Tema 3

“Búsqueda Heurística”

Año académico 2019/20

Profesores:
Alberto Fernández, Holger Billhardt

Heurísticas

Heurística (griego: heuriskein): “encontrar”, “descubrir”

Inteligencia Artificial:

- compila conocimiento “empírico” sobre un problema / un entorno

Interpretación “fuerte”:

- una heurística *suele facilitar* la resolución de un problema, pero *no garantiza* que se resuelva
- búsqueda: optimalidad o incluso completitud no garantizados

Interpretación “débil”:

- método riguroso + información heurística
- información heurística puede mejorar el *rendimiento medio* de un método de resolución de problemas, pero no garantiza una mejora en el *peor caso*
- búsqueda: mejora de complejidad no garantizado

Funciones heurísticas

Funciones heurísticas para búsqueda en el espacio de estados:

- estiman la adecuación de un nodo para ser expandido
- métodos de búsqueda "*el mejor primero*" eligen el nodo más prometedor para expandir

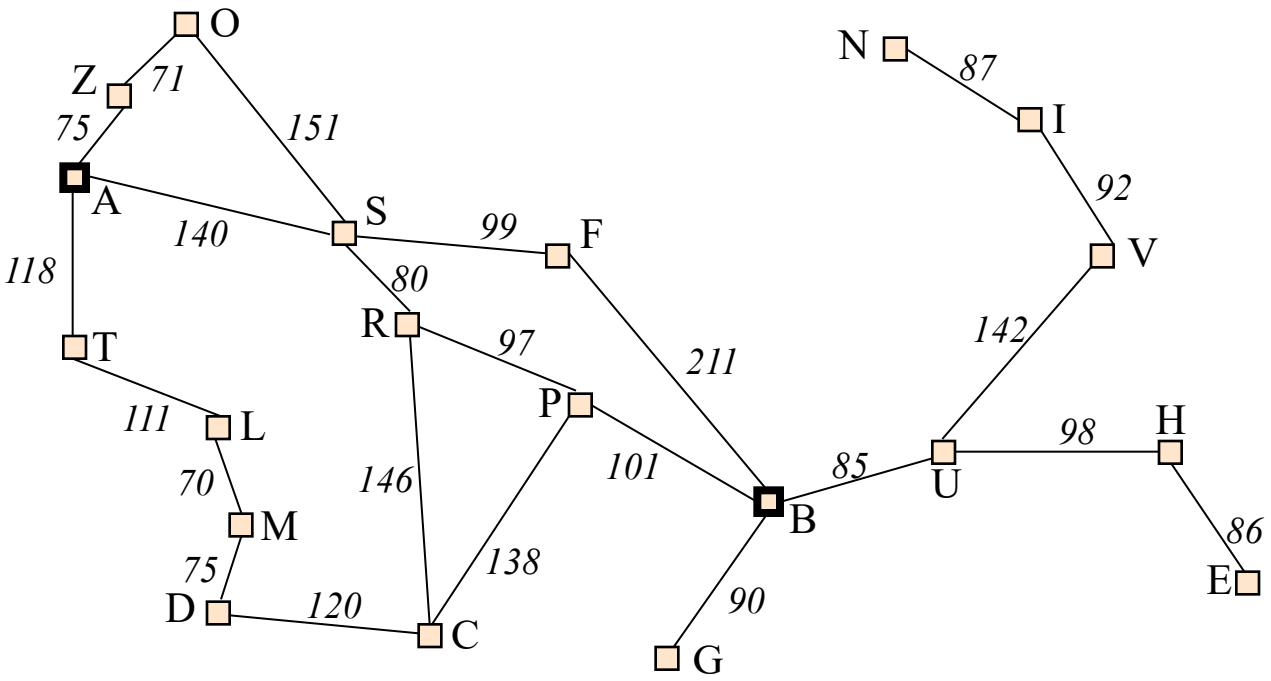
Heurística usual: "distancia" hacia la meta

- $h : N \rightarrow R$ mide el coste *real* desde el nodo n hasta el nodo meta más cercano
- $h^* : N \rightarrow R$ es una función *heurística* que estima el valor de $h(n)$
- una función heurística h^* es *optimista*, si $h^*(n) \leq h(n)$ para todo nodo n

Ejemplos de funciones heurísticas optimistas:

- mundo de los bloques: número de bloques *descolocados*
- encontrar rutas: distancia en *línea recta* hasta un nodo meta

Función heurística para encontrar rutas



	h^*
A	366
B	0
C	160
D	242
E	161
F	178
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	98
R	193
S	253
T	329
U	80
V	199
Z	374

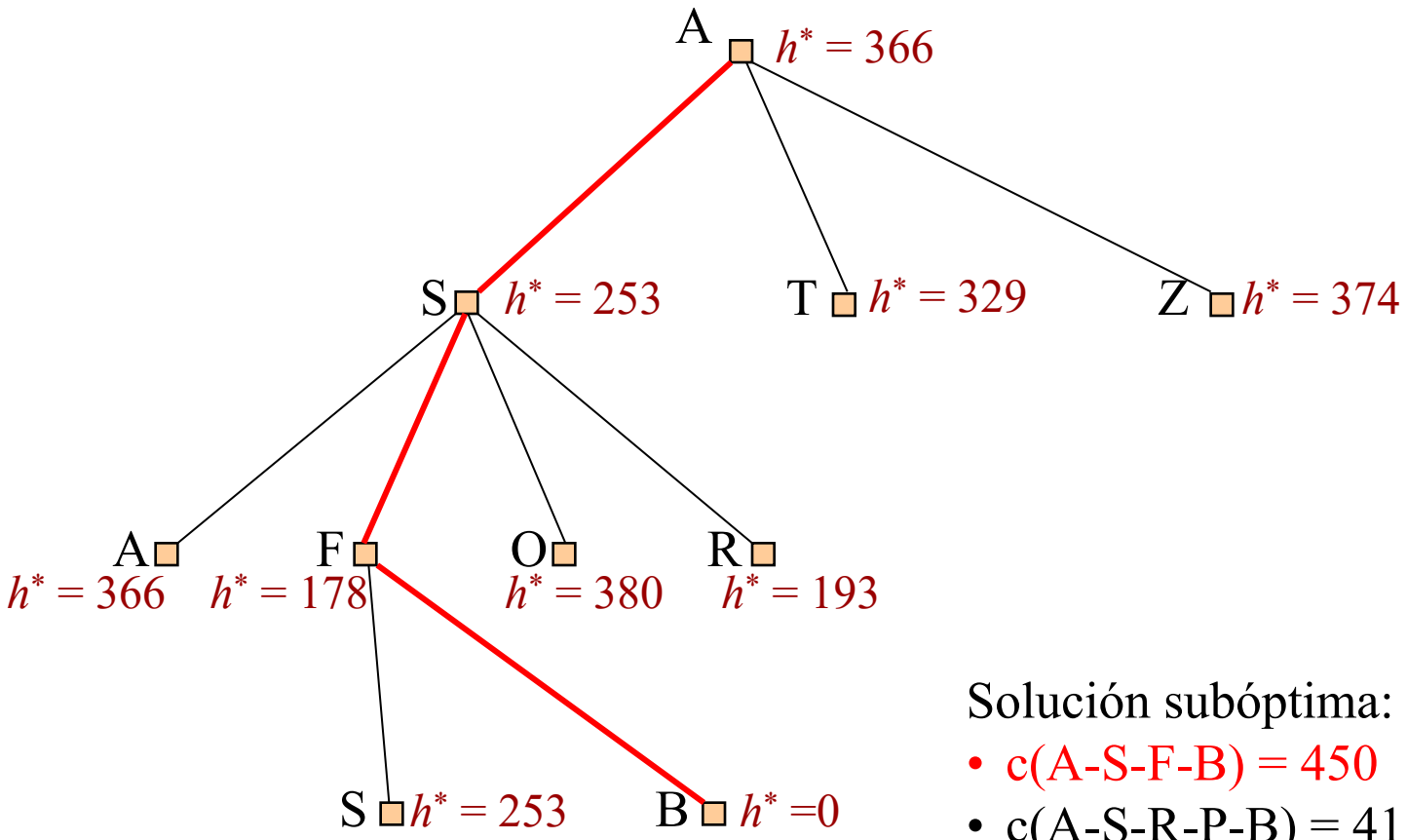
Búsqueda voraz

Búsqueda avara:

- Inglés: greedy search
- Idea:
  - 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}

Ejemplo 1: búsqueda voraz



	h^*
A	366
B	0
C	160
D	242
E	161
F	178
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	98
R	193
S	253
T	329
U	80
V	199
Z	374

Solución subóptima:

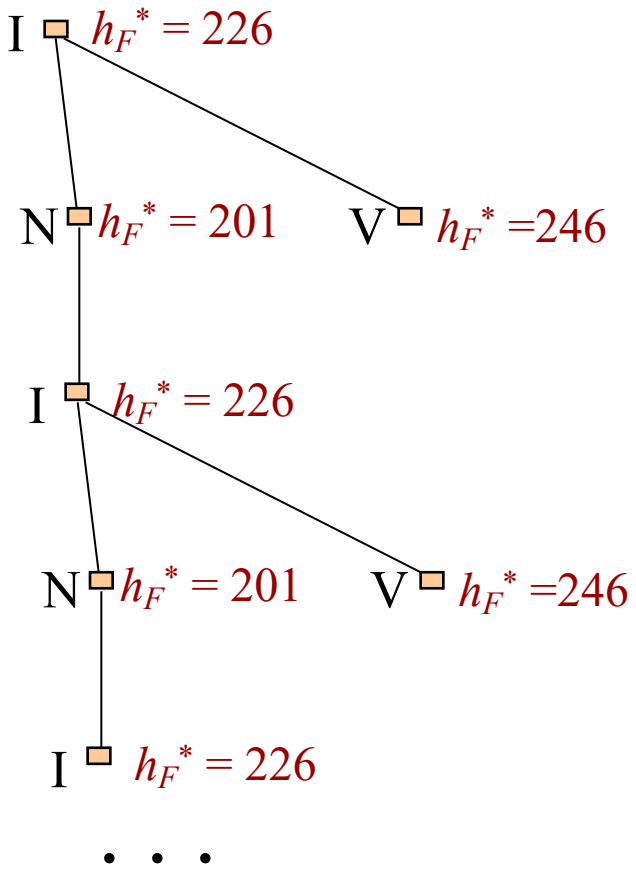
- c(A-S-F-B) = 450
- c(A-S-R-P-B) = 418

Ejemplo 2: búsqueda voraz

Ejemplo ("*mínimo local* de h_F^* "):

- Nodo inicial: *I* (Iasi)
- Nodo meta: *F* (Fagaras)
- h_F^* estima la distancia hasta *F*

	h_F^*
F	0
I	226
N	201
V	246
. . .	



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*

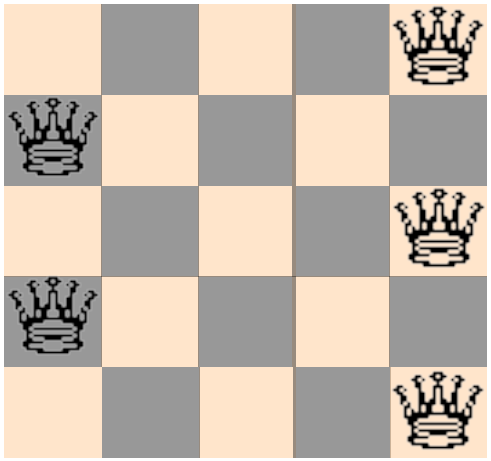
Comentarios:

- problema fundamental de la búsqueda voraz:
  - sólo se fija en la distancia restante desde el nodo actual,
  - no considera el coste para llegar hasta él
- para asegurar la *completitud* habría que evitar todos los estados repetidos
- el método es *óptimo* sólo 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

Ejercicio

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)

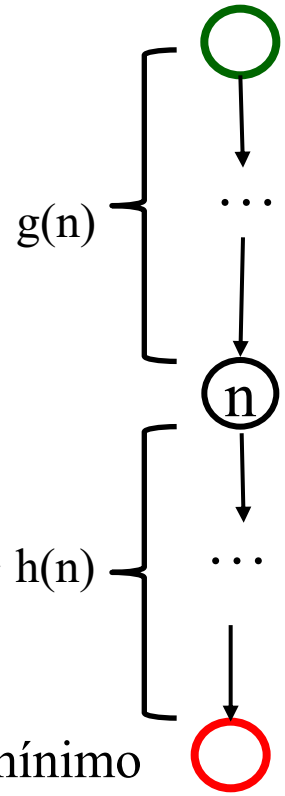
a) encuentre una heurística h^* para el problema de las 5 reinas

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

Búsqueda A*

Idea:

- 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

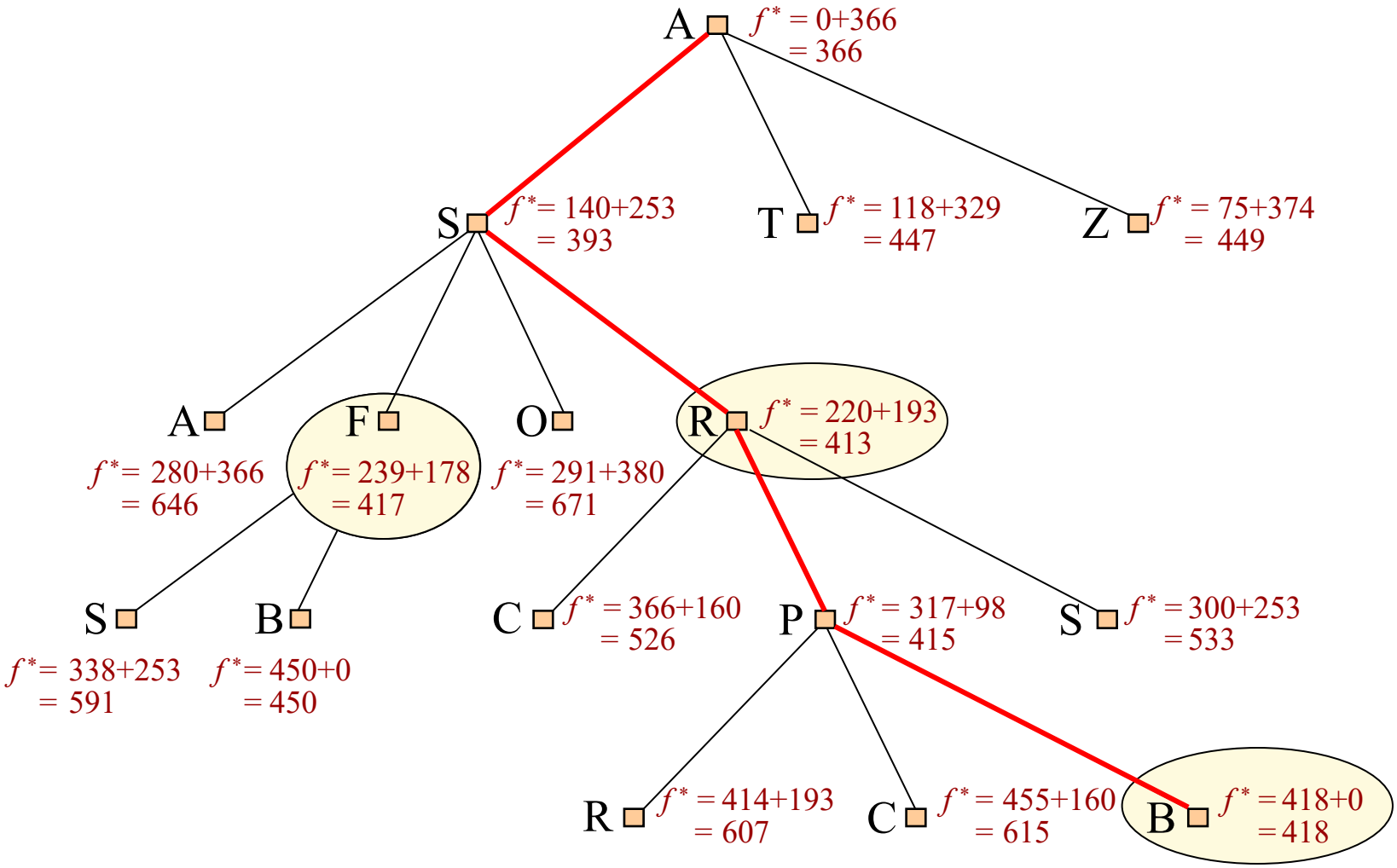
El Algoritmo A*

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^*

{A*}
abierta ← s_0
Repetir
  Si *vacío?*(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}

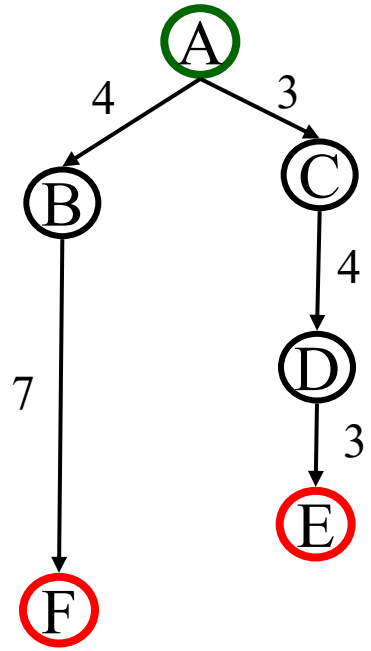
Ejemplo: Búsqueda A*



	h^*
A	366
B	0
C	160
D	242
E	161
F	178
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	98
R	193
S	253
T	329
U	80
V	199
Z	374

Ejemplo

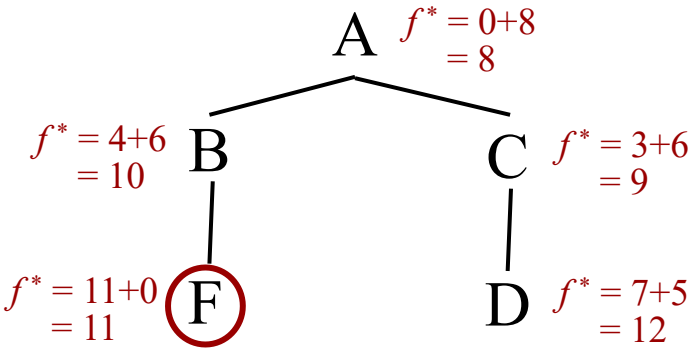
Espacio de estados:



	h^*
A	8
B	6
C	6
D	5
E	0
F	0

- Estado inicial: A
- Estados meta: E, F
- $g(E)=10 < 11=g(F)$

Árbol de búsqueda:

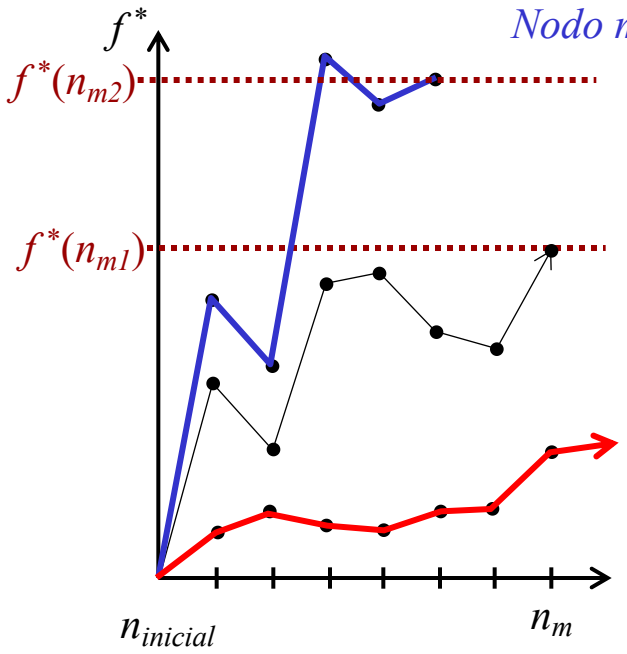


- A^* no encuentra el mejor nodo meta E, sino F
- h^* no es optimista: $h^*(D) = 5 > 3 = h(D)$
- En el camino de A al mejor nodo meta E ($f^*(E) = 10$) hay un "pico" D ($f^*(D) = 12$) que la búsqueda A^* no ha podido superar

Optimalidad y completitud de A*

A* es óptimo y completo si:

- h^* es optimista, y 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*

Resumen A*

Algoritmo A* :

- $f^*(n) = g(n)+h^*(n)$: estimación del coste real del plan (camino) de mínimo coste que lleva del *estado inicial* a un nodo meta pasando por n
- entre las hojas del árbol de búsqueda, elegir el nodo de valor f^* mínimo
- el Algoritmo A* es *completo*
- el Algoritmo A* es *óptimo* para funciones heurísticas optimistas

Cuestiones pendientes:

- ¿ Cómo se pueden encontrar funciones heurísticas *optimistas*?
- ¿ Cómo se puede distinguir entre "*buenas*" y "*malas*" funciones heurísticas?
- ¿Cuál el la *complejidad* del algoritmo A*? ¿Cómo se puede optimizar?

Encontrar Funciones Heurísticas: Aprendizaje

Idea: generar información heurística "sobre la marcha"

- realizar varias búsquedas (ligeramente diferentes) en el mismo dominio (p.e. siempre a Bucarest, pero desde diferentes ciudades iniciales)
- En cada paso de una búsqueda, usar el coste real de un paso parar mejorar el valor de h^*
- En la próxima búsqueda se utilizan los valores de h^* actualizados

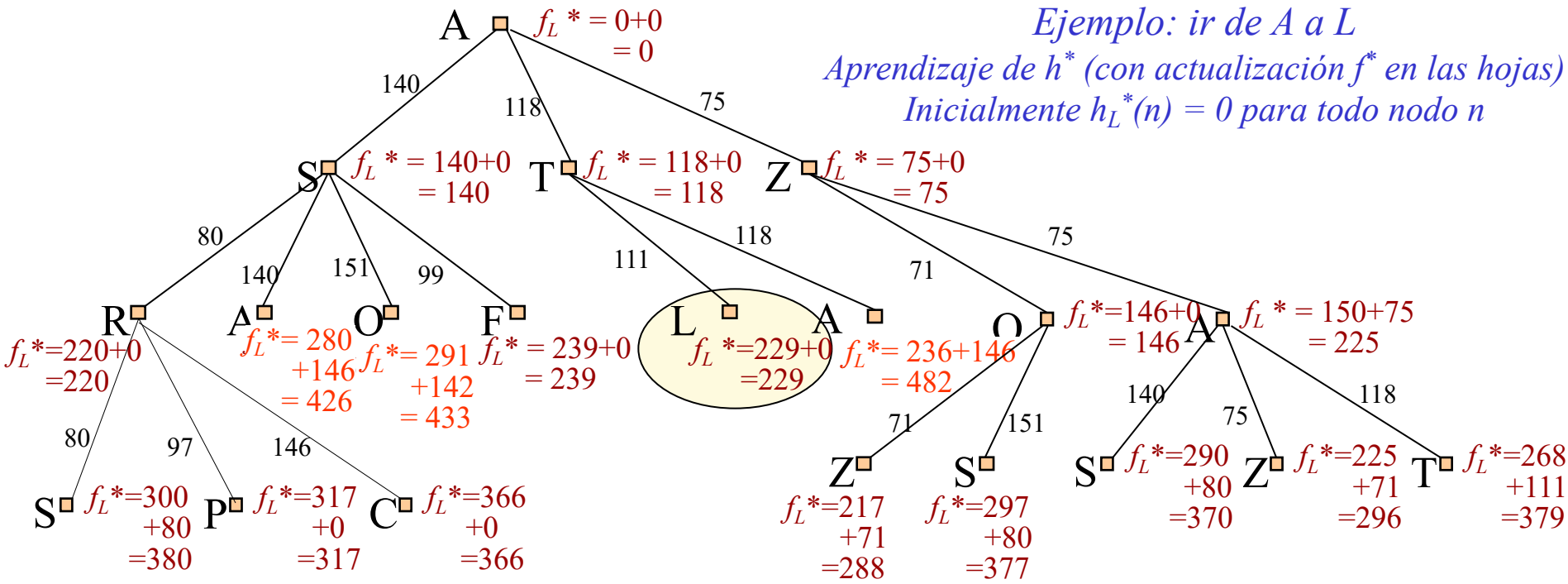
Método:

- Inicialmente, se realiza una búsqueda con $h^*(n) = 0$ para todos los nodos n
- En cada paso de n_i a n_j: $h^*(n_i) \leftarrow \min_{n_j \in expandir(n_i)} \left[h^*(n_j) + c(n_i, n_j) \right]$
- Extensión (opcional):
  - ✓ Al aprender un nuevo valor $h^*(n)$ para un estado n, en caso de hubiera otras copias de n como hojas en el árbol de búsqueda generado por A^*, actualizar sus valores de f^*

Problema:

- Hay que almacenar los valores h^* de *todos* los nodos en una tabla (memoria!)

Ejemplo: A* con Aprendizaje de una Función Heurística



Ejemplo: ir de A a L

Aprendizaje de h^* (con actualización f^* en las hojas)

Inicialmente $h_L^*(n) = 0$ para todo nodo n

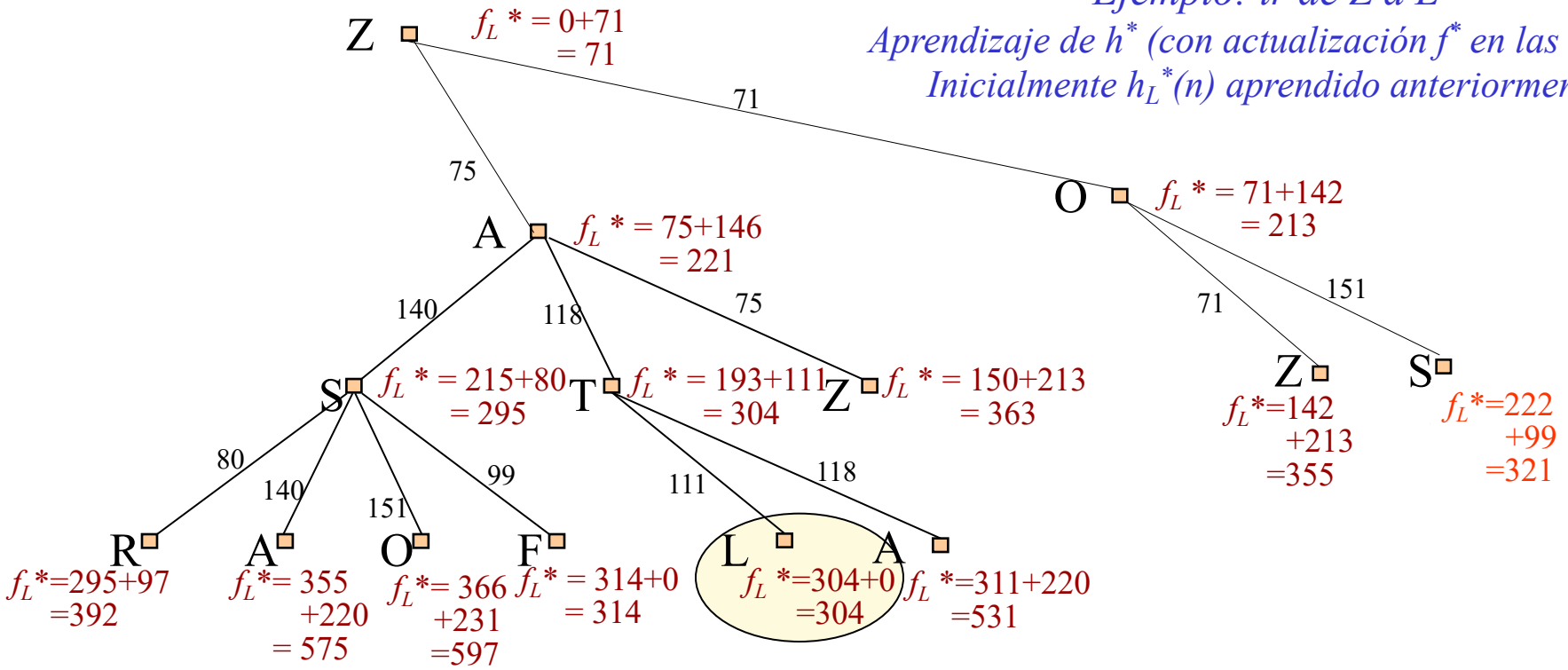
n	A	B	C	D	E	F	G	H	I	L	M	N	O	P	R	S	T	U	V	Z
h_L^*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	~~75~~ 146												142		97	80	111			71

Ejemplo: A* con Aprendizaje de una Función Heurística

Ejemplo: ir de Z a L

Aprendizaje de h^* (con actualización f^* en las hojas)

Inicialmente $h_L^*(n)$ aprendido anteriormente



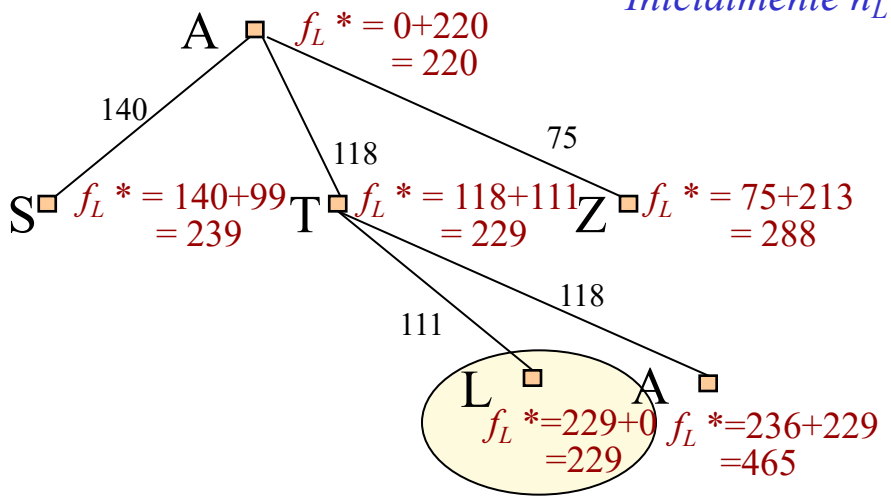
n	A	B	C	D	E	F	G	H	I	L	M	N	O	P	R	S	T	U	V	Z
h_L^*	146	0	0	0	0	0	0	0	0	0	0	0	142	0	97	80	111	0	0	71
	220												***231***			***99***	***111***			***213***

Ejemplo: A* con Aprendizaje de una Función Heurística

Ejemplo: ir de A a L

Aprendizaje de h^* (con actualización f^* en las hojas)

Inicialmente $h_L^*(n)$ aprendido anteriormente



n	A	B	C	D	E	F	G	H	I	L	M	N	O	P	R	S	T	U	V	Z
h_L^*	220	0	0	0	0	0	0	0	0	0	0	0	231	0	97	99	111	0	0	213

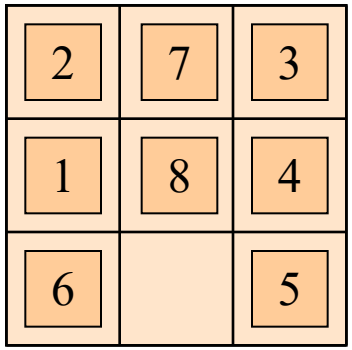
229 ***111***

Encontrar de Funciones Heurísticas: Diseño

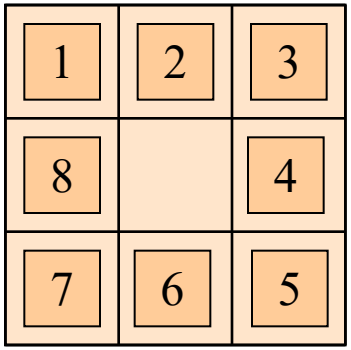
El problema del 8-puzzle:

- **Estados**:
  - posición de cada una de las piezas
- **Operadores:**
  - mover pieza adyacente a la posición del "hueco"
  - de 2 a 4 operadores aplicables, según el estado
- **Coste:**
  - La aplicación de cada operador vale *una* unidad

Estado inicial

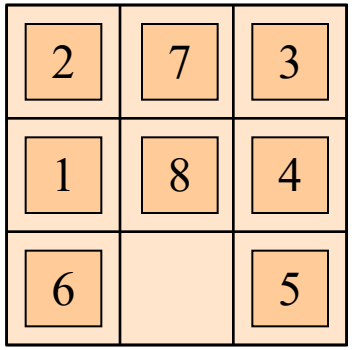


Estado meta

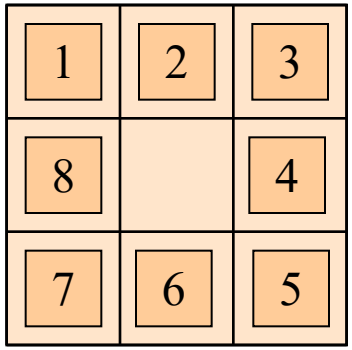


Encontrar de Funciones Heurísticas: Diseño

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 vacío
  c) si A es adyacente a B
- **Funciones heurísticas**:
  a) número de *piezas descolocadas*
     - $h_a^*(s_0) = 5$
  b) suma de *saltos* necesarios
     - $h_b^*(s_0) = 5$
  c) suma de las *distancias de Manhattan*
     - $h_c^*(s_0) = 1+1+1+3+1=7$

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^*

Teorema:

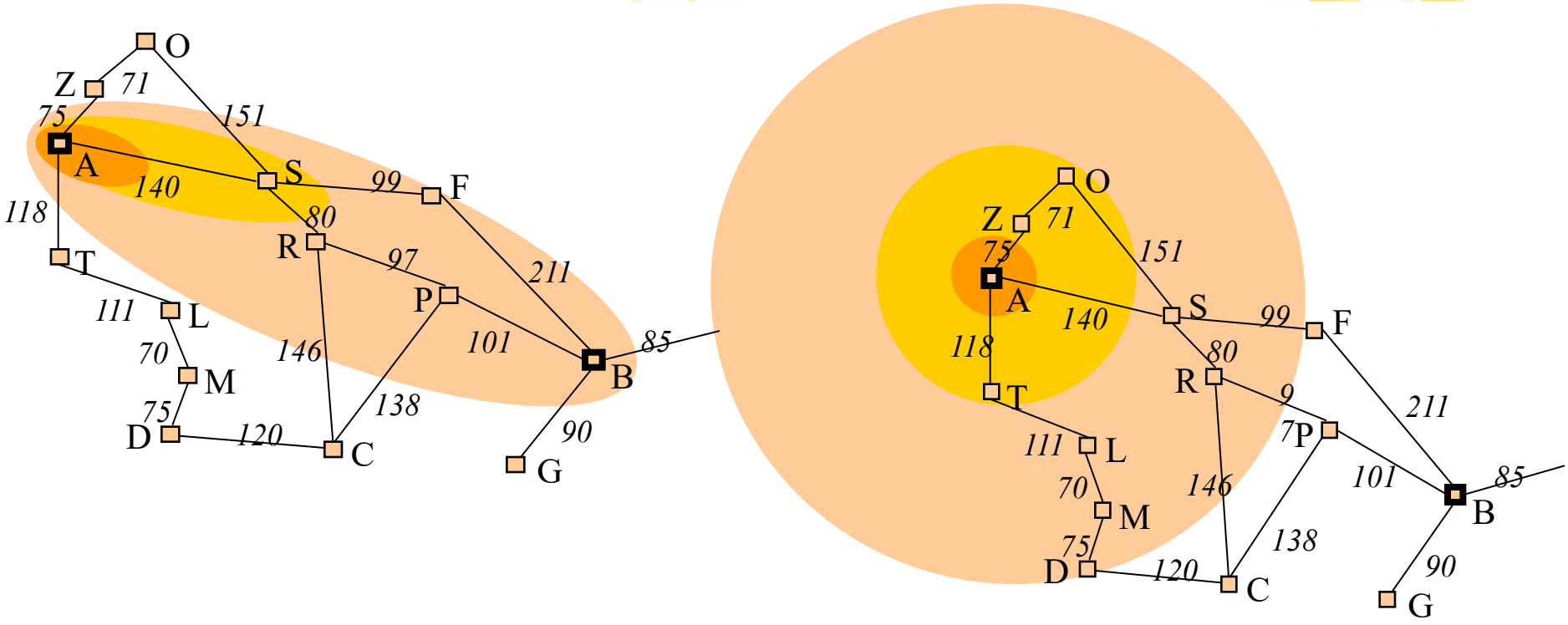
Sean h_1^* y h_2^* dos funciones heurísticas optimistas. Si h_1^* es más informada que h_2^*, entonces $A^*(h_1^*)$ expande igual o menos nodos que $A^*(h_2^*)$.

Conclusión:

- preferir grandes valores de h^*, siempre que se mantenga optimista
- si hay varias funciones heurísticas optimistas:

$$h^*(n) = \max\left(h_1^*(n), h_2^*(n), \ldots, h_m^*(n)\right)$$

Calidad de las Funciones Heurísticas



Función heurística más informada:
$h^*(n)$= distancia en línea recta desde la ciudad n hasta B

Función heurística poco informada:
$h^*(n)=0$ para todos los nodos n

Complejidad de A*

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

- 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 de coste uniforme

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	B. no informada (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

Mejoras de A*

Mejoras usando **heurísticas fuertes**:

- idea: *acotar* el espacio de búsqueda con información heurística
- búsqueda guiada por subobjetivos (*island-driven search*):
  - elegir estados (subobjetivos, "islas") $i_1,\ldots,i_n$ por los que se supone que una solución ha de pasar (p.e. puertos)
  - realizar búsquedas A* del estado inicial n_0 a i_1, de i_1 a $i_2,\ldots$, y de i_n a la meta n_m
- búsqueda jerárquica:
  - *macro-operadores* representan acciones complejas y ficticias (p.e. "ir de puerto a puerto")
  - realizar búsqueda A* a *meta-nivel* con macro-operadores: los estados de la solución son los subobjetivos
  - *refinar* la solución anterior en base a la búsqueda guiada por subobjetivos

Búsqueda en línea

Búsqueda en línea (*online search*):

- denomina una *clase* de métodos que "engranan" búsqueda (elección de acciones) y acción/percepción
- indicados cuando:
  - el espacio de búsqueda es demasiado grande para buscar hasta la solución (y no se pueden aplicar las técnicas anteriores)
  - no es realista usar un modelo determinista de los efectos de acciones
    - por frecuentes contingencias (p.e.: a veces el brazo deja caer un bloque)
    - porque no se dispone de un modelo del entorno (p.e.: un "mapa")
- medida de eficiencia:
  - en general, no se puede asegurar optimalidad ni completitud
  - minimizar el $\textit{índice competitivo} = \frac{\textit{Coste del camino real del agente}}{\textit{Coste del camino óptimo}}$
  - el índice competitivo puede ser infinito, particularmente cuando hay acciones que no son reversibles

Búsqueda con horizonte

Búsqueda con horizonte (*limited-horizon search*):

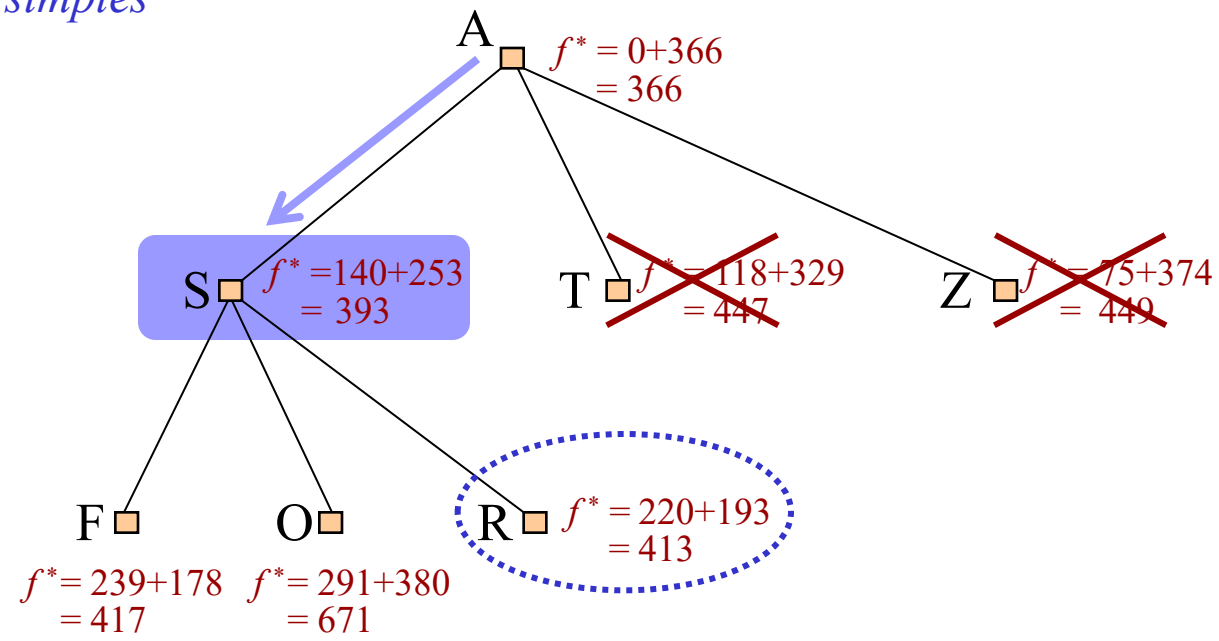
- Idea subyacente:
  - Sólo buscar k acciones hacia adelante, luego decidir la siguiente acción y ejecutarla
  - Sea H_k el conjunto de nodos a los que se puede llegar con k acciones. Para todos los nodos $n \in H_k$, $f^*(n)$ representa el coste total estimado para llegar desde el estado inicial a la meta pasando por n
  - *Heurística (fuerte)* que aplica la búsqueda con horizonte: el mejor camino a la meta es a través del nodo n^* a nivel k de menor valor de f^* (i.e. $n^* \leftarrow \arg\min_{n \in H_k} [f^*(n)]$)
  - Por tanto, se realiza el primer paso en este camino, y se repite el proceso

Búsqueda con horizonte

- Algoritmo:
  - Percibir el estado actual s
  - Aplicar *búsqueda en profundidad limitada* por el nivel k
  - En cada nodo hoja n de nivel k, actualizar un valor α que representa el mínimo valor de f^* encontrado en nodos de nivel k hasta el momento, e.d.:
    - Inicializar $\alpha \leftarrow +\infty$
    - En el nodo n (de nivel k) realizar: $\alpha \leftarrow \min(\alpha, f^*(n))$
  - Si en la búsqueda en profundidad limitada se ha encontrado un nodo meta, sea n^* este nodo. De lo contrario, sea n^* el primer nodo hoja (de nivel k) con $f^*(n^*) = \alpha$
  - Ejecutar la primera acción a^* en el camino que lleva a n^*
  - Repetir hasta que el agente se encuentre en un estado meta

Ejemplo: Búsqueda con horizonte

estado actual s = A
horizonte k = 2
filtrando ciclos simples

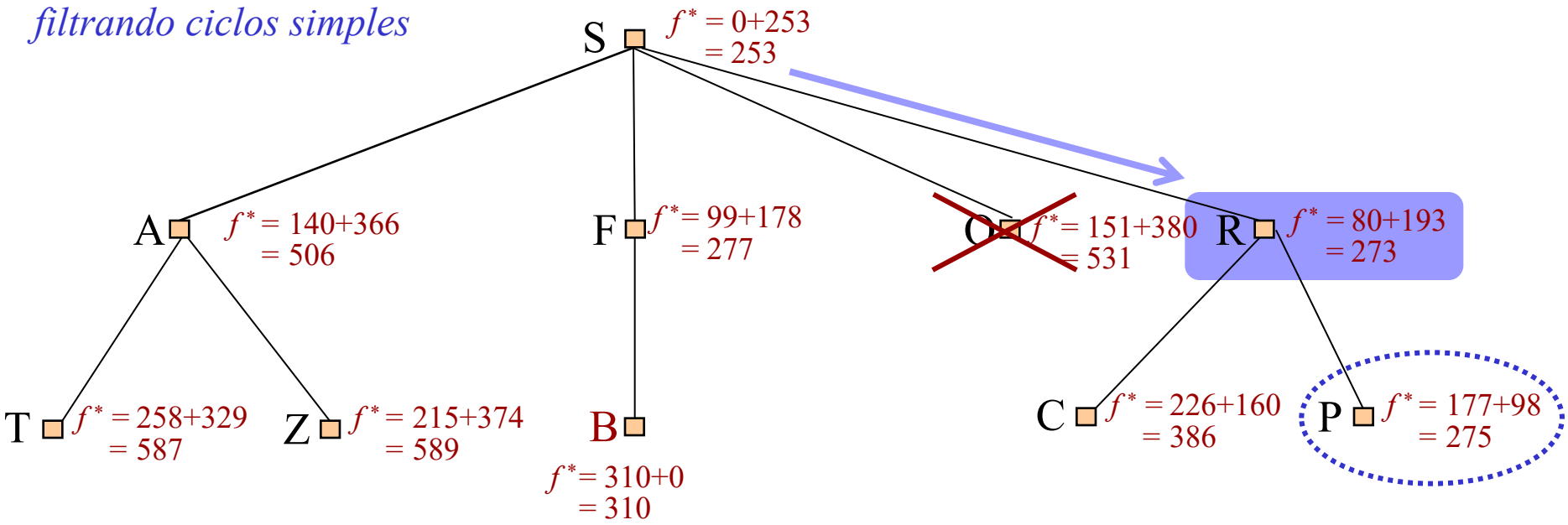


k=2

$\alpha = 417$ $\alpha = 417$ $\alpha = 413$

Ejemplo: Búsqueda con horizonte

estado actual s = S
horizonte k = 2
filtrando ciclos simples

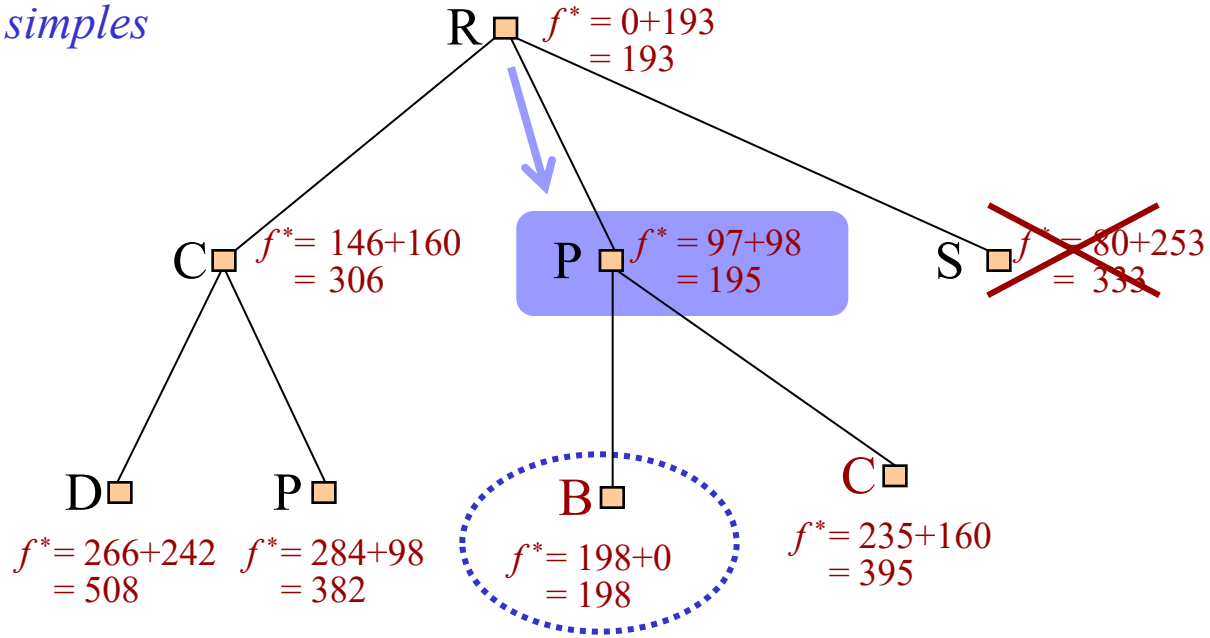


k=2

$\alpha = 587$ $\alpha = 587$ $\alpha = 310$ $\alpha = 310$ $\alpha = 275$

Ejemplo: Búsqueda con horizonte

estado actual s = R
horizonte k = 2
filtrando ciclos simples



k=2

$\alpha = 508$ $\alpha = 382$ $\alpha = 198$ $\alpha = 198$