

Procura Informada

Capítulo 3

Livro

- Capítulo 3, Secções 3.5 – 3.6

Resumo

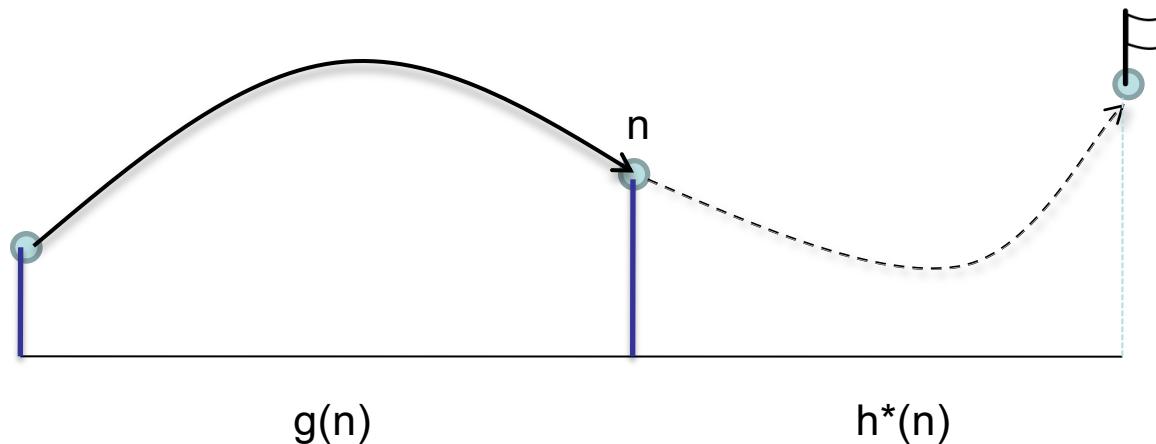
- Estratégias de procura informadas
 - Gananciosa
 - A*
 - IDA*
 - Melhor Primeiro Recursiva (RBFS)
 - E outras...
- Heurísticas

Árvore de Procura

- Uma estratégia de procura determina a **ordem de expansão dos nós**
- As procura informadas usam **conhecimento específico do problema** para determinar a ordem de expansão dos nós
- Tipicamente este conhecimento é incorporado sob a forma de **heurísticas** (estimativas)

Função Heurística

- $g(n)$ – custo do caminho do estado inicial até o nó n
- $h^*(n)$ – custo do **melhor caminho** a partir do nó n até um objetivo
- $h(n)$ – **estimativa** do custo do melhor caminho a partir do nó n até um objectivo
- $h(n) = 0$, se n = estado objectivo



Procura Melhor Primeiro

- Ideia: usar uma **função de avaliação** $f(n)$ para cada nó
 - $f(n)$ pode usar conhecimento específico do problema
 - O “melhor” nó é o que tem o menor valor de $f(n)$
→ Expandir primeiro o nó folha que tem o menor valor de $f(n)$
- Implementação:
Nós na fronteira ordenados por ordem crescente da função de avaliação
 - Fronteira = $\{n_1, n_2, n_3, \dots\} \rightarrow f(n_1) \leq f(n_2) \leq f(n_3) \leq \dots$
- Casos especiais:
 - Procura Gananciosa
 - Procura A*

Procura Melhor Primeiro

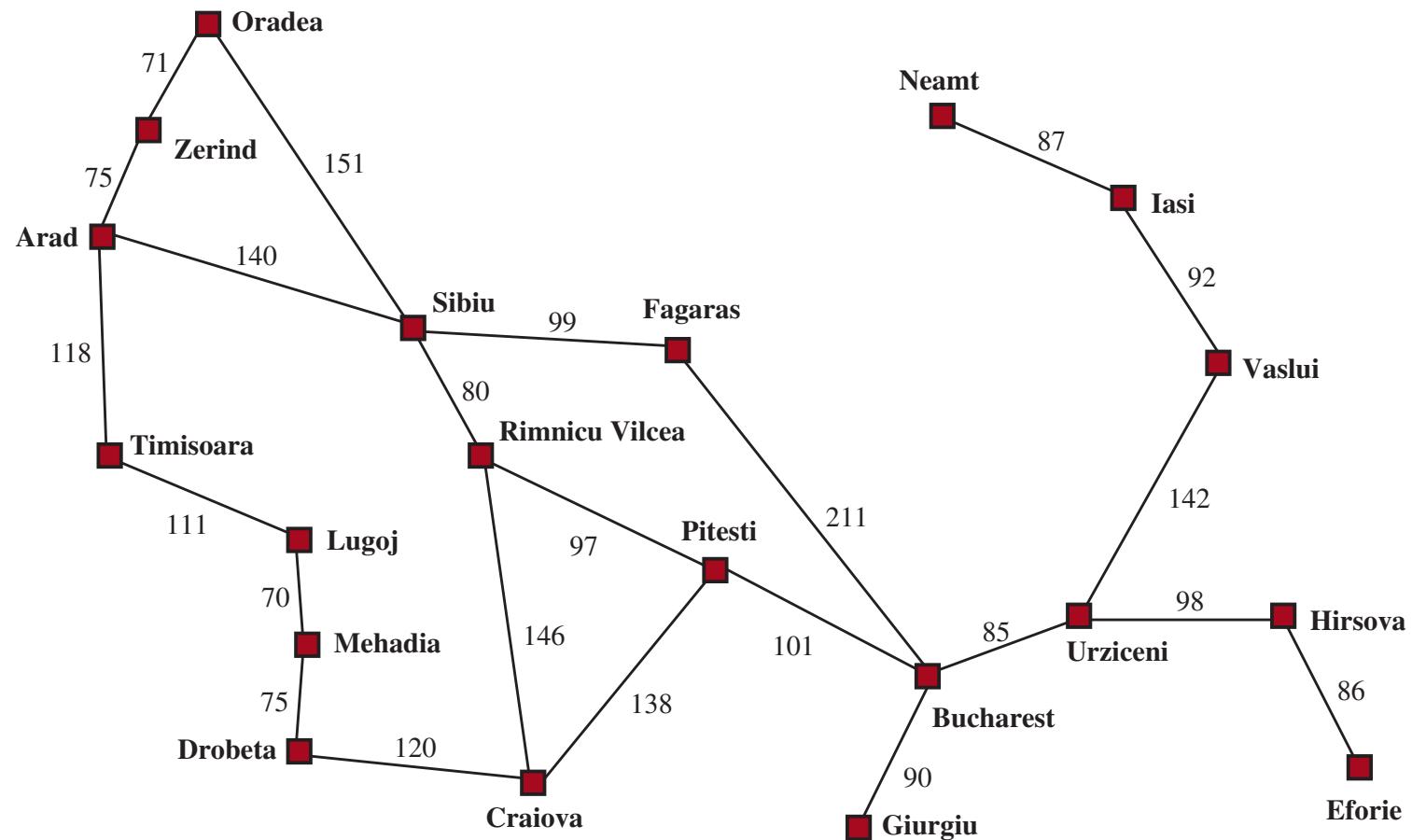
```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
    node  $\leftarrow$  NODE(STATE=problem.INITIAL)
    frontier  $\leftarrow$  a priority queue ordered by f, with node as an element
    reached  $\leftarrow$  a lookup table, with one entry with key problem.INITIAL and value node
    while not IS-EMPTY(frontier) do
        node  $\leftarrow$  POP(frontier)
        if problem.IS-GOAL(node.STATE) then return node
        for each child in EXPAND(problem, node) do
            s  $\leftarrow$  child.STATE
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
                reached[s]  $\leftarrow$  child
                add child to frontier
    return failure
```

```
function EXPAND(problem, node) yields nodes
    s  $\leftarrow$  node.STATE
    for each action in problem.ACTIONS(s) do
        s'  $\leftarrow$  problem.RESULT(s, action)
        cost  $\leftarrow$  node.PATH-COST + problem.ACTION-COST(s, action, s')
        yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Procura gananciosa

- Função de avaliação $f(n) = h(n)$ (**heurística**)
- = estimativa do custo do caminho desde n até ao objetivo
- Exemplo $h_{dlr}(n)$ = distância em linha reta desde n até Bucareste
- Procura gananciosa expande o nó que “**parece**” estar mais próximo do objetivo

Exemplo: Roménia (distância real)



$f(n) = \text{distância linha reta (km) até Bucareste}$

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

Procura Gananciosa: exemplo



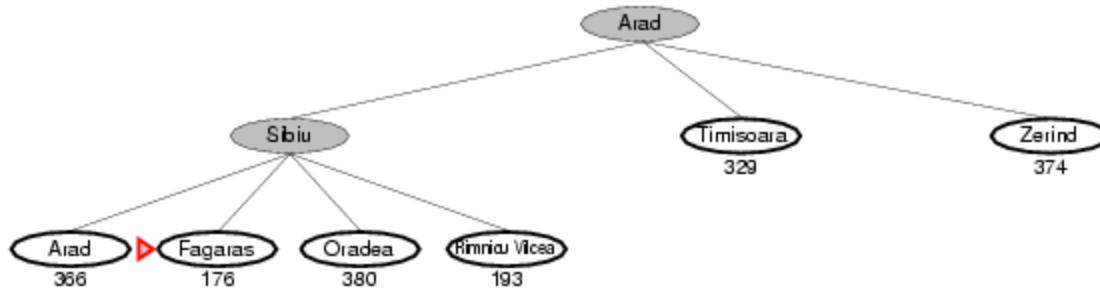
Fronteira = {Arad(366)}

Procura Gananciosa: exemplo



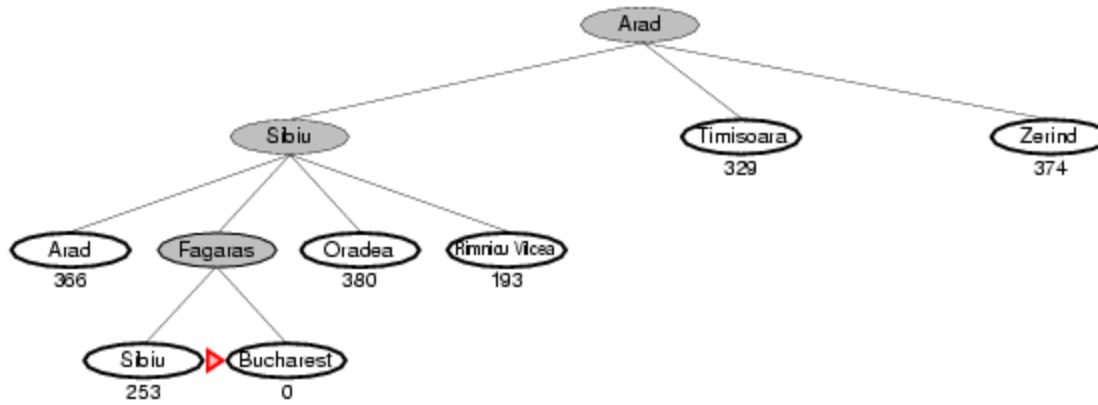
Fronteira = {Sibiu(253), Timisoara(329), Zerind(374)}

Procura Gananciosa: exemplo



Fronteira = {Fagaras(176), Rimnicu Vilcea(193),
Timisoara(329), Arad(366), Zerind(374), Oradea(380)}

Procura Gananciosa: exemplo



Fronteira = {Bucharest(0), Rimnicu Vilcea(193), Sibiu(253),
Timisoara(329), Arad(366), Zerind(374), Oradea(380)}

Procura gananciosa: propriedades

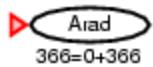
- Completa? Não – pode entrar em ciclo
 - exº, Iasi → Fagaras = Iasi, Neamt, Iasi, ...
- Tempo? $O(b^m)$ mas uma boa heurística pode reduzi-lo dramaticamente
- Espaço? $O(b^m)$ no pior caso mantém todos os nós em memória
- Ótima? Não

- Semelhante à procura em profundidade, mas mais exigente em memória (como a procura em largura)

Procura A*

- Ideia: **evitar expandir caminhos que já têm um custo muito elevado**
- Função de avaliação $f(n) = g(n) + h(n)$
- $g(n)$ = custo desde o nó inicial até n
- $h(n)$ = estimativa do custo deste n até um estado objetivo
- $f(n)$ = estimativa do custo total da solução = caminho desde estado inicial até estado objetivo (passando por n)

Procura A*: exemplo



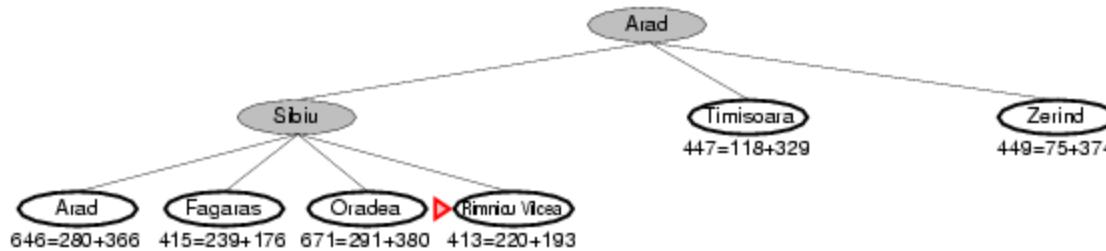
Fronteira = {Arad(366)}

Procura A*: exemplo



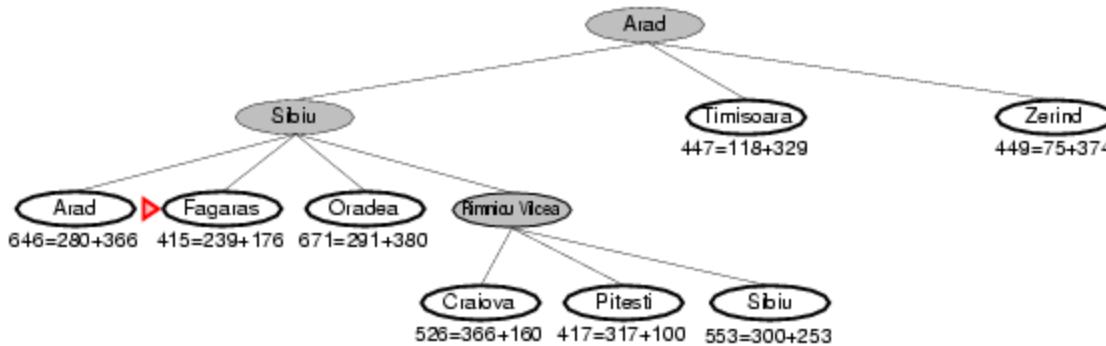
Fronteira = {Sibiu(393), Timisoara(447), Zerind(449)}

Procura A*: exemplo



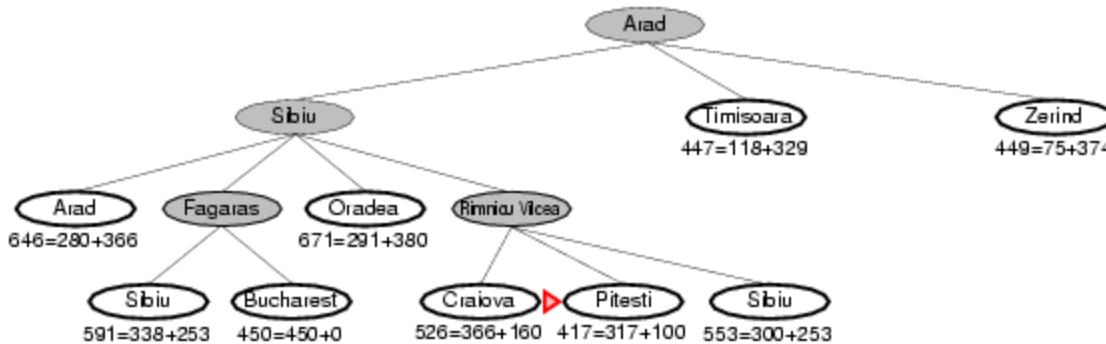
Fronteira = {Rimnicu Vilcea(413), Fagaras(415),
Timisoara(447), Zerind(449), Arad(646), Oradea(671)}

Procura A*: exemplo



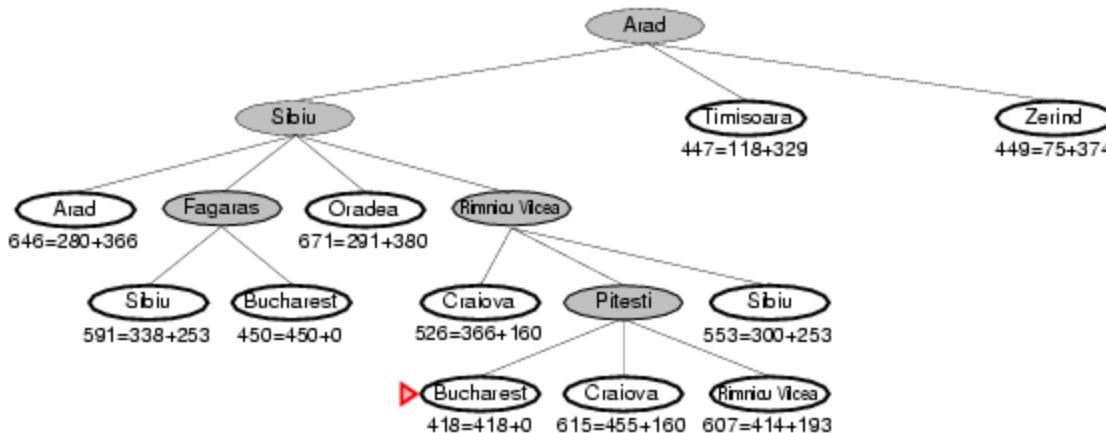
Fronteira = {Fagaras(415), Pitesti(417), Timisoara(447),
Zerind(449), Craiova(526), Sibiu(553), Arad(646), Oradea(671)}

Procura A*: exemplo



Fronteira = {Pitesti(417), Bucharest(450), Timisoara(447),
Zerind(449), Craiova(526), Sibiu(553), Sibiu(591), Arad(646),
Oradea(671)}

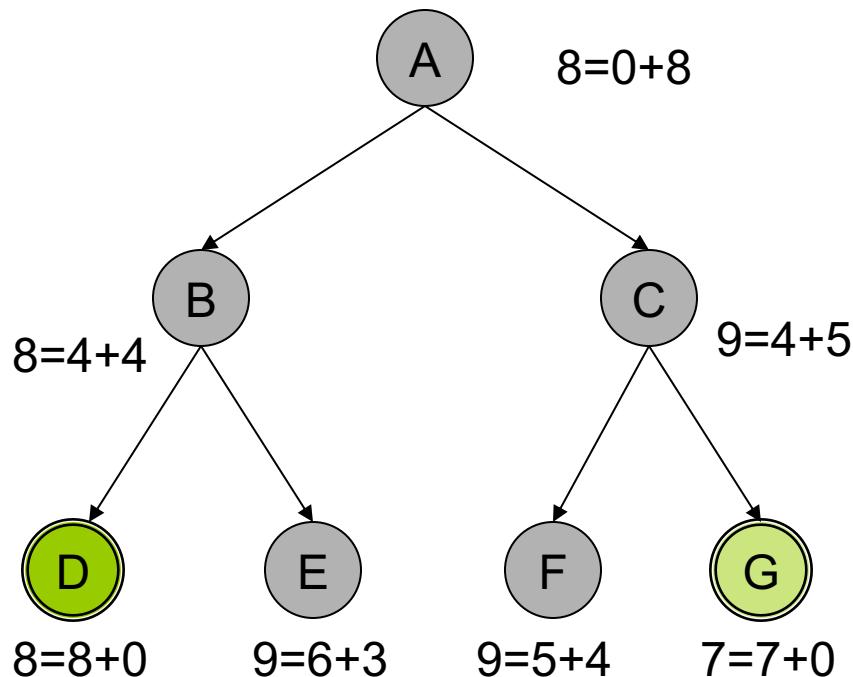
Procura A*: exemplo



Fronteira = {Bucharest(418), Bucharest(450), Timisoara(447), Zerind(449), Craiova(526), Sibiu(553), Sibiu(591), Rimnicu Vilcea(607), Craiova(615), Arad(646), Oradea(671)}

A* é ótima?

- Não! Aqui fica um exemplo...



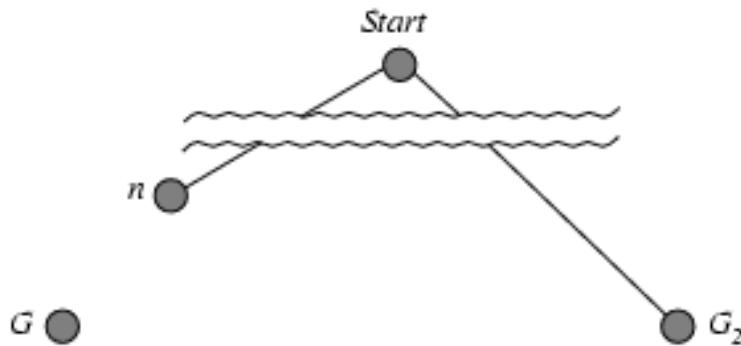
- G é objetivo ótimo mas D é o objetivo encontrado...

Heurísticas admissíveis

- Uma heurística $h(n)$ é **admissível** se para cada nó n se verifica:
 - $h(n) \leq h^*(n)$
 - onde $h^*(n)$ é o custo **real** do melhor caminho desde n até ao objetivo.
- Uma heurística admissível **nunca sobrestima** o custo de atingir o objetivo, i.e. é **otimista**
- Exemplo: $h_{dlr}(n)$ - distância em linha reta nunca sobrestima a distância real em estrada
- **Teorema:** se $h(n)$ é admissível, então a *procura em árvore A** é ótima

A^* é ótima (prova)

- Consideremos um nó objetivo não ótimo G_2 que já foi gerado mas não expandido (nó folha). Seja n um nó folha tal que n está no menor caminho para um nó objetivo ótimo G (com custo C^*).



- $h(G_2) = 0$ porque G_2 é objetivo
- $f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$ porque G_2 não é ótimo
- $f(n) = g(n) + h(n) \leq C^*$ porque h é admissível
- Logo $f(n) \leq C^* < f(G_2)$ e G_2 não será considerado antes de n e G

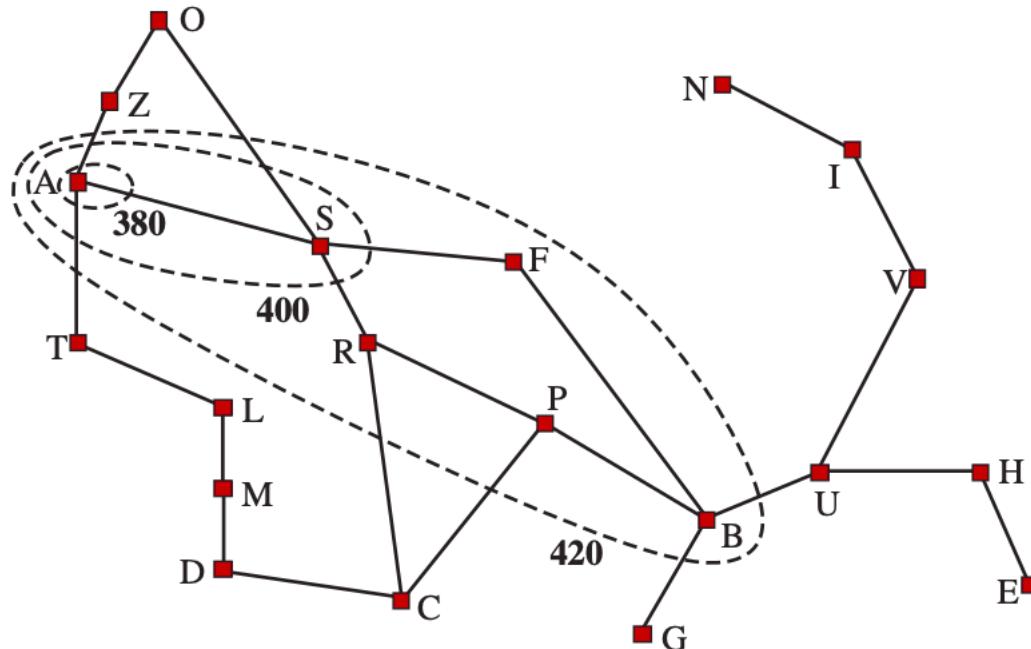
A* é ótima

- Expande todos os nós com
$$f(n) < C^*$$
- Pode depois expandir alguns nós sobre a “curva de nível objetivo” com
$$f(n) = C^*$$

antes de selecionar o estado objetivo

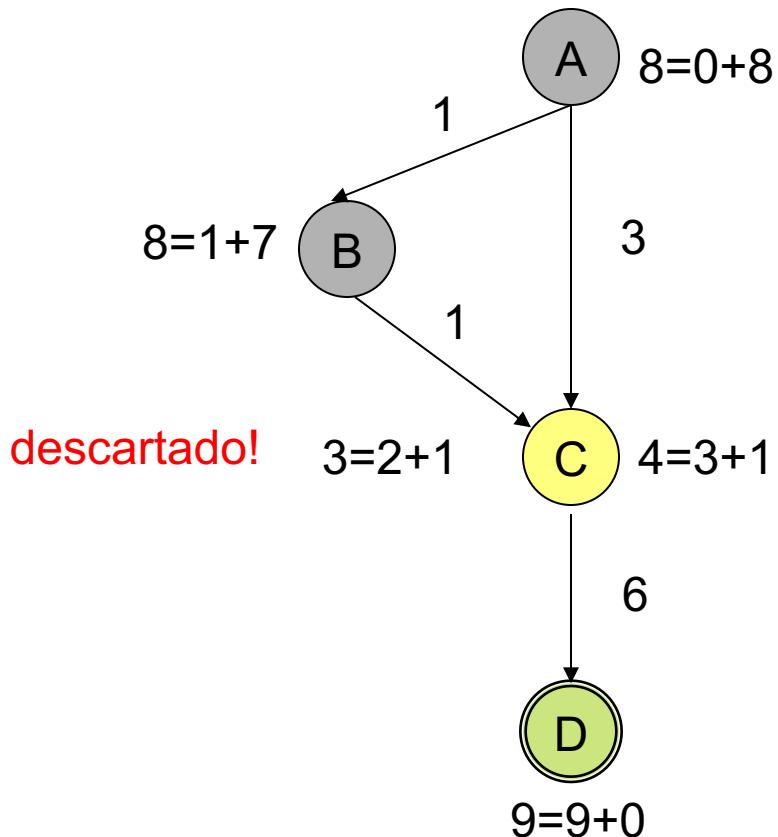
A^* é ótima

- A^* expande os nós por ordem crescente do valor de f
- Gradualmente adiciona contornos/“curvas de nível” (à semelhança dos mapas topográficos) que identificam conjuntos de nós
- Contorno i tem todos os nós com $f \leq f_i$, com $f_i < f_{i+1}$



A* em grafo é ótima?

- Não, mesmo que a heurística seja admissível!
 - Vejamos um exemplo...



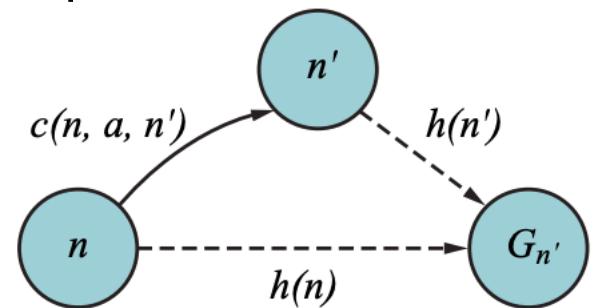
C(3) é descartado
por causa de C(4) e
o objetivo
encontrado (D(9))
não é ótimo...

A* e Procura em Grafo

- A* com procura em grafo não é ótima com heurísticas admissíveis
 - Pode ser descartado um nó que está no caminho que leva à solução ótima pelo facto de o mesmo nó já ter sido explorado no passado
- Pode passar a ser ótima se for mantido o registo dos caminhos e do valor de $f(n)$ associados a todos os nós já explorados
 - Um nó/caminho só é descartado se o valor de $f(n)$ for maior do que o valor registado
- Caso contrário... a heurística tem de ser **consistente!**
 - **Teorema:** se $h(n)$ é **consistente**, então A* usando procura em grafo é ótima

Heurísticas Consistentes

- Uma heurística é **consistente** se para cada nó n , e para cada sucessor n' de n gerado por uma ação a temos,
$$h(n) \leq c(n, a, n') + h(n')$$
 desigualdade triangular
um lado de um triângulo não pode ser maior que
a soma dos outros dois lados



$c(n, a, n')$ é o custo associado ao caminho de n a n' através de a

- Definição formal, mas *não muito intuitiva* de consistência

Heurísticas Consistentes

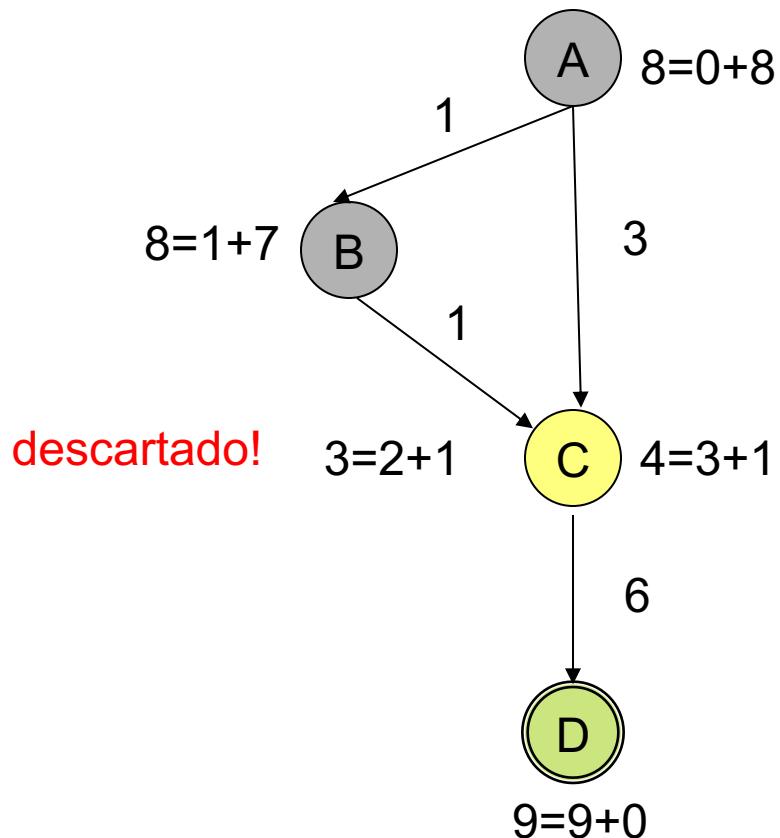
- Garantem que se existirem dois caminhos para chegar ao mesmo objetivo ótimo então o caminho de menor custo é sempre seguido em primeiro lugar
- Se h é consistente, então para n' sucessor de n
$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + \underline{c(n,a,n')} + h(n') \end{aligned}$$


pela definição de consistência $\geq h(n)$

$$\begin{aligned} f(n') &\geq g(n) + h(n) \\ f(n') &\geq f(n) \end{aligned}$$
- Logo, o valor de $f(n)$ nunca decresce/diminui ao longo de um caminho

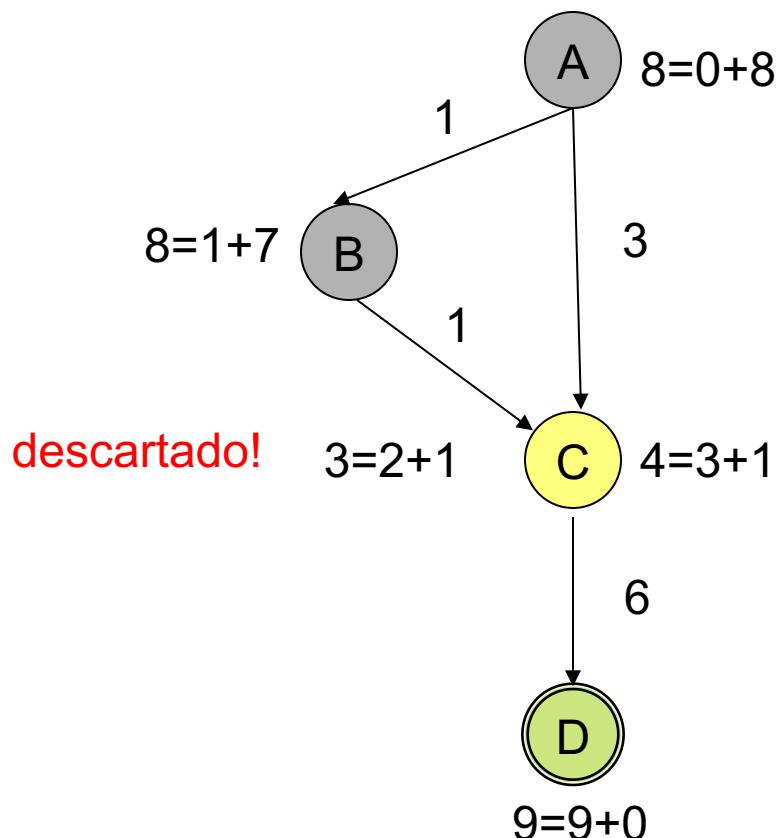
Heurísticas Consistentes

- A heurística usada é consistente ou não?



Heurísticas Consistentes

- A heurística usada é consistente ou não?



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

$$h(A) \leq c(A,B) + h(B) ?$$

$8 \leq 1 + 7$ ✓

$$h(B) \leq c(B,C) + h(C) ?$$

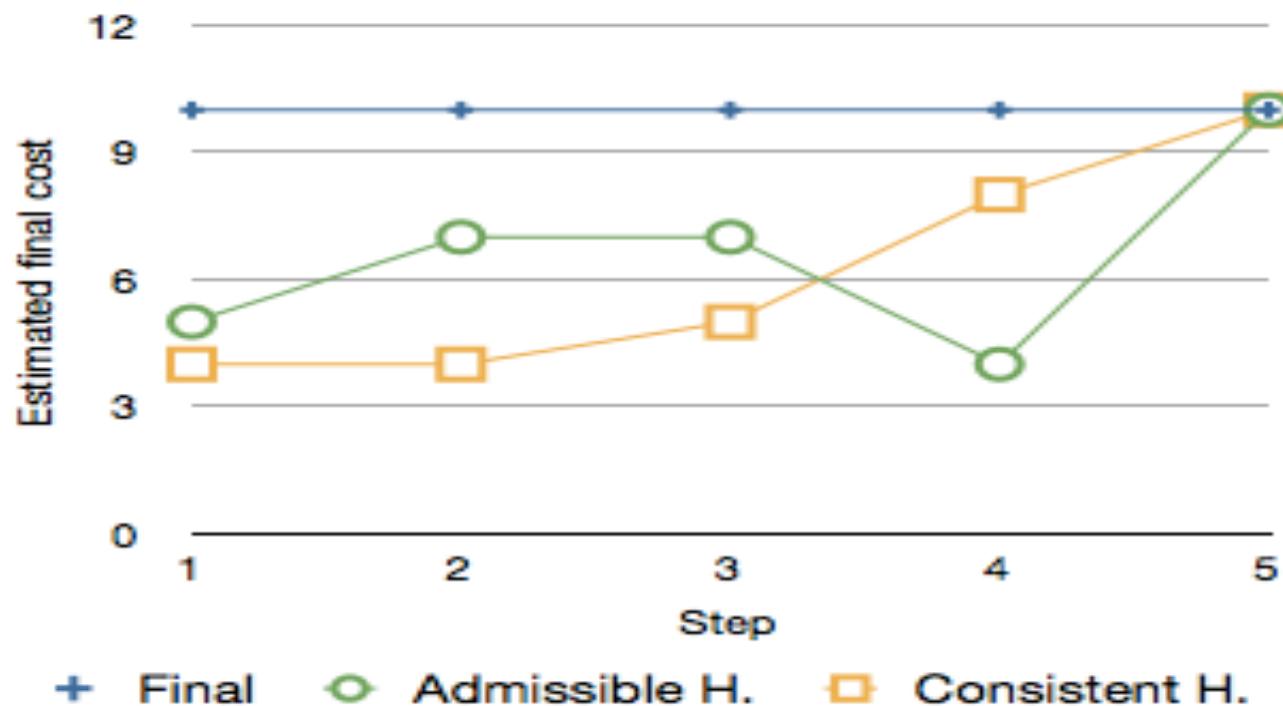
$7 \leq 1 + 1$ ✗

$$h(A) \leq c(A,C) + h(C) ?$$

$8 \leq 3 + 1$ ✗

Monotonidade

- As heurísticas consistentes são **monótonas**, i.e. garantem que o valor de f nunca diminui ao longo de um caminho



Heurísticas Consistentes

- h consistente $\rightarrow h$ admissível
 - Prova para TPC ☺
- h admissível $\not\rightarrow h$ consistente

Propriedades de A*

- Completa? Sim
 - exceto se o número de nós com $f \leq f(G)$ for infinito
- Tempo? Exponencial
- Espaço? Exponencial
 - mantém todos os nós em memória (no pior caso)
- Ótima? Sim
- **A* otimamente eficiente**
 - Para qualquer função heurística *consistente*, não há qualquer outro algoritmo que expanda menos nós
 - Não expandindo todos os nós com $f(n) < C^*$ corre-se o risco de perder optimalidade

Propriedades A*

- Complexidade exponencial no caso geral, mas muito dependente da **qualidade da heurística**
 - $\Delta \equiv h^* - h \rightarrow$ **erro absoluto da heurística**
 - $\varepsilon \equiv (h^* - h)/h^* \rightarrow$ **erro relativo da heurística**
- Se $h(n) = h^*(n)$, $\Delta = 0$ ($\varepsilon=0$)
 - A* só expande nós no caminho da solução ótima
- Se $h(n) = 0$, $\Delta = h^*(n)$ ($\varepsilon=1$)
 - Heurística é admissível pela definição, mas não dá informação nenhuma
 - A* equivale a procura de custo uniforme
 - A* vai expandir muitos nós

Propriedades A*

- Criar heurísticas admissíveis com erro pequeno nem sempre é fácil/possível
 - Se sacrificarmos otimalidade podemos usar heurística não admissível com um erro menor
 - Tempo de procura pode diminuir substancialmente

Propriedades A*

- Principal problema A*
 - Complexidade espacial exponencial
 - Precisamos de guardar todos os nós em memória
 - Ficamos sem espaço antes de ficar sem paciência...
- Idealmente...
 - Complexidade espacial $O(b \cdot d)$

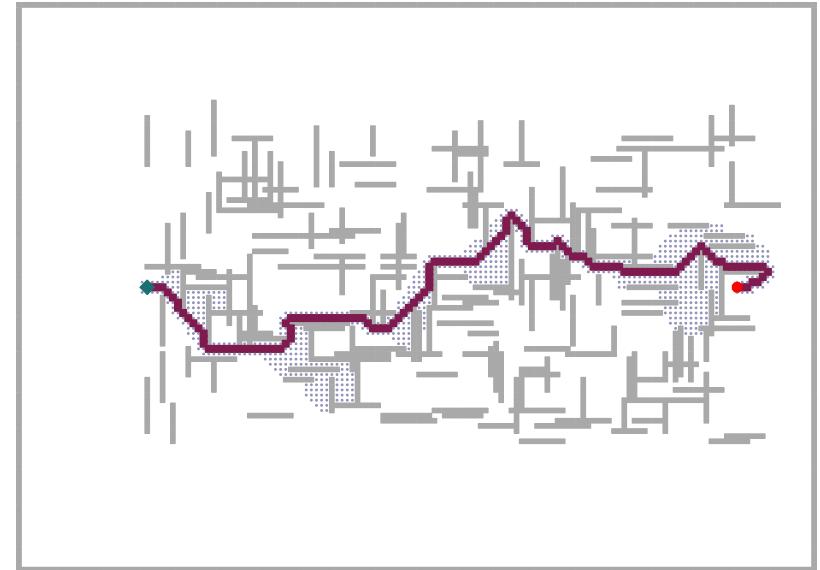
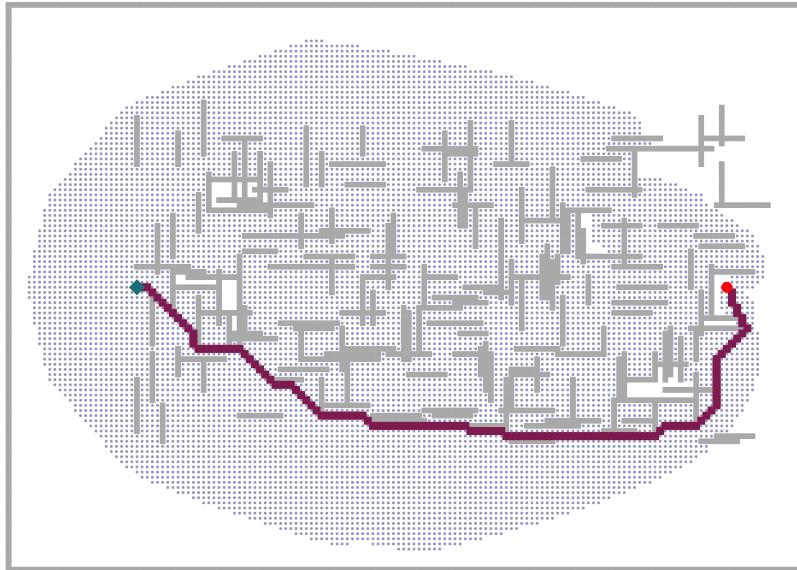
Procuras sub-ótimas

- Consideradas *good enough*
 - *Bounded sub-optimal search*
 - Procura por solução que está a uma distância de um fator constante $W (>0)$ da solução ótima
 - Exº weighted A* onde $f(n) = g(n) + W \times h(n)$
 - *Bounded-cost search*
 - Solução com custo inferior a constante C
 - *Unbounded-cost search*
 - Solução com qualquer custo
 - Por exº, *speedy search* – versão da procura gananciosa que usa como heurística estimativa do nº de ações até encontrar solução

Weighted A*

- $f(n) = g(n) + W \times h(n)$
- Custo da solução ótima C^*
- Custo da solução está entre C^* e $W \cdot C^*$

Weighted A*



- A* (imagem à esquerda)
- Weighted A* com $W=2$ (dir) explora 7x menos estados e encontra caminho com custo 5% superior

Procuras com memória limitada

- Utilização de memória?
 - Fronteira
 - Estados visitados (*reached*)
- Algumas formas de reduzir memória
 - Não haver interseção entre as duas estruturas
 - Remover estados de visitados quando podemos provar que já não vão ser necessários (separation property)
 - *Reference count*: definir limite de vezes que um nó pode ser alcançado; manter contador do nº de vezes que um nó foi alcançado

Beam search

- Limita o tamanho da fronteira
- Mantém os **k nós com os melhores valores de f**
 - Procura incompleta e sub-ótima
- Alternativa:
manter nós cujo valor de $f > \text{melhor_f} - \delta$

Complexidade espacial

- Exponencial na procura A*
- Linear para procura em profundidade
- Como combinar as duas para reduzir espaço?
 - Procura IDA*
 - Procura RBFS

IDA*

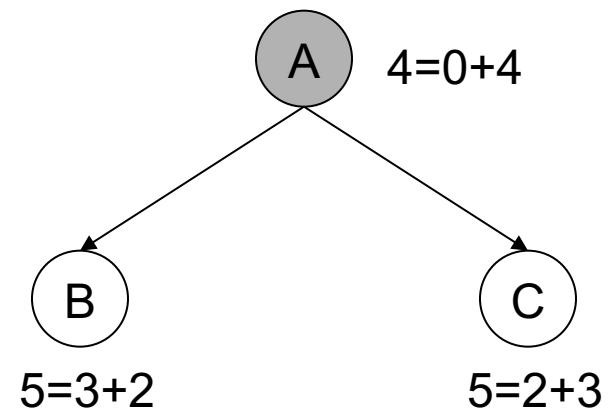
- IDA*: *Iterative Deepening A**
- Versão **iterativa** em profundidade da procura A*
- **Critério de corte/limite:**
 - $f(n) = g(n) + h(n)$
 - Em vez da profundidade!
 - Inicializado com $f(\text{estado inicial})$
- Em cada iteração é feita uma **procura em profundidade primeiro**
 - Se $f(n) > \text{limite}$ o nó é cortado
- Em cada nova iteração o valor limite é atualizado com o menor valor de $f(n)$ para os nós cortados na iteração anterior

IDA*

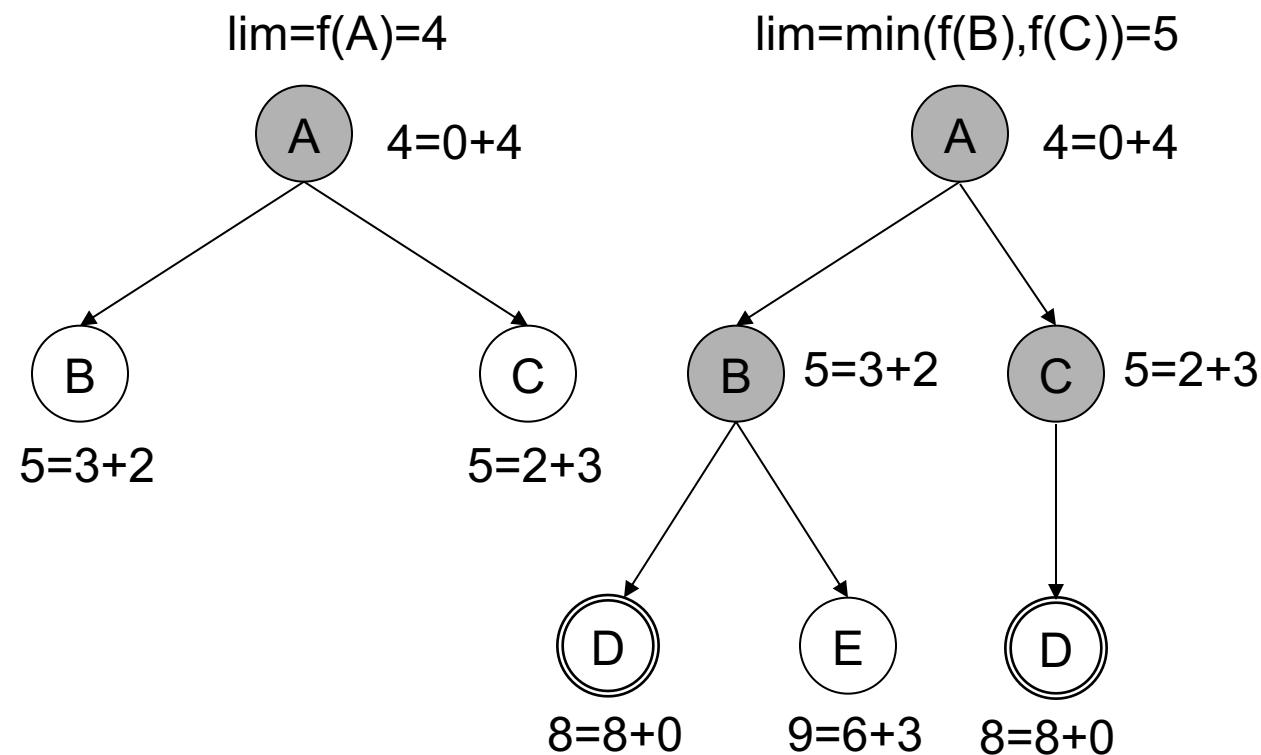
- Em cada iteração é feita uma **procura em profundidade** primeiro
- Ou seja, ...
- O valor de f de um nó **NÃO É** usado para ordenar os nós
 - Apenas para decidir se é cortado
- Critério de ordenação é a **profundidade**

IDA*: exemplo

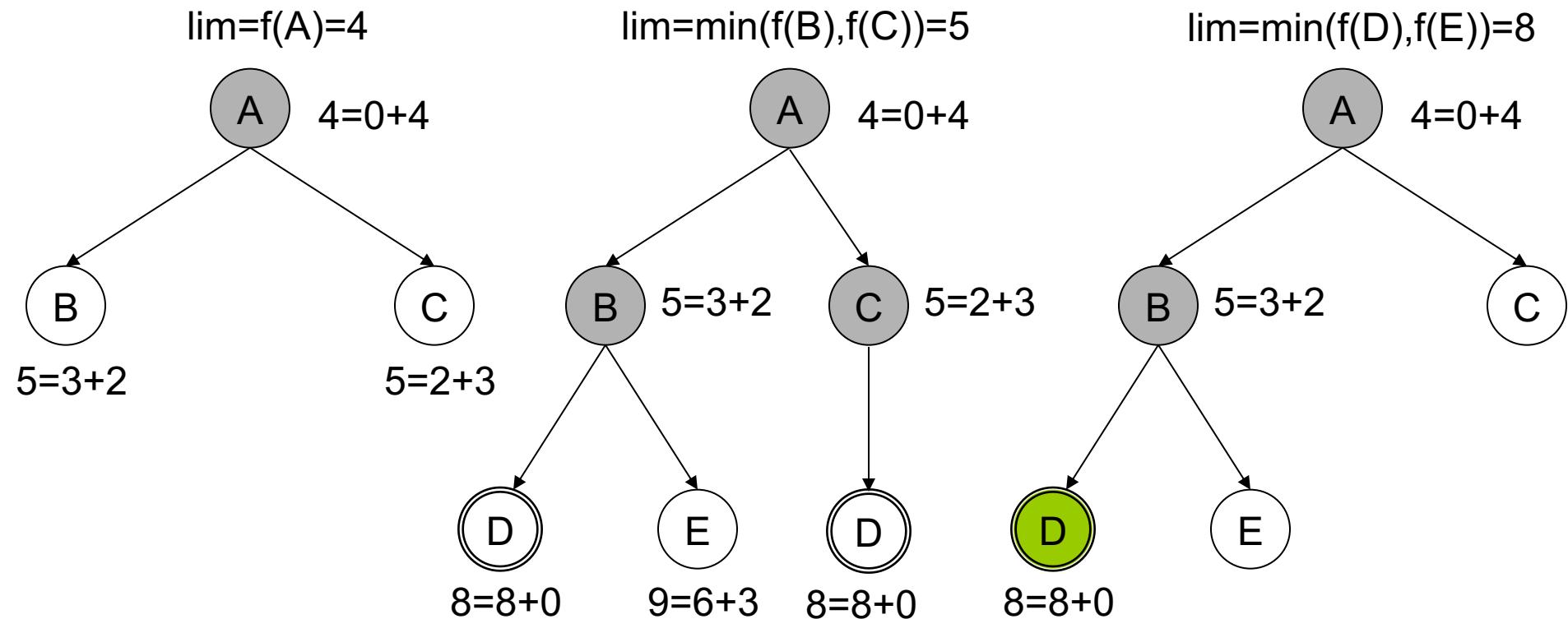
$\text{lim} = f(A) = 4$



IDA*: exemplo



IDA*: exemplo

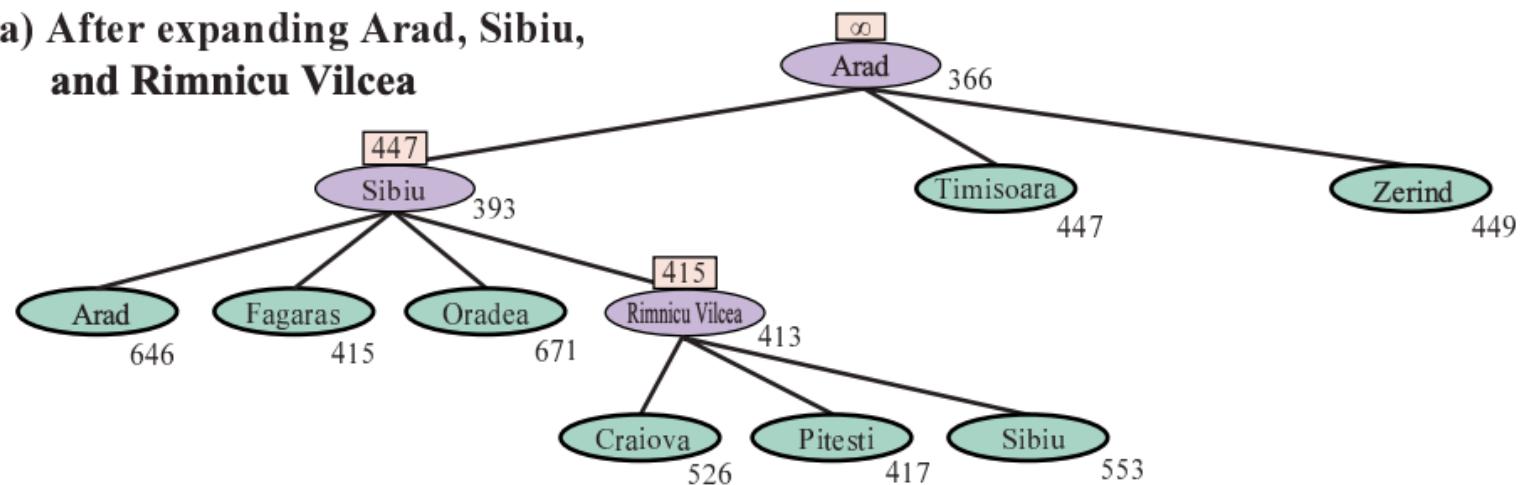


Melhor Primeiro Recursiva (RBFS)

- Melhor primeiro (A^*) com espaço linear (em d)
- Semelhante à procura em profundidade (implementação recursiva)
 - Para cada nó expandido, mantém **registo do caminho alternativo com menor valor de f**
 - Para **escolher** o próximo nó a expandir olha apenas para **os filhos do nó atual**
 - Se o valor de f para o melhor filho **excede o valor em memória**, a recursão permite recuperar o melhor caminho alternativo
- Uma alteração corresponde a uma iteração IDA*
- Ótima se $h(n)$ é admissível

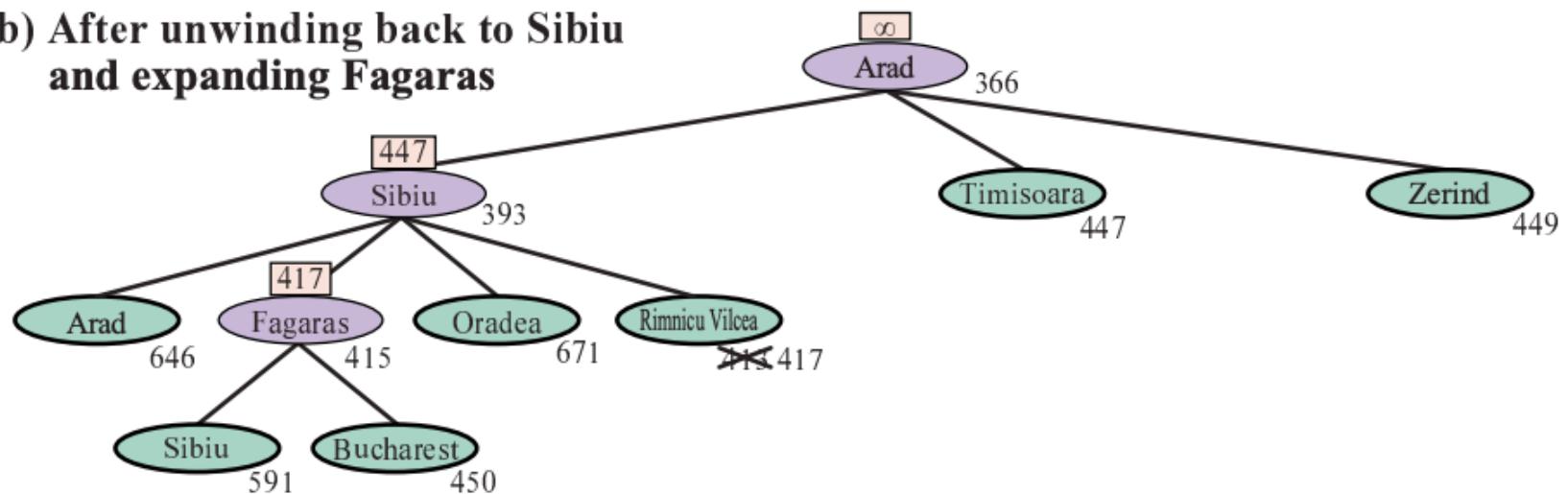
Melhor Primeiro Recursiva: exemplo

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea



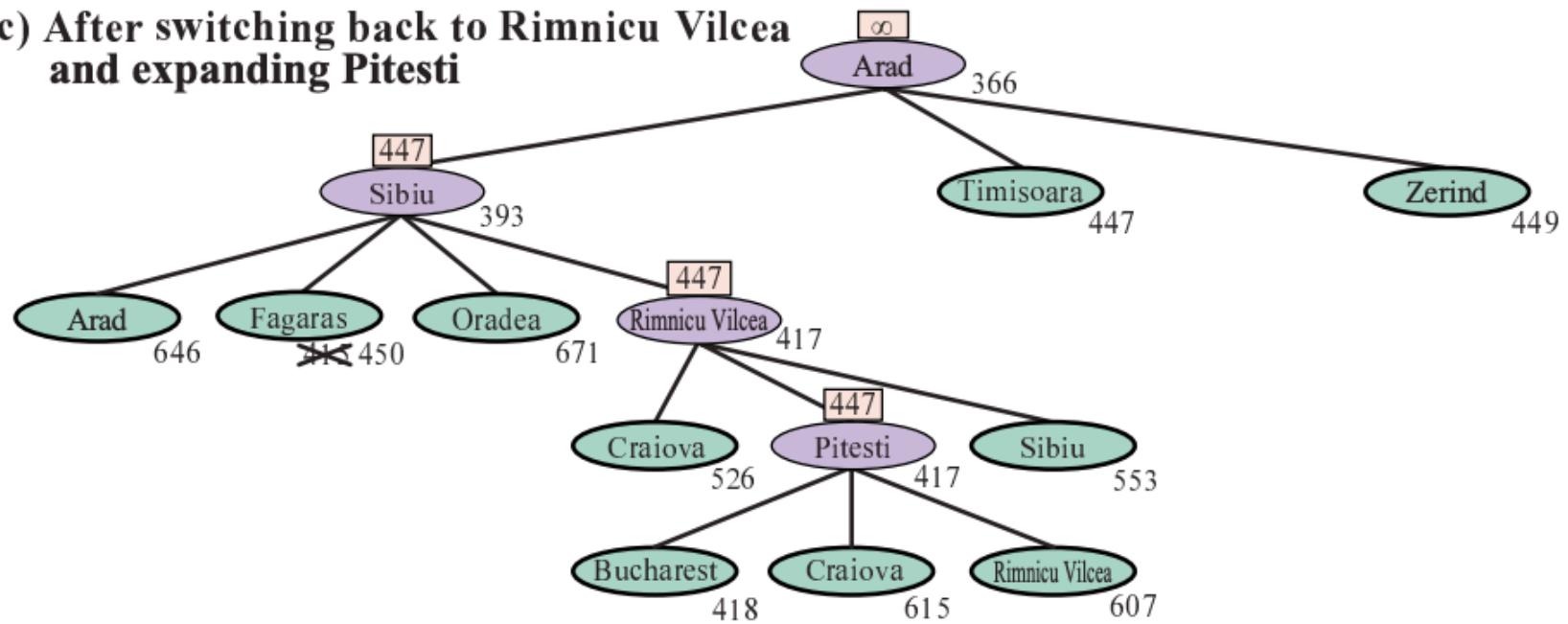
Melhor Primeiro Recursiva: exemplo

(b) After unwinding back to Sibiu
and expanding Fagaras



Melhor Primeiro Recursiva: exemplo

(c) After switching back to Rimnicu Vilcea and expanding Pitesti



Melhor Primeiro Recursiva (RBFS)

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution or failure
    solution, fvalue  $\leftarrow$  RBFS(problem, NODE(problem.INITIAL),  $\infty$ )
    return solution

function RBFS(problem, node, f_limit) returns a solution or failure, and a new f-cost limit
    if problem.IS-GOAL(node.STATE) then return node
    successors  $\leftarrow$  LIST(EXPAND(node))
    if successors is empty then return failure,  $\infty$ 
    for each s in successors do      // update f with value from previous search
        s.f  $\leftarrow$  max(s.PATH-COST + h(s), node.f)
    while true do
        best  $\leftarrow$  the node in successors with lowest f-value
        if best.f > f_limit then return failure, best.f
        alternative  $\leftarrow$  the second-lowest f-value among successors
        result, best.f  $\leftarrow$  RBFS(problem, best, min(f_limit, alternative))
        if result  $\neq$  failure then return result, best.f
```

Propriedades RBFS

- Ligeiramente mais eficiente que o IDA*
- Ainda sofre de demasiada regeneração de nós...
 - De vez em quando, muda de opinião
 - Cada mudança de opinião corresponde a uma iteração do IDA*
 - Muitas re-expansões de nós esquecidos
 - Apenas para recriar melhor caminho e estendê-lo com mais um nó

Propriedades RBFS

- **Completude**
 - Completa, se b finito e custo dos ramos $> \varepsilon$
- **Optimalidade**
 - Ótima, se heurística $h(n)$ for admissível
- **Complexidade espacial**
 - $O(b.d)$ - linear
- **Complexidade temporal**
 - Depende de
 - Precisão da heurística
 - Quantas vezes muda o melhor caminho à medida que os nós vão sendo expandidos

IDA* e RBFS

- Podem ver a complexidade temporal exponencialmente agravada com **procura em grafo**
 - Não podem detetar nós repetidos, a não ser no caminho atual
 - Podem re-expandir o mesmo estado muitas vezes
- Ambos sofrem por usar **demasiado pouca memória**

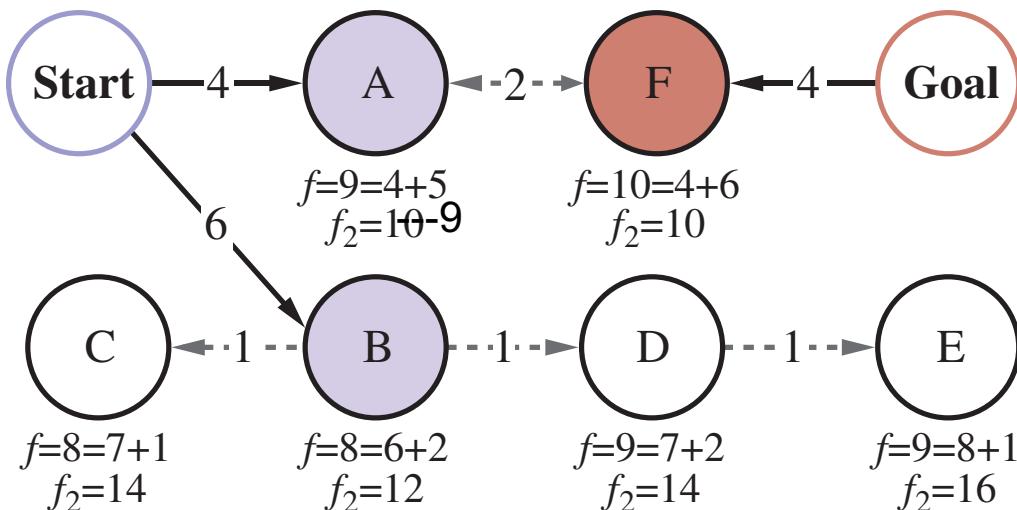
SMA*

- Simplified Memory A*
- Procede primeiro como o A*
 - Expande sempre o melhor nó na fronteira
 - Até a memória ficar cheia
- Quando a memória está cheia
 - Esquece o pior nó na fronteira para poder acrescentar outro
 - Mas, tal como o RBFS, regista o valor do nó esquecido no pai
- O nó esquecido pode ser regenerado se todos os caminhos forem piores do que o seu valor

Procura heurística bidirecional

- $f(n) = g(n) + h(n)$ não garante optimalidade
- Consideramos pares de nós...
- Necessidade de definir um lower bound no custo da solução

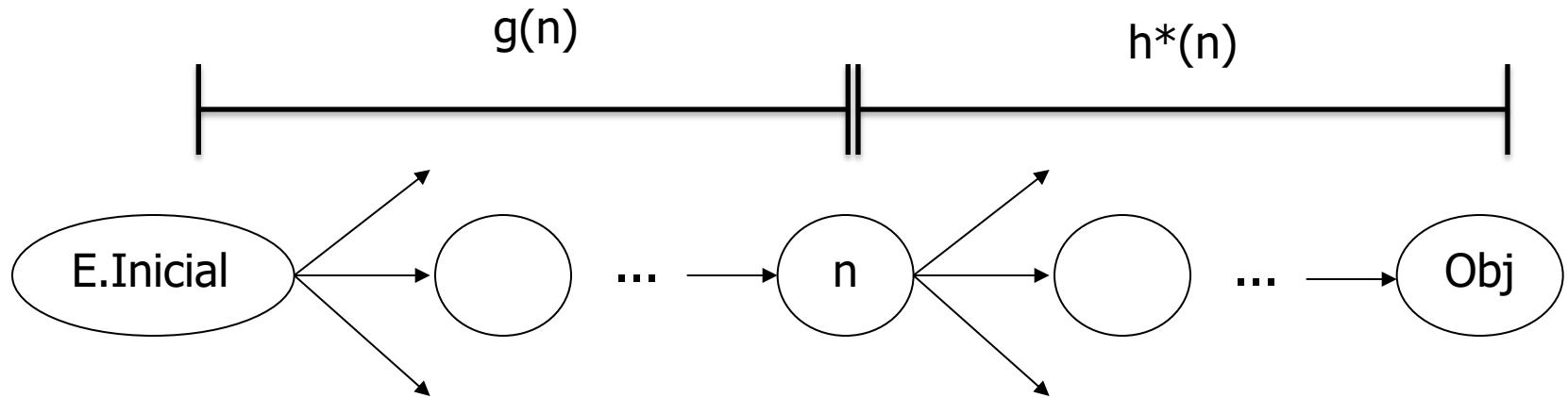
Procura heurística bidirecional



- A e B sucessores de Start, F antecessor de Goal
- Funções $f=g+h$ e $f_2=\max(2g,f)$
 - Expandir nós com menor valor de f_2
- Solução ótima Start–A–F–Goal com $C^*=10$
 - Nunca expandimos nós com $g > C^*/2 = 5$
- Completa e ótima com f_2 e h admissivel

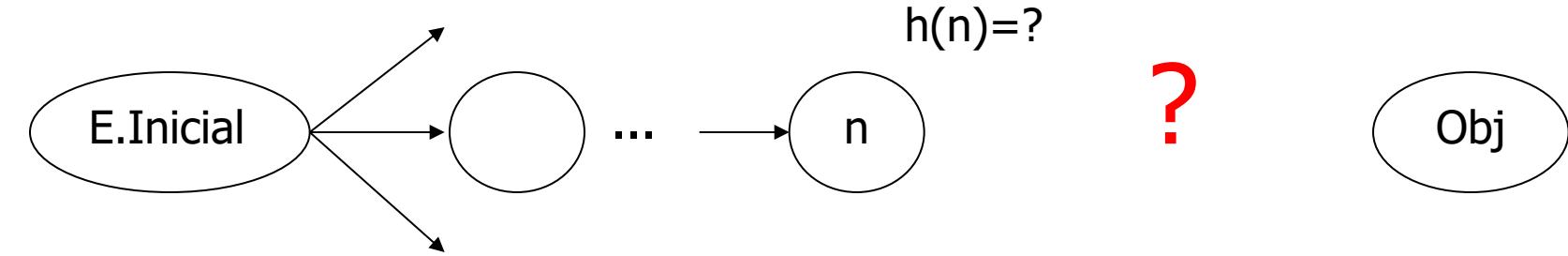
Funções Heurísticas

- $h^*(n)$ – custo do melhor caminho a partir do nó n até um objetivo



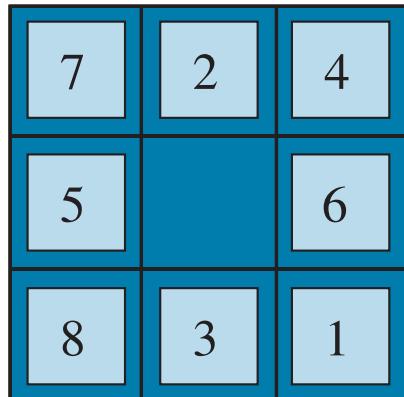
Funções Heurísticas

- $h(n)$ – **estimativa** do custo do melhor caminho a partir do nó n até um objectivo
- Mas como estimar $h^*(n)???$
 - Não sabemos o caminho
 - Apenas sabemos qual o estado n e o objectivo

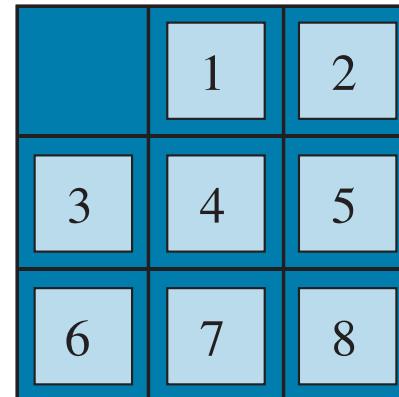


Funções Heurísticas

- Exemplo: **Problema 8-puzzle**
- Profundidade média solução: 22
- Fator ramificação: ~3
- Logo temos que considerar 3^{22} estados
 - Precisamos de uma boa heurística



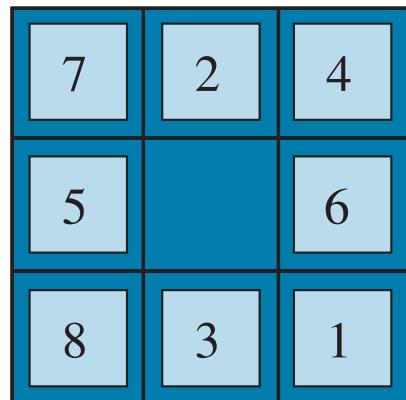
Start State



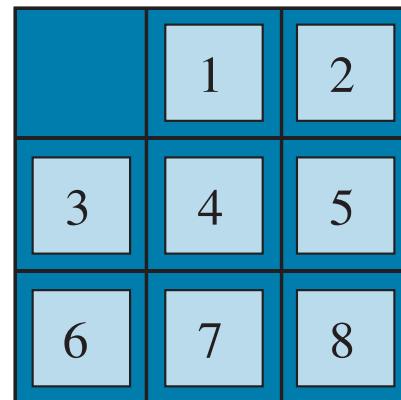
Goal State

Funções heurísticas para 8-puzzle

- Qual o **número mínimo de passos necessários** para ir do estado inicial ao final?
- Só saberemos ao certo depois de resolver o problema
- Mas podemos estimar um valor por baixo



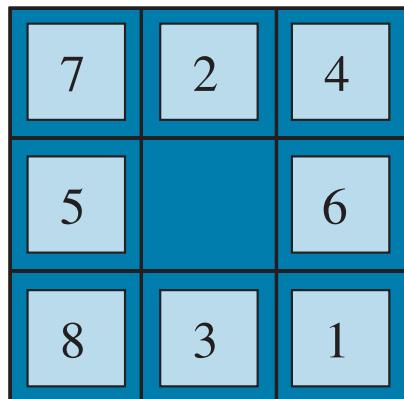
Start State



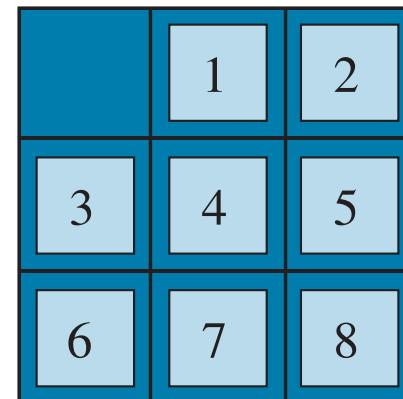
Goal State

Funções heurísticas para 8-puzzle

- $h_1(n)$ = número de peças mal colocadas (i.e. fora do sítio)
 - Peça na posição final: não necessita de qualquer deslocação
 - Peça fora da posição final: apenas um passo para chegar à posição final (muito otimista!)
 - Heurística admissível
 - Peça fora de posição movida pelo menos uma vez para chegar à posição final
- $h_1(\text{StartState}) = ?$ 8 (todas as peças estão fora de posição)



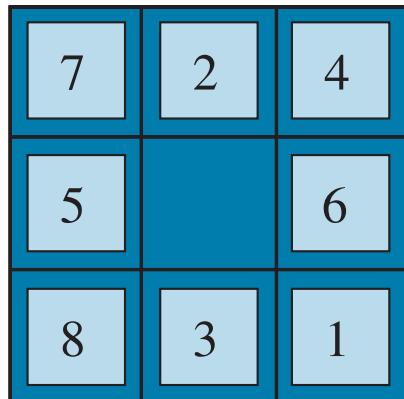
Start State



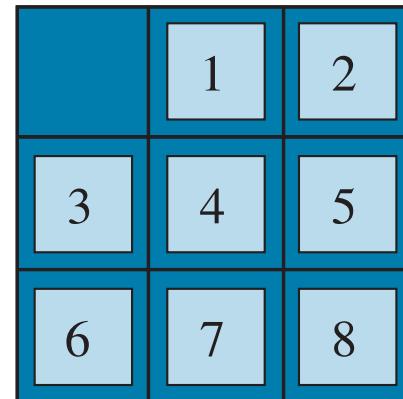
Goal State

Funções heurísticas para 8-puzzle

- $h_2(n)$ = soma da distância de Manhattan (i.e., nº de quadrados até à localização desejada para cada peça)
 - Se a peça está na posição final, não precisamos de mover
 - Se a peça não está na posição final, precisamos de fazer movimentos horizontais e verticais para a colocar na posição final.
 - Heurística admissível – cada jogada só move uma peça uma posição horizontal ou uma posição vertical.
- $h_2(\text{StartState}) = ? \quad 3+1+2+2+2+3+3+2 = 18 (< \text{custo real } 26)$



Start State



Goal State

Qualidade de uma heurística

- Seja N o número de **nós gerados** por uma procura A*
- Seja d a profundidade da solução
- $b^* = \text{fator de ramificação}$ que uma árvore uniforme de profundidade d teria de ter para conter $N + 1$ nós
 - $N+1=1+b^*+(b^*)^2 + \dots + (b^*)^d$
- Exº para $d=5$ e $N=52$ temos $b^*=1.92$
- Número de nós gerados variam muito com a profundidade d
- b^* varia pouco com a profundidade
 - Bom para comparar procura e heurísticas

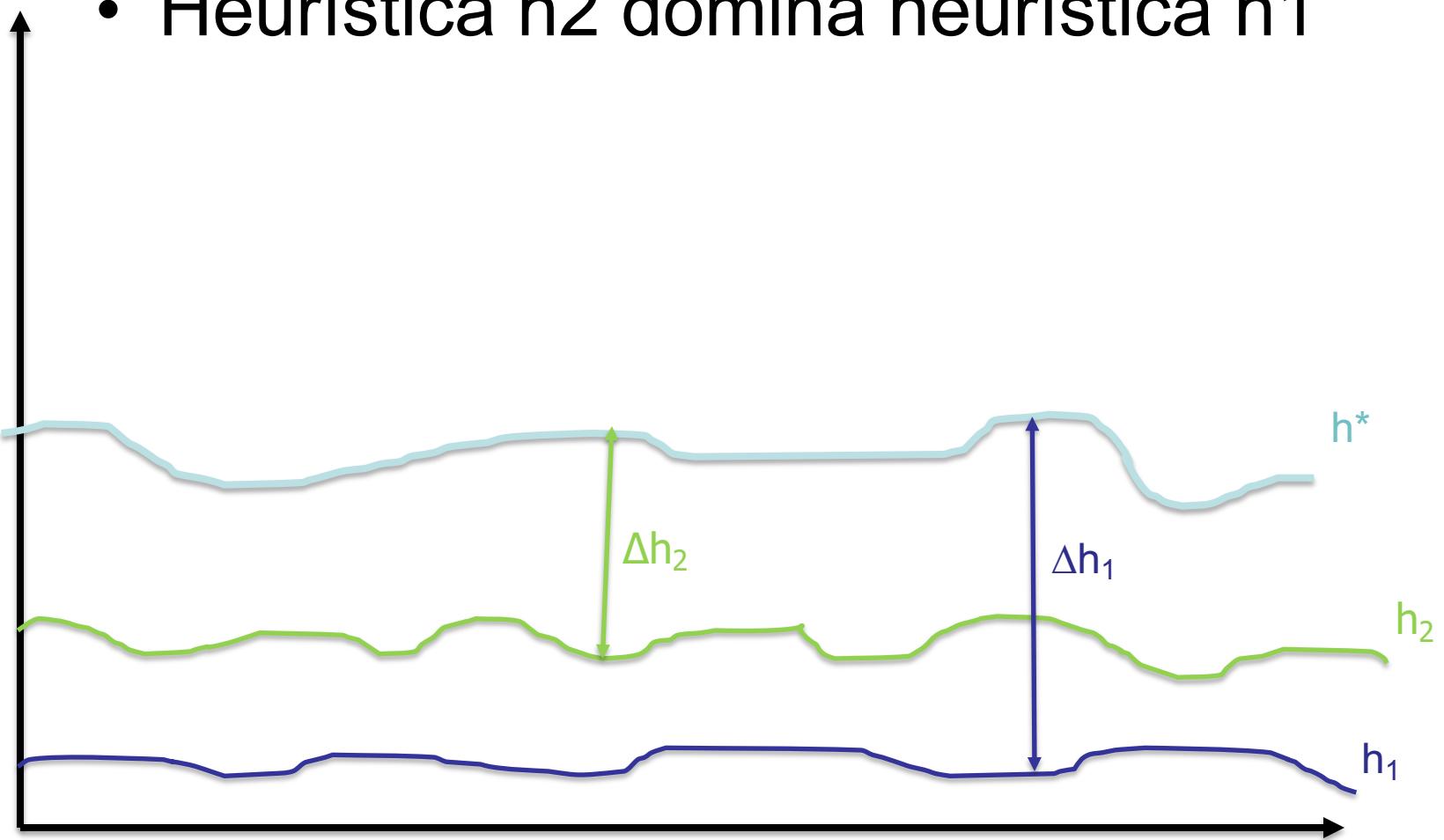
Comparar funções heurísticas

d	Search Cost (nodes generated)			Effective Branching Factor		
	BFS	A*(h_1)	A*(h_2)	BFS	A*(h_1)	A*(h_2)
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	4102	751	1.72	1.49	1.34
20	91493	9905	1318	1.69	1.50	1.34
22	175921	22955	2548	1.66	1.50	1.34
24	290082	53039	5733	1.62	1.50	1.36
26	395355	110372	10080	1.58	1.50	1.35
28	463234	202565	22055	1.53	1.49	1.36

Média para 100 puzzles

Dominância entre heurísticas

- Heurística h_2 domina heurística h_1



Dominância

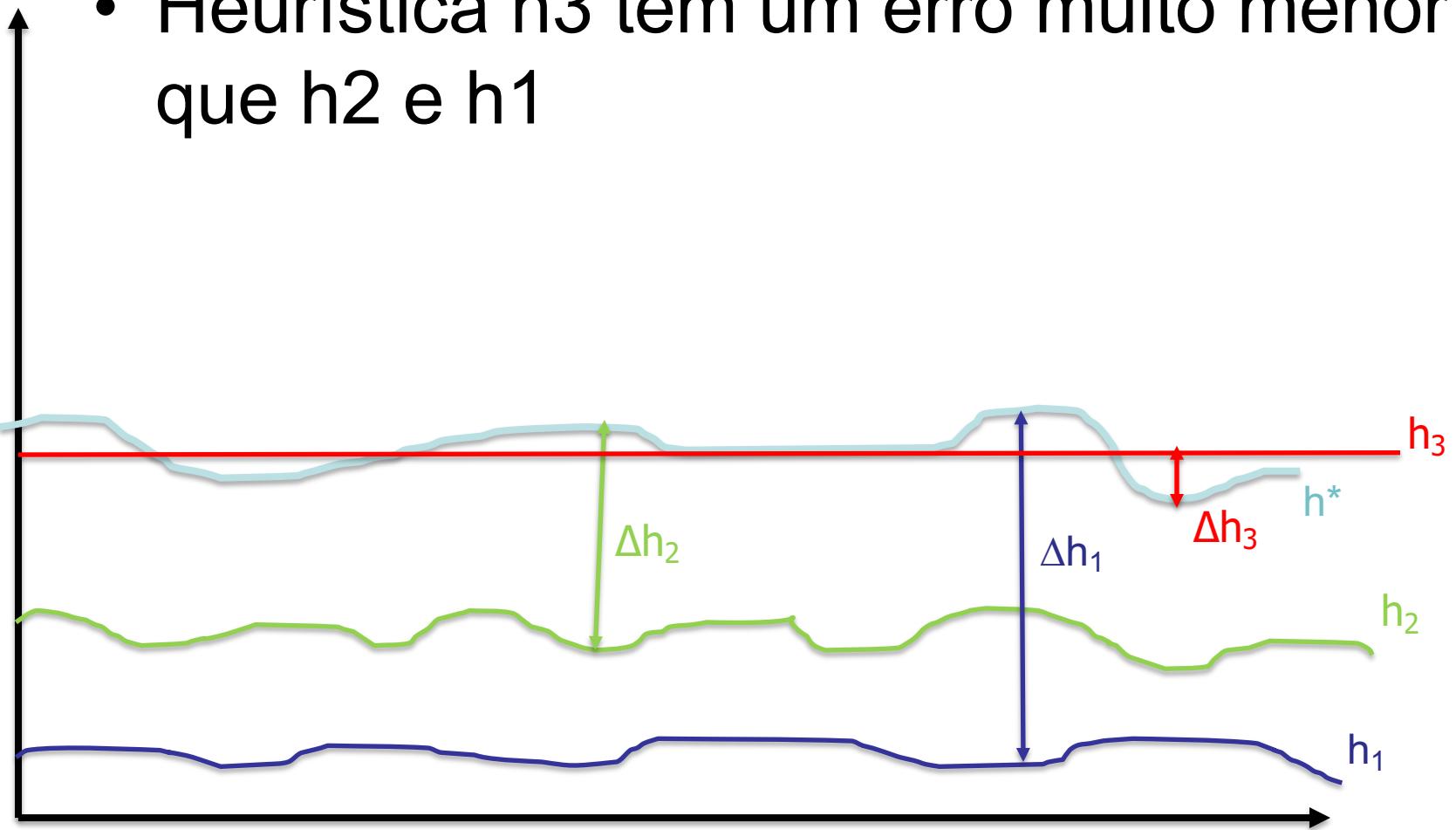
- Se $h_2(n) \geq h_1(n)$ para todos nós n
 - então h_2 **domina** h_1
- Dominância → Eficiência
 - $A^*(h2)$ nunca irá expandir mais nós que $A^*(h1)$
- h_2 é melhor para a procura
 - Expande menos nós porque não é tão otimista / tem um erro menor

Heurísticas não admissíveis

- Por vezes, se a solução ótima não for imprescindível
 - Podemos considerar heurísticas não admissíveis
 - Se tiverem um **erro baixo** são muito eficientes a encontrar uma solução
 - Embora **não garantam a solução ótima**

Heurísticas não admissíveis

- Heurística h_3 tem um erro muito menor que h_2 e h_1



Heurísticas Admissíveis: como inventá-las?

- Heurísticas h_1 e h_2 do exemplo 8-puzzle são estimativas do custo para o problema
 - Mas são também medidas reais de versões simplificadas do problema
 - Por simplificado, queremos dizer com menos restrições nas ações que podem ser feitas
- Imaginem um “batoteiro” que para resolver o problema, simplesmente tira as peças mal colocadas do jogo, e coloca-as na posição final desejada.
 - Heurística h_1 retorna o custo exato da melhor solução para o problema: retirar as 8 peças do puzzle e voltar a colocá-las.
- Um problema com menos restrições nas ações consideradas é chamado **problema relaxado**

Heurísticas Admissíveis: como inventá-las?

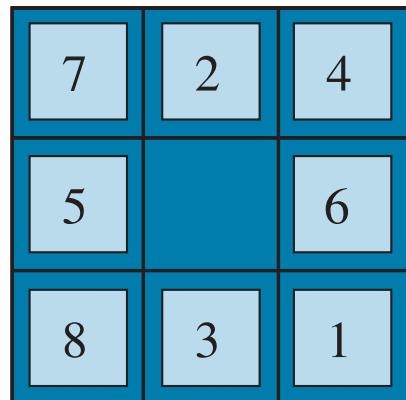
- O **custo** de uma solução ótima para um **problema relaxado** corresponde a uma **heurística admissível e consistente** para o problema original
- Exemplo do 8-puzzle:
 - Uma peça pode mover-se de uma posição A para B se:
 - A é verticalmente ou horizontalmente adjacente a B
 - E B está vazio
 - Podemos gerar 3 problemas relaxados:
 - Uma peça pode mover-se de A para B sempre → $h_1(n)$
 - Uma peça pode mover-se de A para B se A for adjacente a B → $h_2(n)$
 - Uma peça pode mover-se de A para B se B está vazio → ??

Heurísticas Admissíveis: como inventá-las?

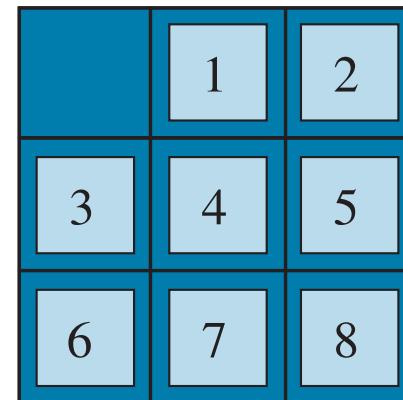
- Problema relaxado:
 - Uma peça pode mover-se de A para B se B está vazio
 - Ideia para nova heurística h_3
- Ideia para heurística problema relaxado
 - Para cada peça
 - Se peça está na **posição final**
 - Custo 0
 - Se **posição final da peça está vazia** atualmente
 - Custo 1, basta fazer um salto
 - Se **posição final da peça não está vazia**
 - Custo 2, temos de fazer dois saltos

Heurísticas Admissíveis: como inventá-las?

- $h_3(\text{StartState}) = 2 + 2 + 2 + 1 + 2 + 2 + 2 + 2 = 15$
Somente a peça 4 tem o custo 1



Start State



Goal State

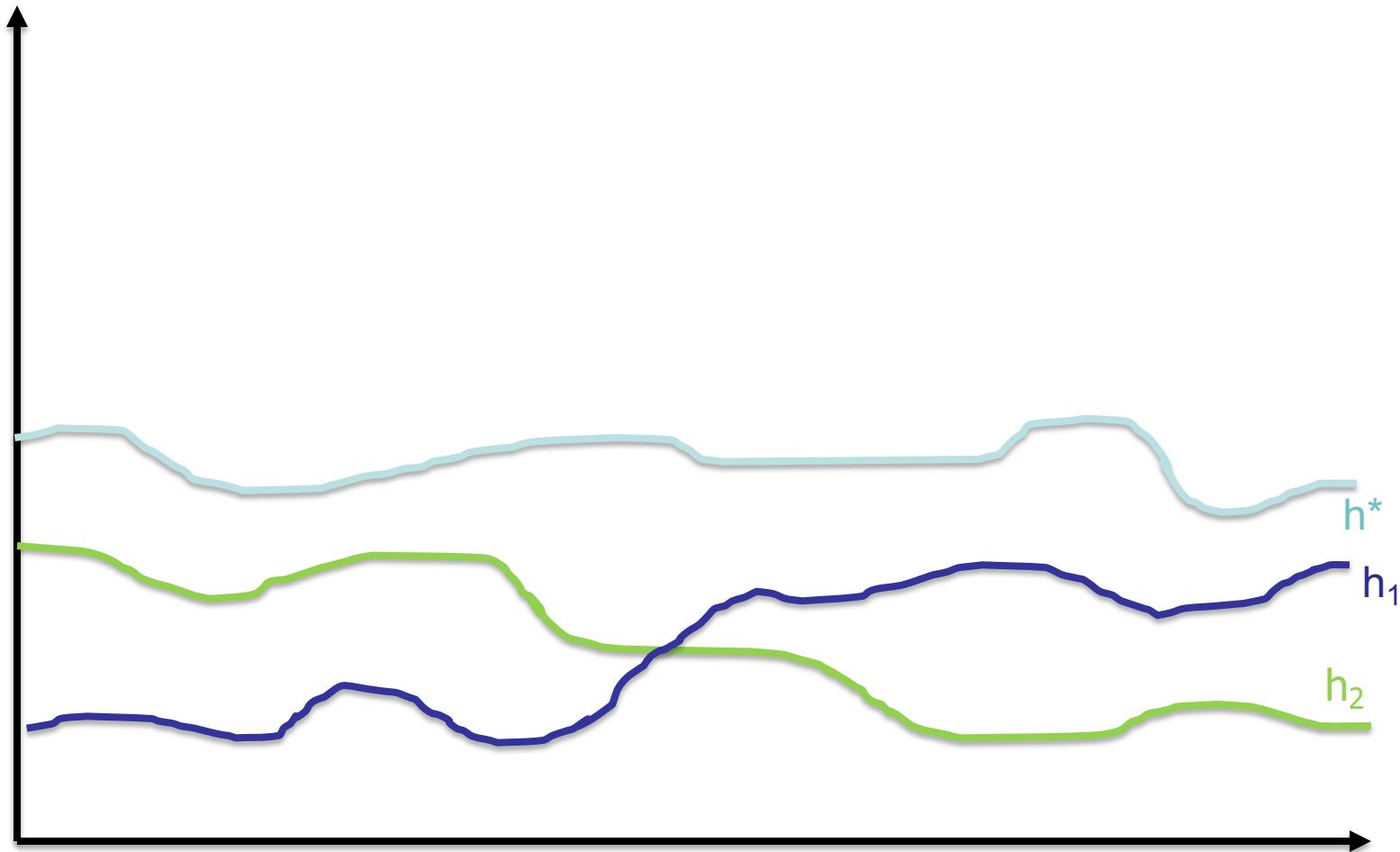
Heurísticas Admissíveis: como inventá-las?

- Importante
 - Problemas relaxados devem ser resolvidos sem recorrer a procura
 - Heurísticas têm que ser eficientemente calculadas
- É necessário pesar a **precisão da heurística** com o seu **custo** (tempo que leva a calcular)

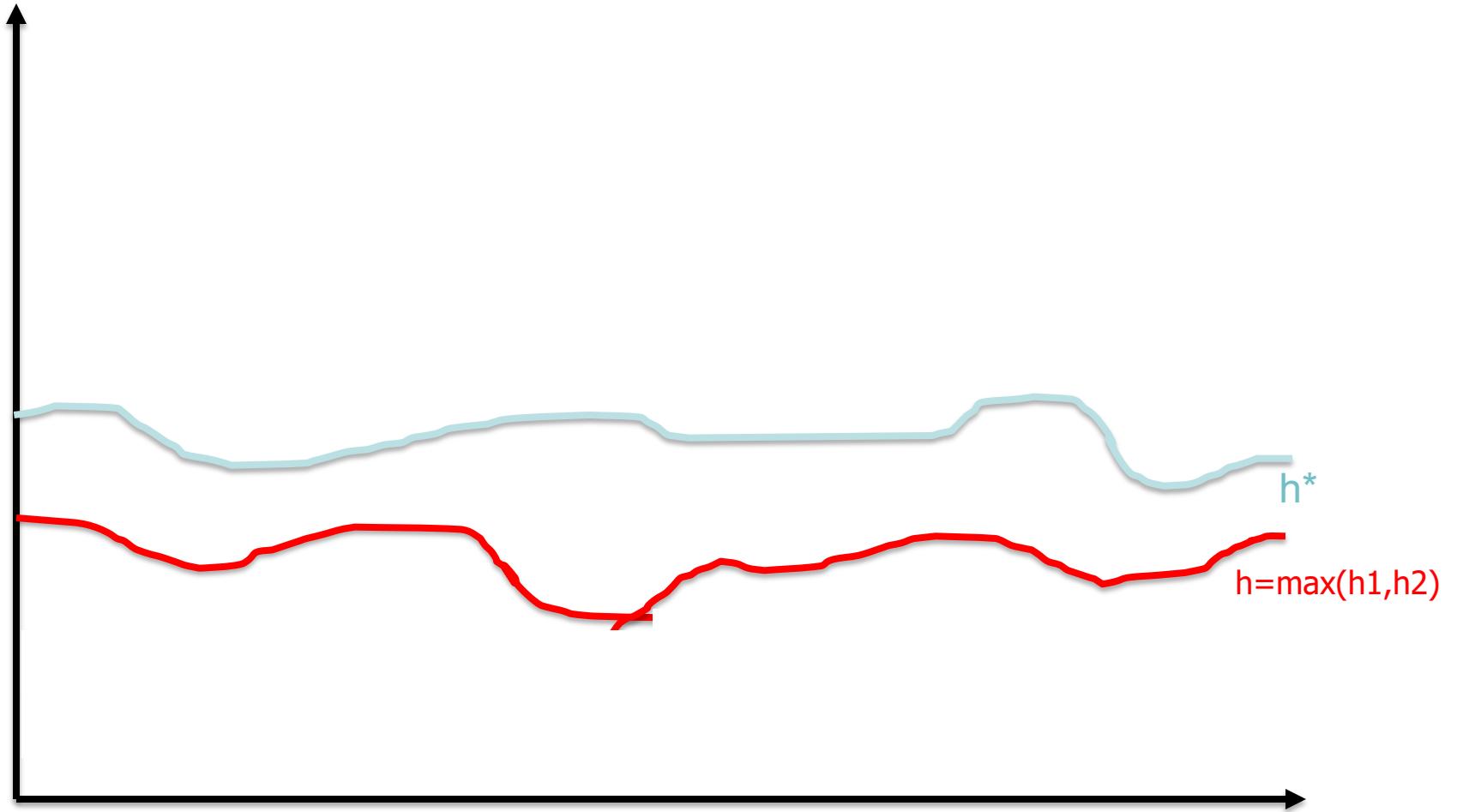
Heurísticas Admissíveis: como inventá-las?

- Que fazer se temos várias heurísticas admissíveis h_1, h_2, \dots, h_k
 - E nenhuma delas domina todas as outras
 - Não precisamos de escolher ☺
 - $h(n) = \max(h_1(n), \dots, h_k(n))$
 - Se todas as heurísticas forem admissíveis, $h(n)$ também é, e domina todas elas

Heurísticas Admissíveis: como combiná-las?

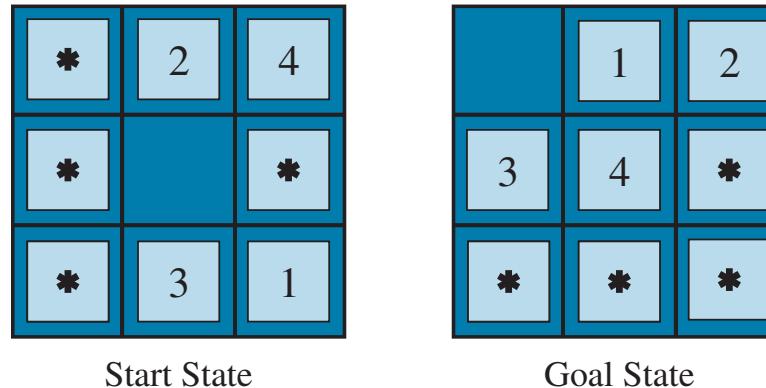


Heurísticas Admissíveis: como combiná-las?



Heurísticas Admissíveis: como inventá-las?

- Outra alternativa para criar heurísticas
 - Usar o custo de **solução** de um sub-problema do problema dado
 - Ex: colocar as peças 1,2,3,4 na posição final
 - Heurística admissível para o 8-puzzle



Heurísticas Admissíveis: como inventá-las?

- **Bases de dados de padrões**
 - Guardar **custos** exatos das soluções para todas as possíveis configurações iniciais
 - das peças 1,2,3,4
 - movimentos das peças * contam para o custo
 - Heurística obtida procurando a configuração atual do estado na base de dados
 - Base de dados construída ao fazer uma procura regressiva a partir do objetivo
 - E guardando na BD o custo de cada estado gerado
 - Custo de geração da BD amortizado ao longo de muitas procuras

Heurísticas Admissíveis: como inventá-las?

- **Bases de dados de padrões**
 - Podemos construir BD para outros sub-problemas
 - 5,6,7,8
 - 2,3,4,5
 - Podemos **combinar as heurísticas** de cada sub-problema calculando o **máximo** valor delas
 - E não podemos somar? $(1,2,3,4) + (5,6,7,8)$
 - Se os sub-problemas não forem independentes não, porque perdemos admissibilidade
 - Movimentos partilhados entre os 2 sub-problemas
 - No entanto, se o custo usado não contabilizar movimentos partilhados
 - Ex: não contabilizar os movimentos das peças *
 - Aí já podemos somar o valor das duas heurísticas

Heurísticas Admissíveis: como inventá-las?

- Aprender através da experiência
 - Resolver muitas instâncias diferentes do 8-puzzle
 - Cada solução ótima contém **vários exemplos** que podem ser usados para aprender $h(n)$
 - Um exemplo consiste num estado do caminho da solução e respetivo custo da solução a partir daquele ponto
 - Aprendizagem usando características (*features*) do estado
 - $x_1(n)$ – número de peças fora da posição desejada
 - $x_2(n)$ – número de pares de peças adjacentes que não são adjacentes no estado final

Heurísticas Admissíveis: como inventá-las?

- Combinação linear de várias features
 - $h(n) = c_1x_1(n) + c_2x_2(n)$
- Coeficientes c_1 e c_2 *aprendidos* e ajustados de modo a melhor se adaptarem aos exemplos usados
- Não se garante admissibilidade