierda, derecha
- e es el costo estimado de la acción para ir a la meta

grafo G

12 movimientos



- en rojo los valores de h*
- el coste de cada acción es 1

Búsqueda A* con aprendizaje en tiempo real: Analisis

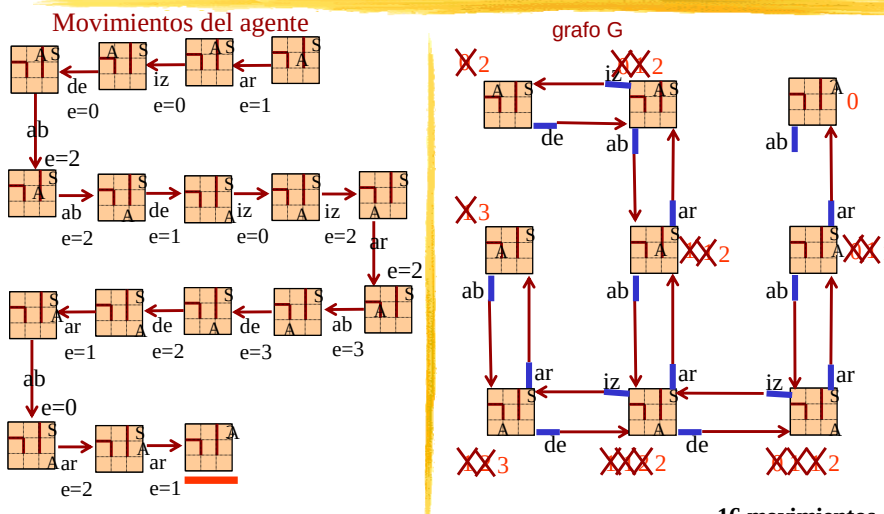
Aprendizaje de h*:

- los valores de h* aprendidos sirven para nuevas instanciaciones del problema:
  - el estado meta tiene que ser el mismo
  - el estado inicial puede ser distinto
- después de un número suficiente de búsquedas sucesivas con el mismo estado meta, h* converge a h
  - se convierte en una estimación exacta del coste de un nodo a la meta

Problemas:

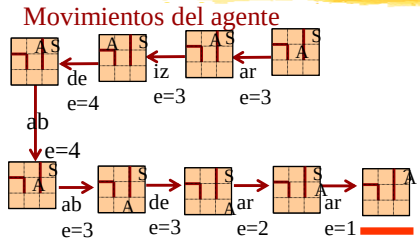
- En situaciones donde el estado meta puede cambiar, los valores de h* aprendidos no sirven para nuevas búsquedas

Nueva búsqueda (con h* aprendido; mismo estado meta /otro estado inicial)

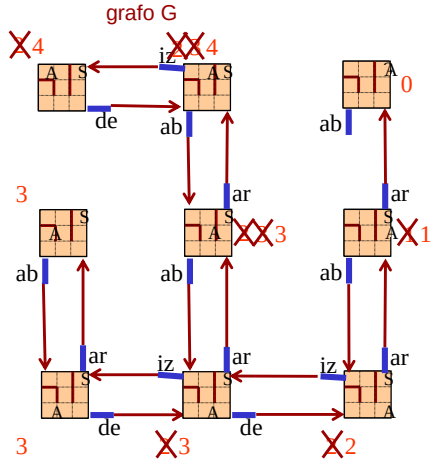


- Condiciones: igual que antes (sólo cambia el estado inicial)

Nueva búsqueda (con h* aprendido; mismo estado meta / estado inicial que antes)

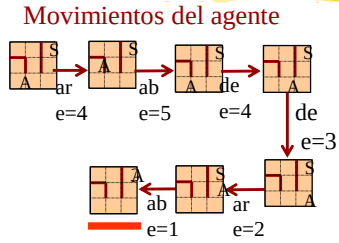


8 movimientos

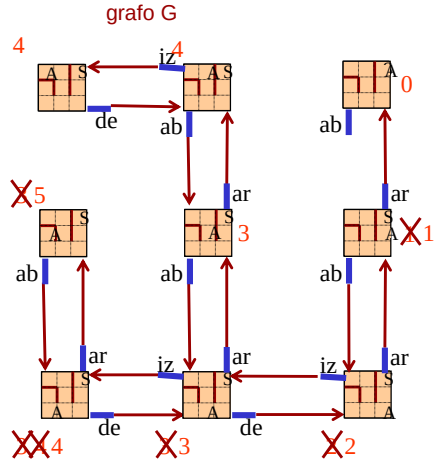


• Condiciones: igual que antes

Nueva búsqueda (búsqueda inicial con h* aprendido)



6 movimientos (comparado con los 12 de la primera búsqueda)



• Condiciones: igual que antes

Búsqueda A* con aprendizaje en tiempo real: Analisis

Complejidad:

- espacio: la misma que en la búsqueda en profundidad en tiempo real: $O(n \cdot b)$ (para guardar el grafo)
- tiempo:
  - inicialmente igual que en la búsqueda en profundidad en tiempo real: $O(n^2)$ (peor caso)
  - mejora con el tiempo al ser la función heurística h* cada vez más exacta

Completitud:

- es completo: en espacios finitos y si de todos los nodos a que se puede llegar existe un camino al nodo meta

Optimalidad:

- No se es óptimo
- Se convierte en óptimo, cuando h*(n)=h(n) para todos los nodos n

Problemas:

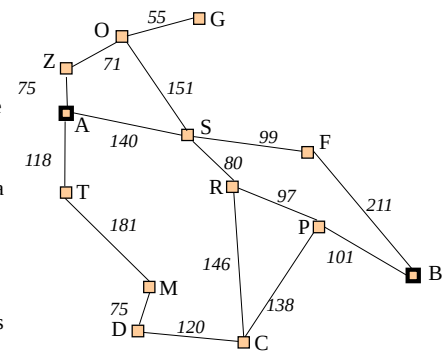
- Los requerimientos de espacio son importantes, especialmente para problemas con un número grande de estados distintos
- Ejemplo: el 8-Puzzle tiene 9!/2=181440 estados (15-Puzzle: 10^{13}estados)

Ejercicio 2.15

Búsqueda A* con aprendizaje en tiempo real:

Imagínate el siguiente escenario. Un agente tiene que realizar una carrera de la ciudad A a la ciudad B (ver mapa de carreteras con ciudades y kilómetros). La peculiaridad de la carrera consiste en que el agente no conoce el mapa. Una vez que el agente llega a una ciudad, se le dice el nombre de esta ciudad, las carreteras que salen y la longitud de la carretera que acaba de recorrer.

a) Aplique el algoritmo de búsqueda A* con aprendizaje en tiempo real a este problema. Si no hay ningun criterio mejor, el agente prefiere carreteras que van al norte y luego sucesivamente los que van al este, sur y oeste.
b) Repite la misma búsqueda con los valores de la función heuristica aprendido anteriormente.
c) Que cambiaría si, al llegar a una ciudad nueva, no sólo se le dice al agente las carreteras que salen, sino también los kilómetros que hay hasta la siguiente ciudad para cada una de estas carreteras (pero no a que ciudades se llegan).



Ejercicio 2.16

Aprendizaje de la función heurística en problemas con información completa:

El mecanismo de aprendizaje puede aplicarse igual si se conoce el grafo de antemano (problemas con información completa). De hecho, una vez que se haya obtenido el grafo completo, se hace eso. Además, en vez de inicializar los valores de la función heurística a 0, se puede utilizar como valores iniciales los de una función heurística conocida.

a) Implemente el algoritmo de aprendizaje para el 8-puzzle. Para inicializar los valores de la función heurística para cada estado utiliza la función h_c*.
b) Genere 100 estados iniciales aleatorios y busque una solución con el algoritmo implementado, y aprendiendo (mejorando) la función heurística a lo largo de estas 100 búsquedas. Utilice como estado meta el estado meta utilizado en los ejemplos.
c) Genera un gráfico que indica el número de movimientos realizados en cada búsqueda en función del número de búsquedas realizados anteriormente. Comenta este gráfico.