TÉCNICO LISBOA

# Estratégias de Procura Informadas

TÉCNICO LISBOA

# Livro

- Capítulo 3 Secções 5 - 6

TÉCNICO LISBOA

# Resumo

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

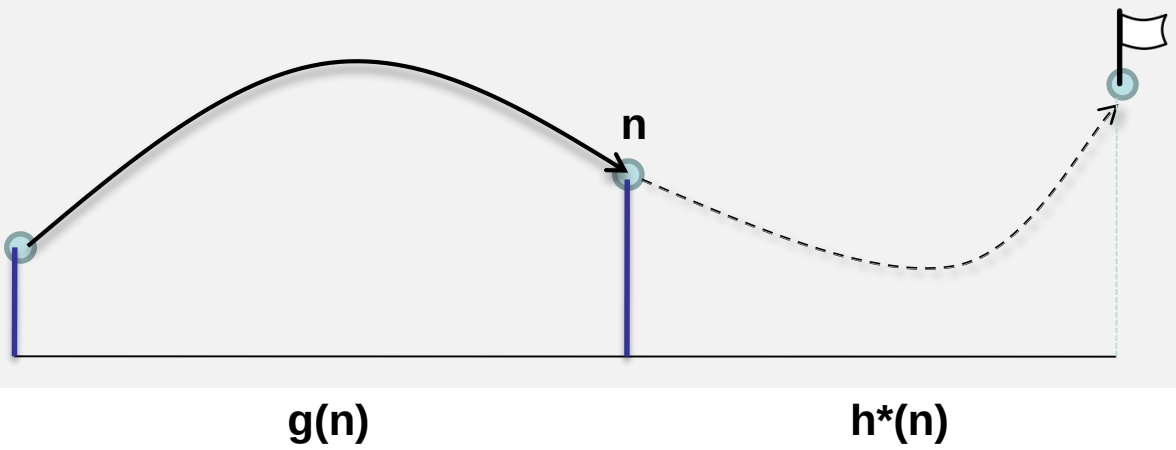
TÉCNICO LISBOA

# Árvore de Procura

- Uma estratégia de procura determina a ordem de expansão dos nós
- As procuras 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 objectivo
- 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)* usa conhecimento específico do problema
  - O "melhor" nó é o que tem o menor valor de *f(n)*

  → Expandir primeiro o nó que tem o menor valor de *f(n)*
- Implementação:

  Nós na fronteira ordenados por ordem crescente da função de avaliação
  - Fronteira = $\{n_1, n_2, n_3, \ldots\}$ → $f(n_1) \leq f(n_2) \leq f(n_3) \leq \ldots$
- Casos especiais:
  - Procura Gananciosa
  - Procura $A^*$

TÉCNICO LISBOA

# Procura Melhor Primeiro

- Algoritmo Procura Melhor Primeiro
  - Igual à Procura Custo Uniforme
  - Fronteira ordenada por f(n) em vez do custo
  - Procura custo uniforme pode ser vista como Procura Melhor Primeiro
    - f(n) = g(n)

# P. Melhor Primeiro (Árvore)

```
function BEST-FIRST-SEARCH (problem) returns a solution, or failure
    node ← a node with State = problem.Initial-State, Path-Cost = 0
    frontier ← a priority queue ordered by F(node), with node as the only element

    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)
        if problem.Goal-Test(node.State) then return Solution(node)

        for each action in problem.Actions(node.State) do
            child ← CHILD-NODE(problem,node,action)

            frontier ← Insert(child,frontier)
```

TÉCNICO LISBOA

# P. Melhor Primeiro(Grafo)

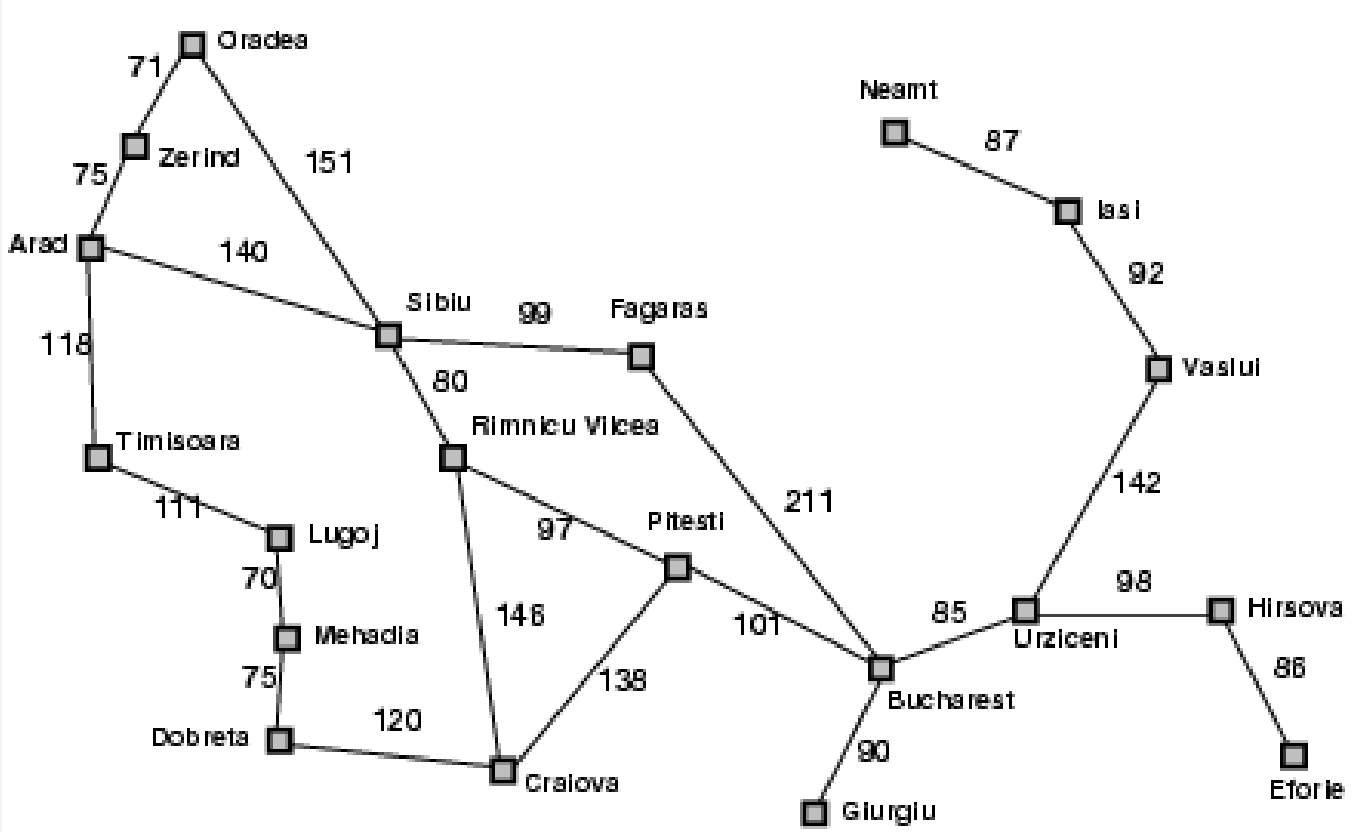
```
function BEST-FIRST-SEARCH (problem) returns a solution, or failure
    node ← a node with State = problem.Initial-State, Path-Cost = 0
    frontier ← a priority queue ordered by F(node), with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)
        if problem.Goal-Test(node.State) then return Solution(node)
        add node.State to explored
        for each action in problem.Actions(node.State) do
            child ← CHILD-NODE(problem,node,action)
            if child.State is not in explored or frontier then
                frontier ← Insert(child,frontier)
            else if child.State is in frontier with higher F-Value then
                replace that frontier node with child
```

TÉCNICO LISBOA

# Procura gananciosa

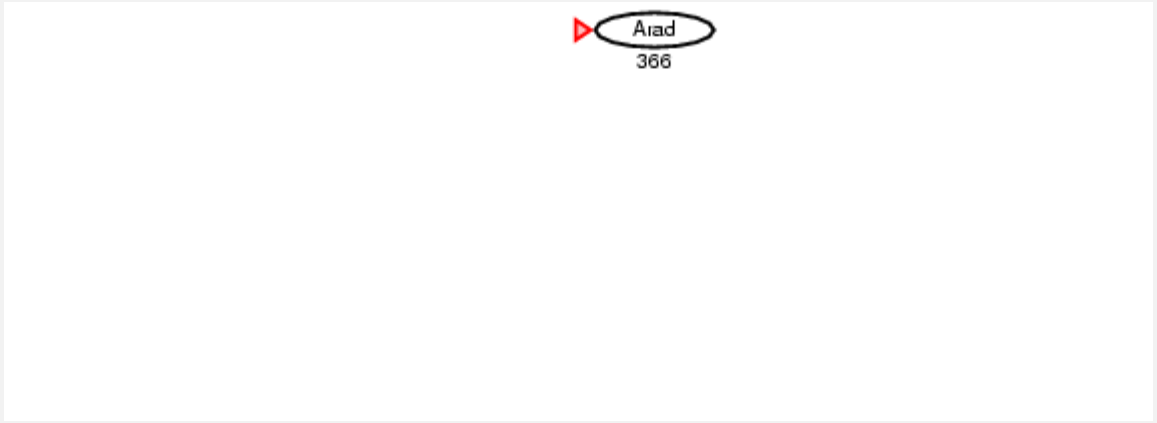
- Função de avaliação $f(n) = h(n)$ (heurística)
- = estimativa do custo do caminho desde $n$ até ao objectivo
- e.g., $h_{dlr}(n)$ = distância em linha recta desde $n$ até Bucareste
- Procura gananciosa expande o nó que "parece" estar mais próximo do objectivo

# Roménia f(n) = distância linha recta (km)



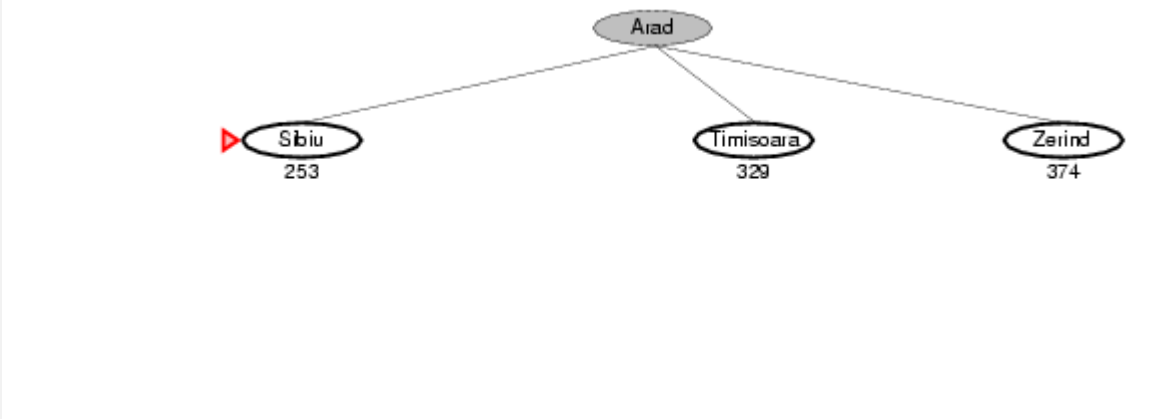

| Straight–line distance to Bucharest | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Dobreta | 242 |
| Eforie | 161 |
| Fagaras | 176 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Iasi | 226 |
| Lugoj | 244 |
| Mehadia | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 10 |
| Rimnicu Vilcea | 193 |
| Sibiu | 253 |
| Timisoara | 329 |
| Urziceni | 80 |
| Vaslui | 199 |
| Zerind | 374 |

# Procura Gananciosa: exemplo



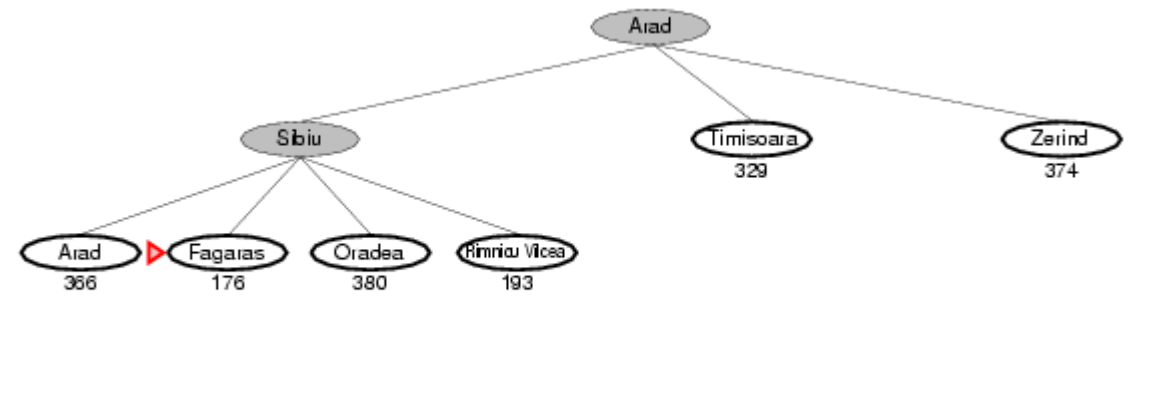

**Fronteira = {Arad(366)}**

# Procura Gananciosa: exemplo



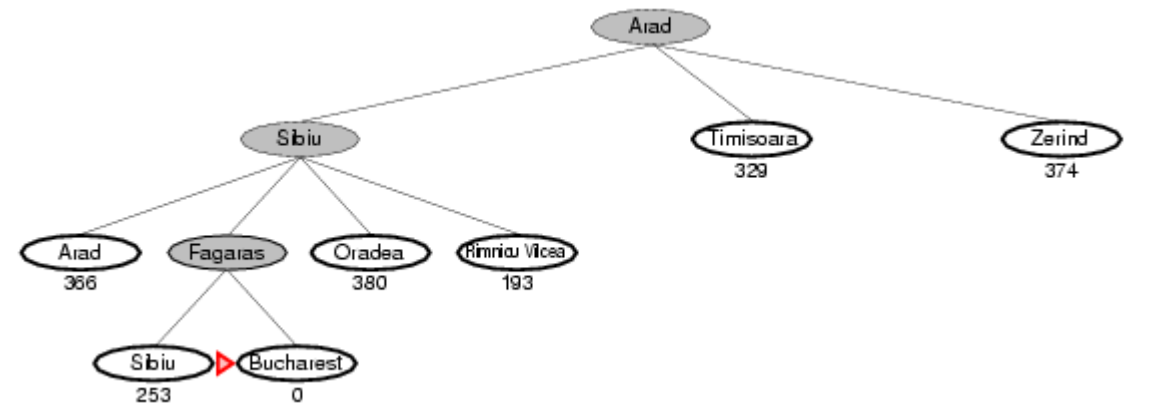

**Fronteira = {Sibiu(253),Timisoara(329), Zerind(374)}**

# 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
- Óptima? Não

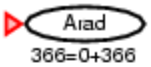
- Semelhante à procura em profundidade, mas mais exigente em memória (como a procura em largura)

TÉCNICO LISBOA

# 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 desde $n$ até um estado objectivo
- $f(n)$ = estimativa do custo total da melhor solução que passa por n
  - caminho desde o estado inicial até estado objectivo (passando por $n$)

# Procura A*: exemplo

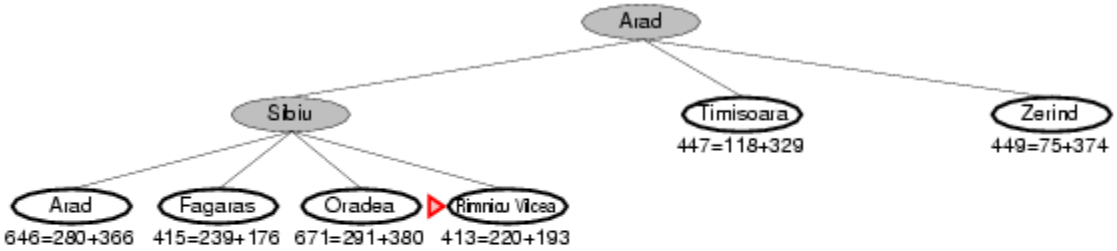



**Fronteira = {Arad(366)}**

# Procura A*: exemplo

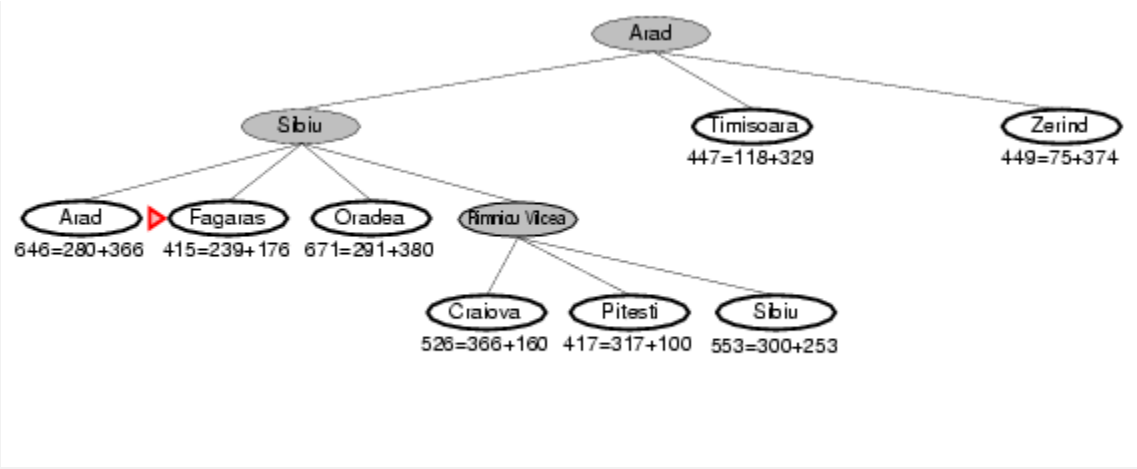

**Fronteira = {Sibiu(393), Timisoara(447), Zerind(449)}**

# Procura A*: exemplo



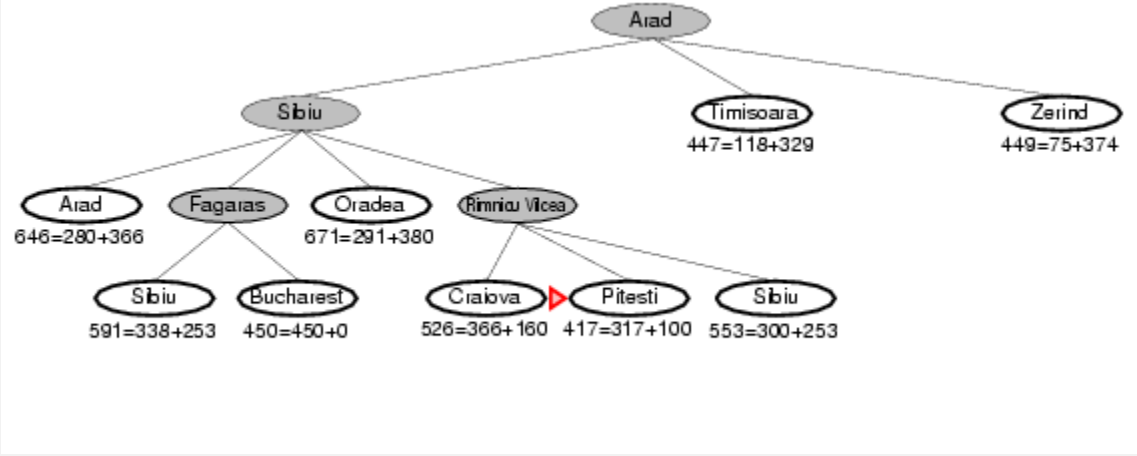

**Fronteira = {Rimnicu Vilcea(413), Fagaras(415), Timisoara(447), Zerind(449), Arad(646),Oradea(671)}**

# Procura A*: exemplo



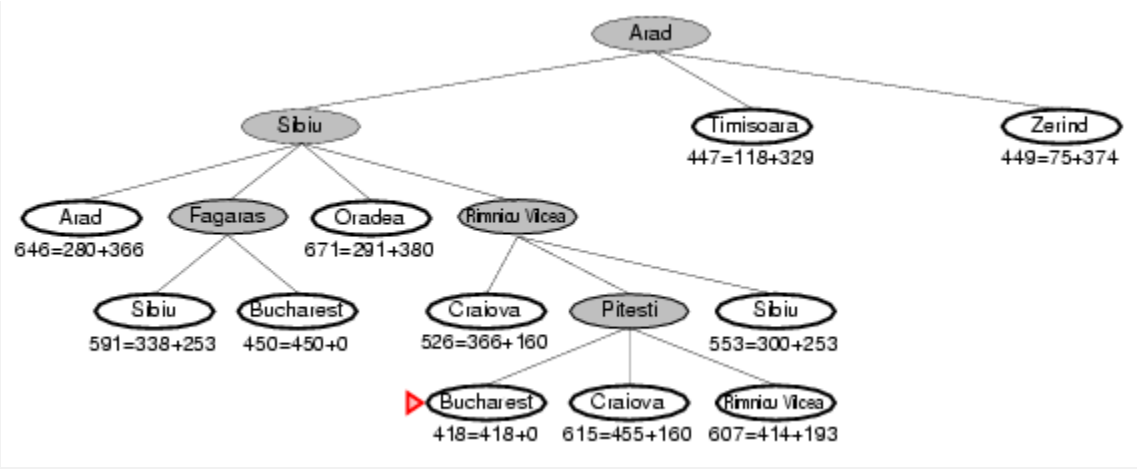

**Fronteira = {Fagaras(415), Pitesti(417), Timisoara(447), Zerind(449), Craiova(526), Sibiu(553), Arad(646), Oradea(671)}**

# Procura A*: exemplo




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

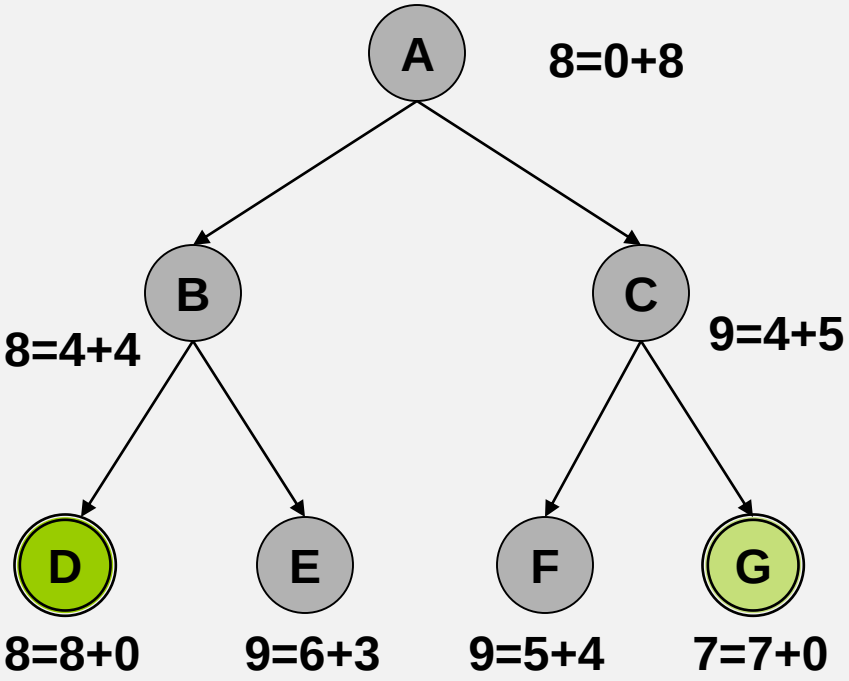
# Procura A*: exemplo




**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 óptima?

- Não! Aqui fica um contra exemplo...




- G é objectivo óptimo mas D é o objectivo encontrado...

TÉCNICO LISBOA

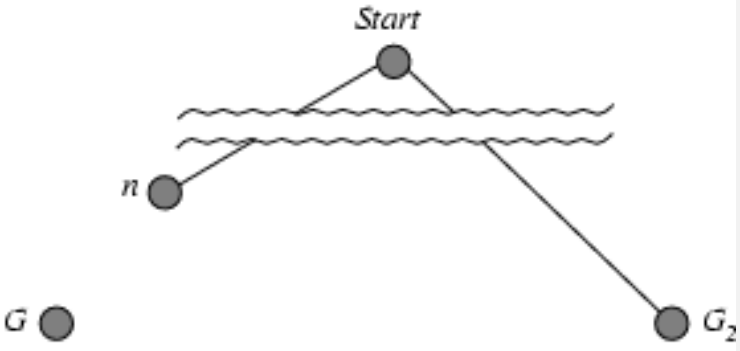
# Heurísticas admissíveis

- 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 objectivo.
- Uma heurística admissível nunca sobrestima o custo de atingir o objectivo, i.e. é optimista
- Exemplo: $h_{dlr}(n)$ (nunca sobrestima a distância real em estrada)

- Teorema: se *h(n)* é admissível, então a procura em árvore A* é óptima

# A* é Óptima (prova)

- Consideremos um nó objectivo não óptimo $G_2$ que já foi gerado mas não expandido (nó fronteira). Seja *n* um nó na fronteira tal que *n* está no menor caminho para um nó objectivo óptimo *G (com custo C*)*.




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

TÉCNICO LISBOA

# A* é Óptima

- Expande todos os nós com

$$f(n) < C^*$$

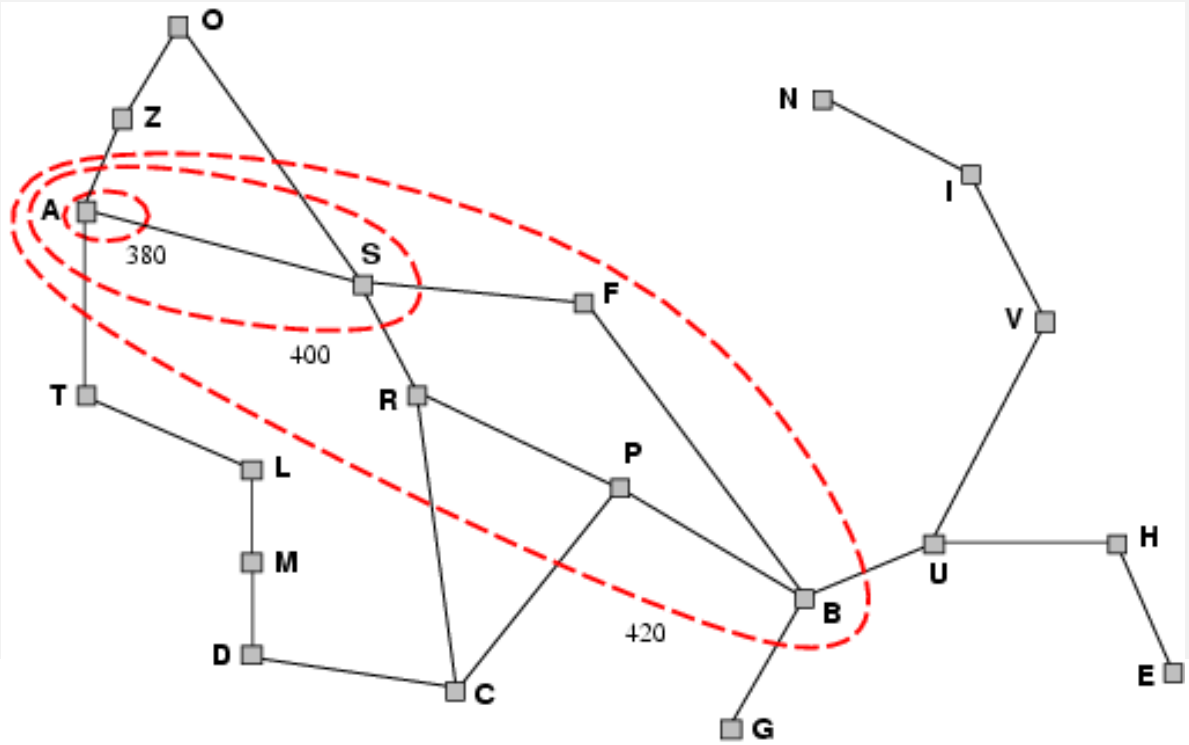
- Pode depois expandir alguns nós sobre a "curva de nível objectivo" com

$$f(n) = C^*$$

- Antes de seleccionar o estado objectivo

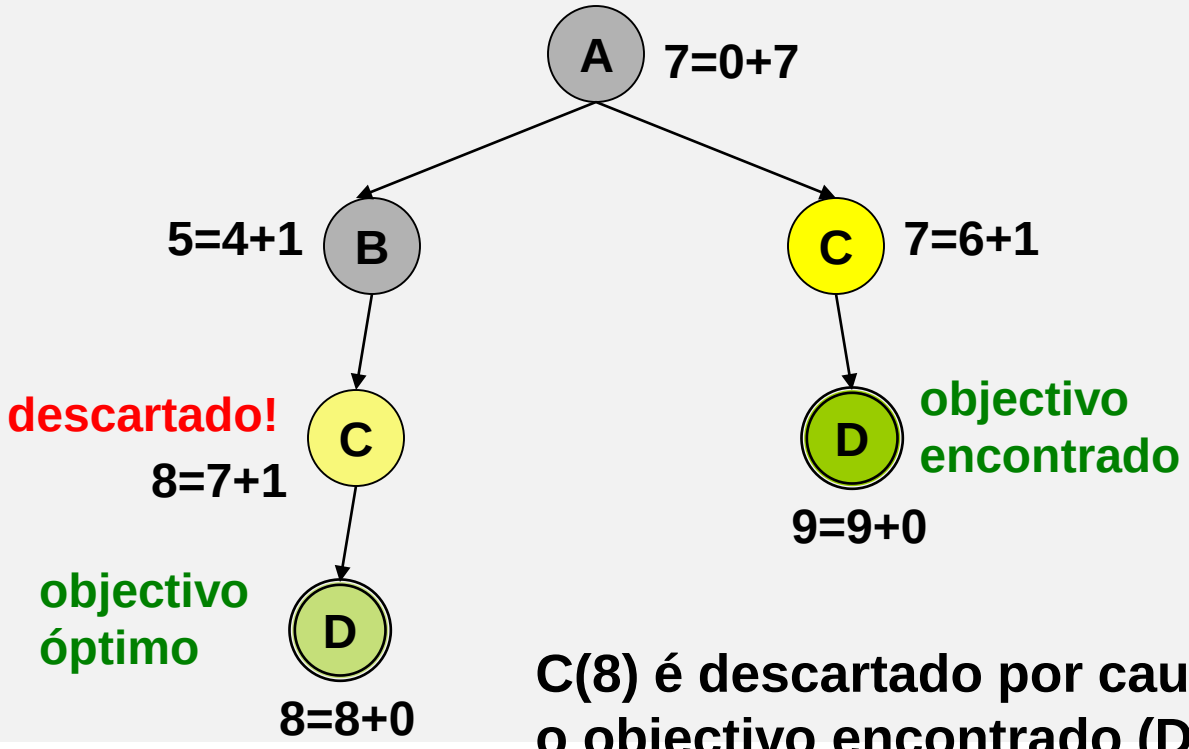
# A* é Óptima

- 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 óptima?

- Não, mesmo que a heurística seja admissível! Aqui fica um contra exemplo...




**C(8) é descartado por causa de C(7) e o objectivo encontrado (D(9)) não é óptimo...**

TÉCNICO LISBOA

# A* e Procura em Grafo

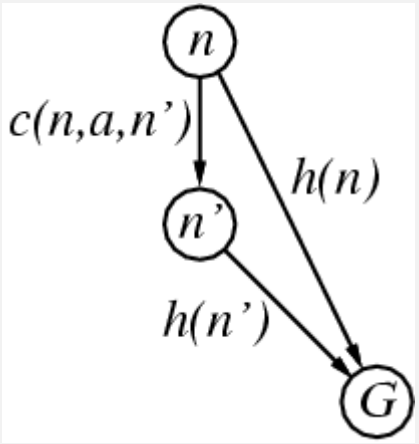
- A* com procura em grafo não é óptima com heurísticas admissíveis
- Pode ser descartado um nó que está no caminho que leva à solução óptima pelo facto de o mesmo nó já ter sido explorado no passado
- Pode passar a ser óptima se for mantido o registo dos caminhos e do valor de f(*n*) associados a todos 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 óptima

# Heurísticas Consistentes

- Uma heurística é consistente se para cada nó *n*, e para cada sucessor *n'* de *n* gerado por uma acção *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 objectivo óptimo então o caminho de menor custo é sempre seguido em primeiro lugar

- Se *h* é consistente, então temos para todo n' sucessor de n

$$f(n') = g(n') + h(n')$$
$$= g(n) + c(n,a,n') + h(n')$$

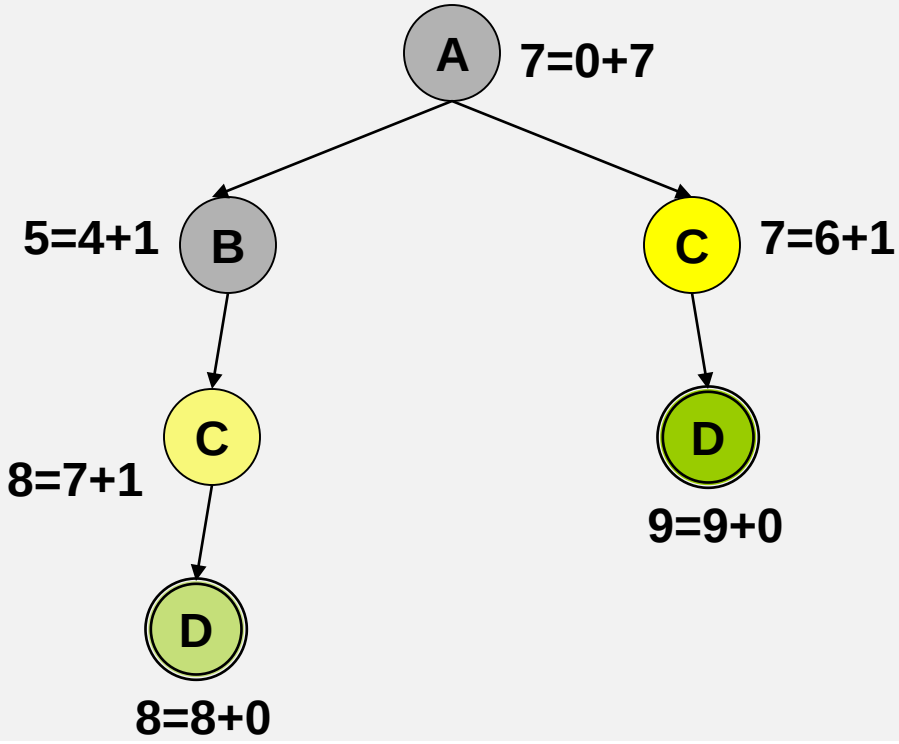
pela definição de consistência $\geq h(n)$

$$f(n') \geq g(n) + h(n)$$
$$f(n') \geq f(n)$$

- Logo, o valor de *f(n)* nunca decresce/diminui ao longo de um caminho

# Heurísticas Consistentes

- Conseguem ver se a heurística usada é consistente ou não?

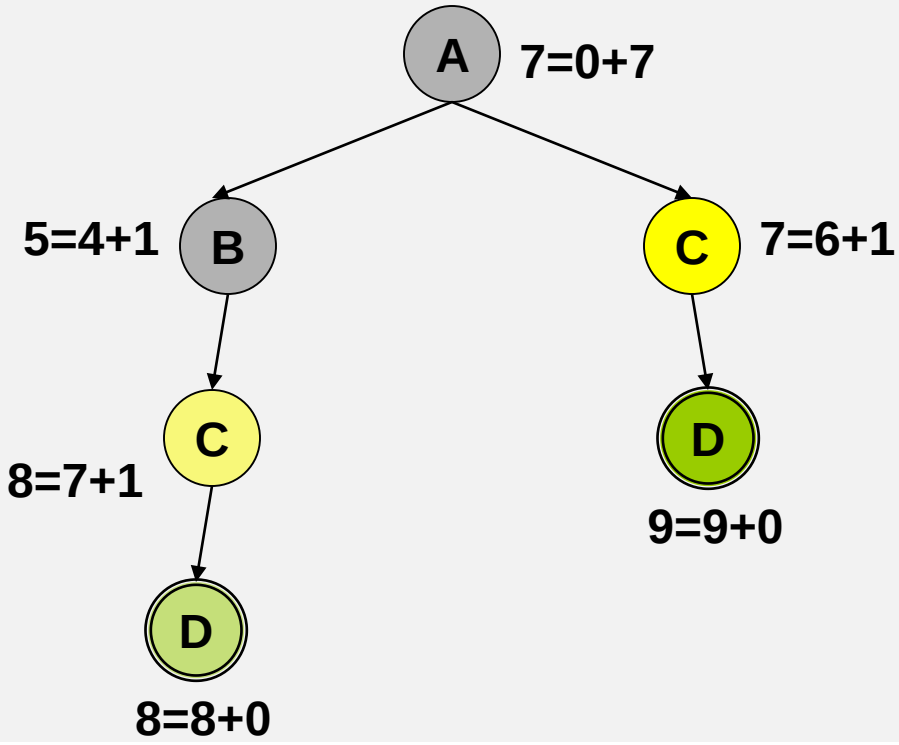


# Heurísticas Consistentes

- Prova formal de que não é

h(a) ≤ c(a,b) + h(b)

7 ≰ 4 + 1



# Heurísticas Consistentes

- h consistente → h admissível
  - Deixo a prova para TPC ☺
- h admissível ↛ h consistente

TÉCNICO LISBOA

# Propriedades de A*

- Completa? Sim (excepto se o número de nós com f ≤ *f(G)* for infinito)
- Tempo? Exponencial
- Espaço? Exponencial: mantém todos os nós em memória (no pior caso)
- Óptima? Sim

- A* Óptimalmente eficiente
  - Para qualquer função heurística, 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

TÉCNICO LISBOA

# Propriedades A*

- Complexidade exponencial no caso geral, mas muito dependente da qualidade da heurística
  - $\Delta \equiv h^* - h \rightarrow$ erro da heurística
- Se $h(n) = h^*(n)$, $\Delta = 0$
  - A* só expande nós no caminho da solução óptima
- Se $h(n) = 0$, $\Delta$ = máximo possível
  - Heurística é admissivel pela definição, mas não dá informação nenhuma
  - A* torna-se na procura de custo uniforme
  - A* vai expandir muitos nós

TÉCNICO LISBOA

# Propriedades A*

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

TÉCNICO LISBOA

# Propriedades A*

- Principal problema A*
  - Complexidade espacial exponencial
  - Precisamos de guardar todos os nós em memória
  - Ficamos sem espaço antes de ficar sem paciência
- Ideia: aplicar a procura em profundidade iterativa com o A*
  - Complexidade espacial – O(b.d)

TÉCNICO LISBOA

# 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 l
  - Inicializado com f(estado inicial)
- Em cada iteração é feita uma procura em profundidade primeiro com o seguinte corte:
  - Quando um nó *n* é gerado, se f(n) > limite o nó é cortado
- Em cada nova iteração o valor limite é actualizado com o menor valor de f(*n*) para os nós cortados na iteração anterior

TÉCNICO LISBOA

# IDA*

- Todos os anos digo a mesma coisa…
- Em cada iteração é feita uma procura em profundidade primeiro com o seguinte corte:
  - Quando um nó *n* é gerado, se f(n) > limite o nó é cortado
- E todos os anos vocês se enganam ☹

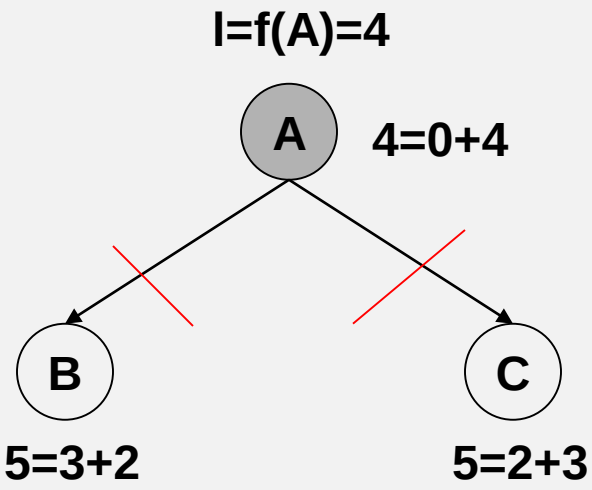
TÉCNICO LISBOA

# IDA*

- Em cada iteração é feita uma procura em profundidade primeiro
- Ou seja, …
- O valor de f de um nó **NÃO É** usado para escolher o próximo nó a expandir
  - Apenas para decidir se é cortado
- Critério de escolha é a profundidade

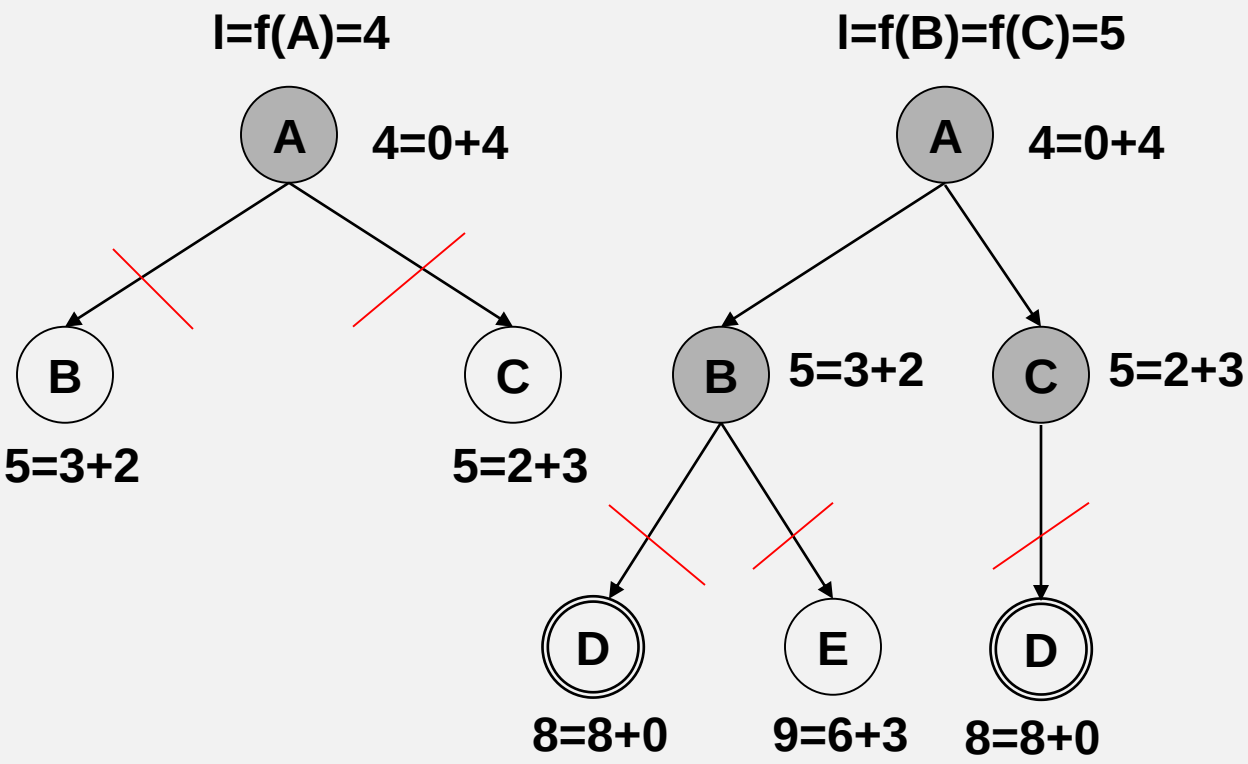
TÉCNICO LISBOA

# IDA*: exemplo



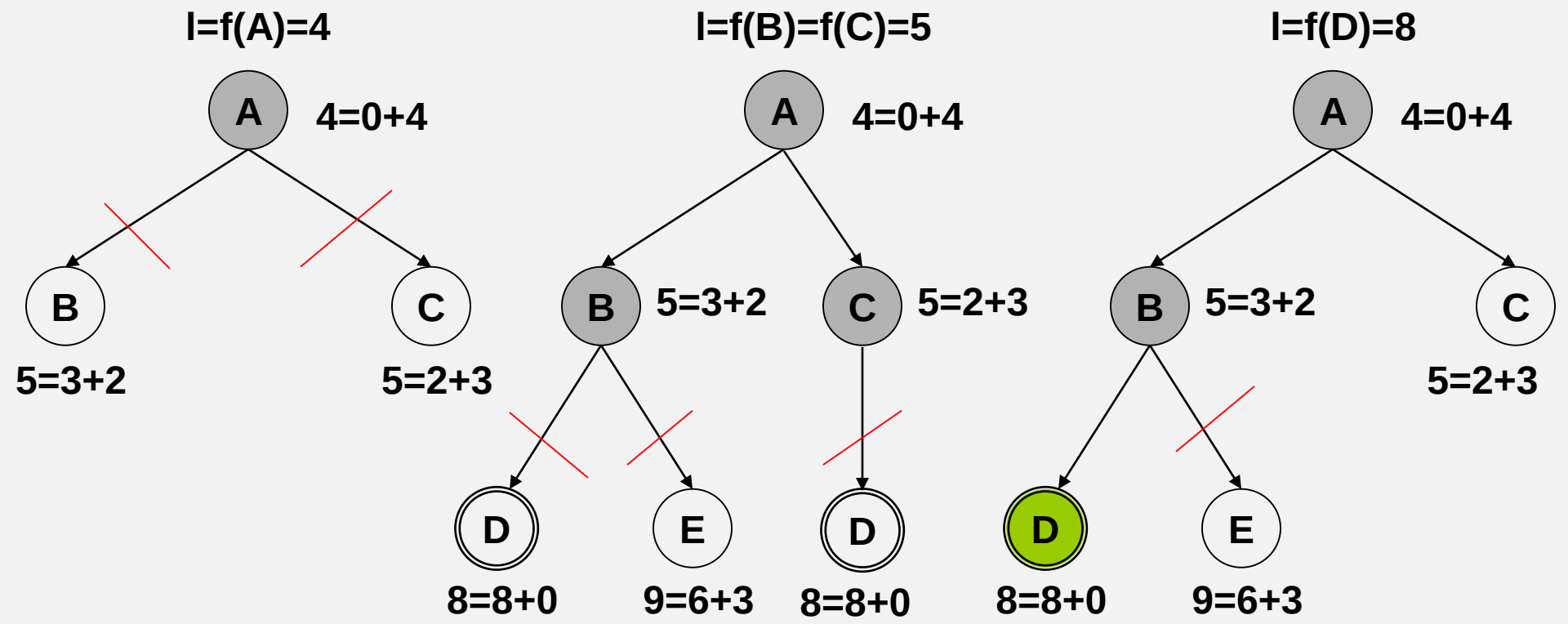
TÉCNICO LISBOA

# IDA*: exemplo



TÉCNICO LISBOA

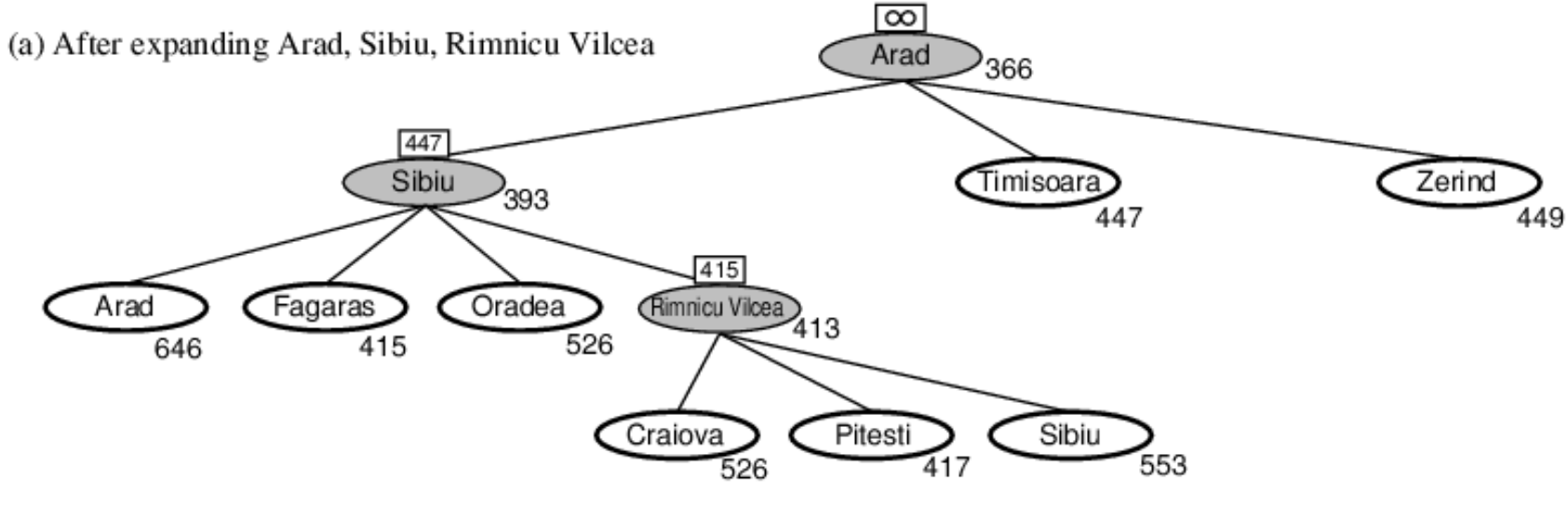
# IDA*: exemplo



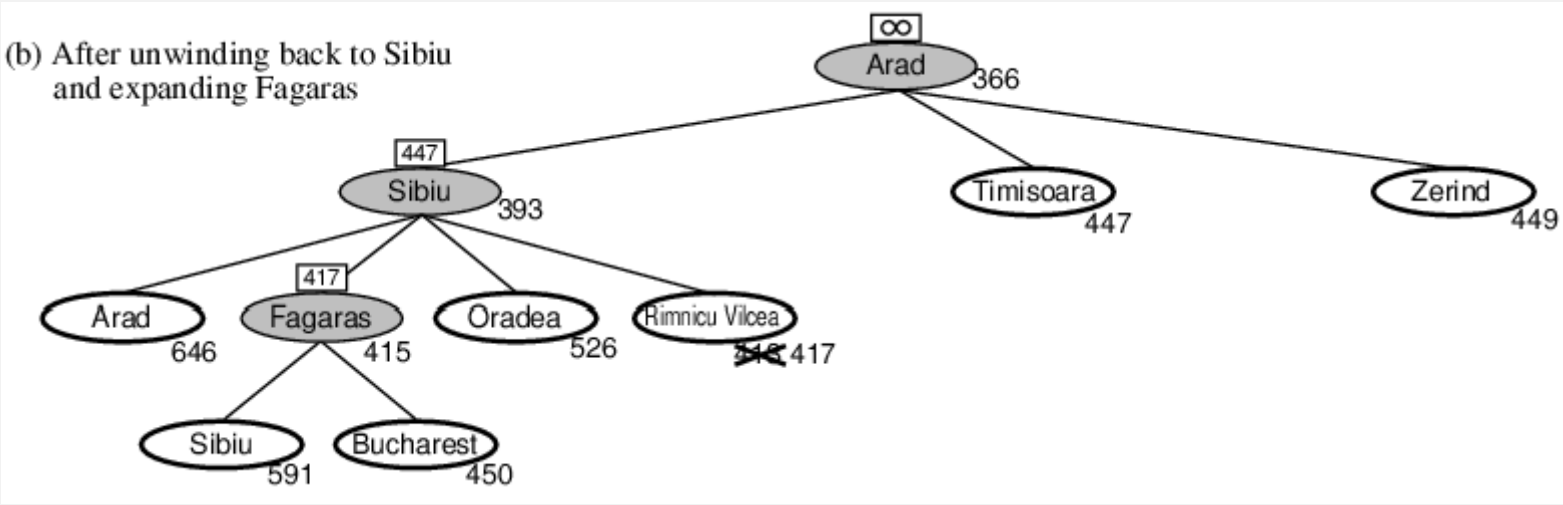
# Melhor Primeiro Recursiva (RBFS)

- Melhor primeiro (A*) com espaço linear (em *d*)
- Semelhante à procura em profundidade (implementação recursiva)
  - Para cada nó explorado, mantém registo do caminho alternativo com menor valor de f
  - Para escolher o próximo nó a expandir olha apenas para os filhos do nó actual
  - 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*
- Óptima se h(n) é admissível

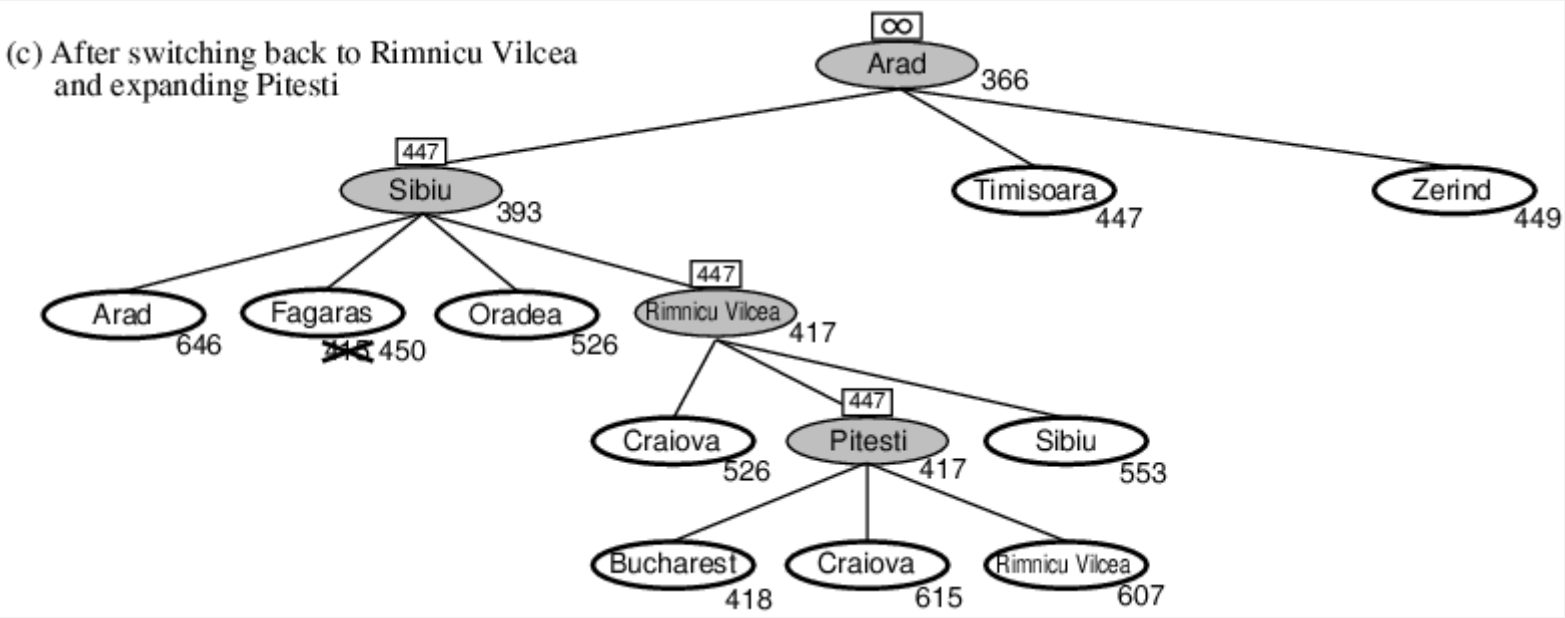
# Melhor Primeiro Recursiva: exemplo



# Melhor Primeiro Recursiva: exemplo



# Melhor Primeiro Recursiva: exemplo



# Melhor Primeiro Recursiva (RBFS)

```
function RECURSIVE-BEST-FIRST-SEARCH (problem) returns a solution, or failure
    return RBFS(problem,Make-Node(problem.Initial-State),∞)

function RBFS(problem,node,f_limit) returns a solution, or failure and a new f-cost limit

    if problem.Goal-Test(node.State) then return Solution(node)
    successors ← []
    for each action in problem.Actions(node.State) do
        add Child-Node(problem,node,action) into successors
    if successors is empty then return failure, ∞
    for each s in successors do /*update f with value from previous search, if any*/
        s.f ← max(s.g + s.h, node.f)
    loop do
        best ← the lowest f-value node in successors
        if best.f > f_limit then return failure, best.f
        alternative ← the second-lowest f-value among successors
        result,best.f ← RBFS(problem,best,min(f_limit,alternative))
        if result ≠ failure then return result
```

TÉCNICO LISBOA

# Propriedades RBFS

- Ligeiramente mais eficiente que o IDA*
- Mas 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 o melhor caminho e extendê-lo com mais um nó

TÉCNICO LISBOA

# Propriedades RBFS

- Completude
  - Completa, se b finito e custo dos passos > $\varepsilon$
- Optimalidade
  - Óptima, se heurística *h(n)* for admissível
- Complexidade espacial

  $\mathcal{O}(bd)$ - *linear*
- Complexidade temporal
  - Depende de
    - Precisão da heurística
    - Quantas vezes muda o melhor caminho à medida que os nós vão sendo expandidos

TÉCNICO LISBOA

# IDA* e RBFS

- Podem ver a complexidade temporal exponencialmente agravada ao procurar em grafos
  - Não podem detectar nós repetidos, a não ser no caminho nactual
  - Podem re-expandir o mesmo estado muitas vezes
- Ambos sofrem por usar demasiado pouca memória

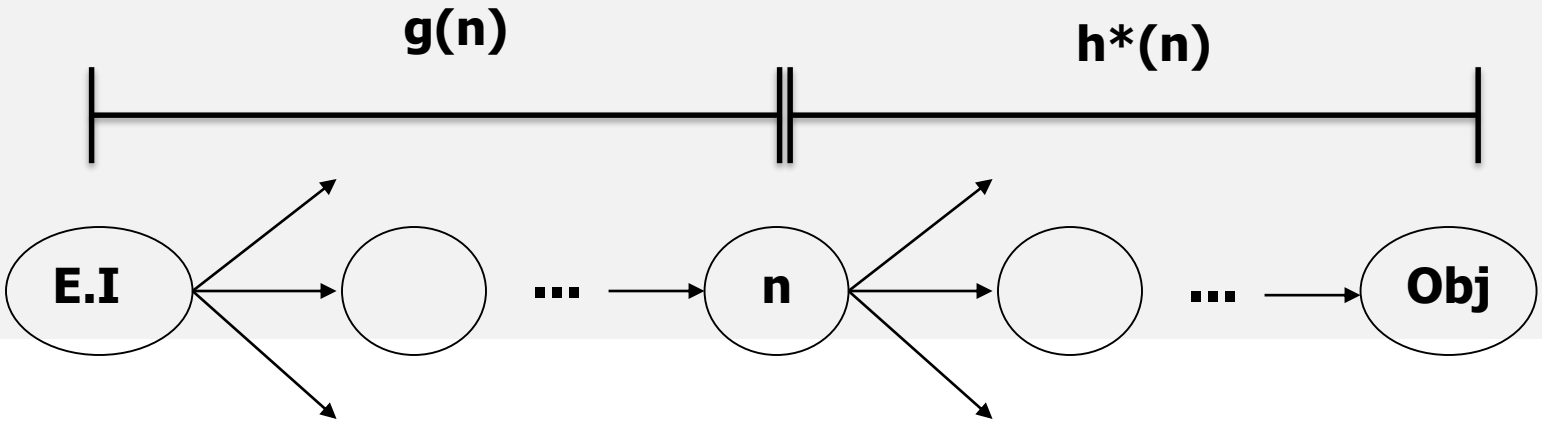
TÉCNICO LISBOA

# 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 da nó esquecido no pai
- O nó esquecido pode ser regenerado se todos os caminhos forem piores do que o seu valor

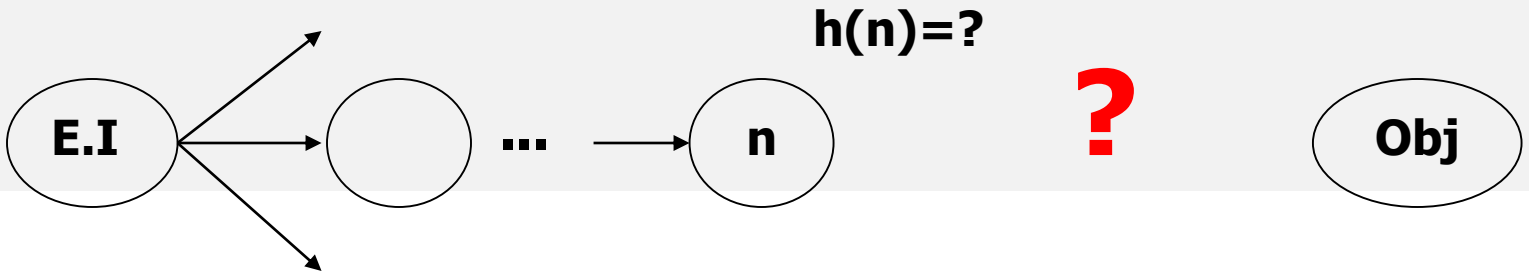
# Funções Heurísticas

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



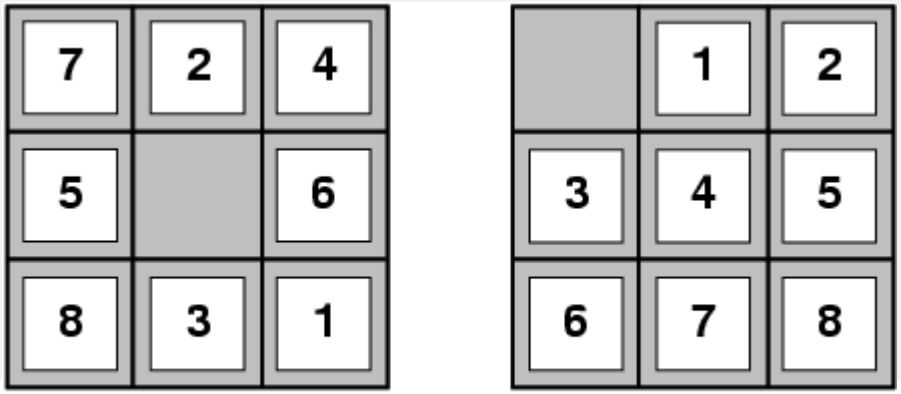
# Funções Heurísticas

- 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



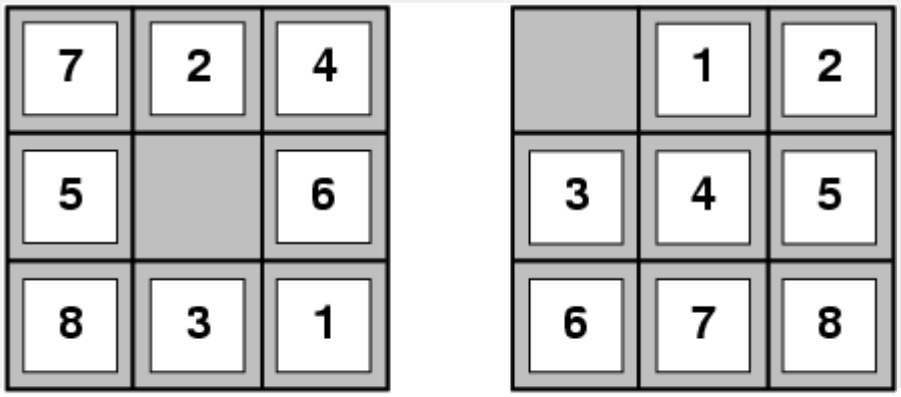
# Funções Heurísticas

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



# 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

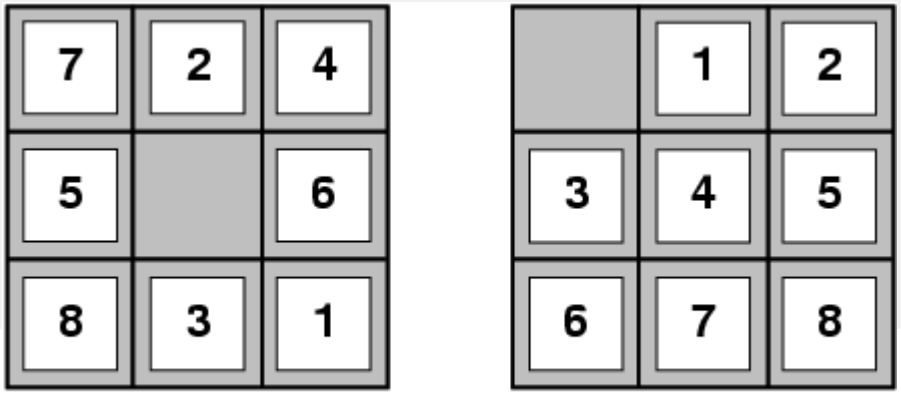



Start State

Goal State

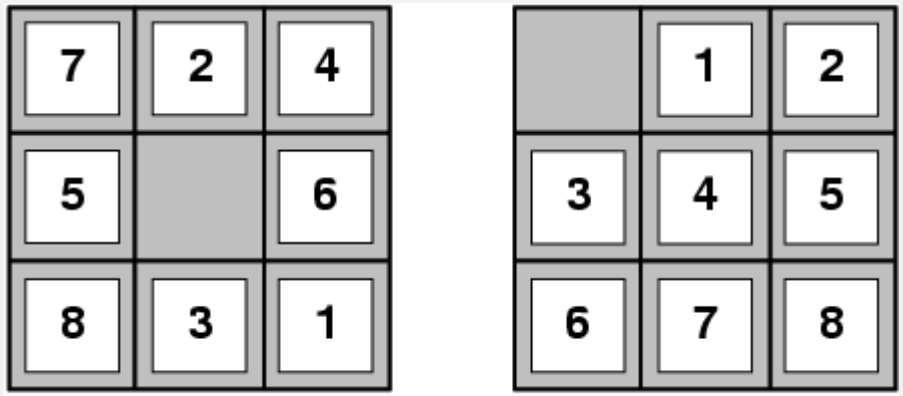
# Funções heurísticas para 8-puzzle

- Já agora, são necessários 26 passos para resolver este 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
    - Necessita de apenas um passo para a colocar na posição final (muito optimista!)
  - Heurística admissível
    - Qualquer peça fora de posição precisa de ser movida pelo menos uma vez para chegar à posição final
- $h_1$(StartState) = ? 8
  - todas as peças estão fora de posição



# 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.
- $h_2$(StartState) = ? 3+1+2+2+2+3+3+2 = 18




Start State  Goal State

# Comparar funções heurísticas

- Qualidade de uma heurística
  - b* - factor de ramificação efectivo
    - Sendo N, o número de nós gerados por uma procura A*
    - d – a profundidade da solução
    - b* - factor de ramificação que uma árvore uniforme de profundidade d teria de ter para conter N +1 nós
      - $N+1=1+b^*+(b^*)^2 + \ldots + (b^*)^d$
  - Número de nós gerados variam muito com a profundidade d
  - b* varia pouco com a profundidade
    - Bom para comparar procuras e heurísticas

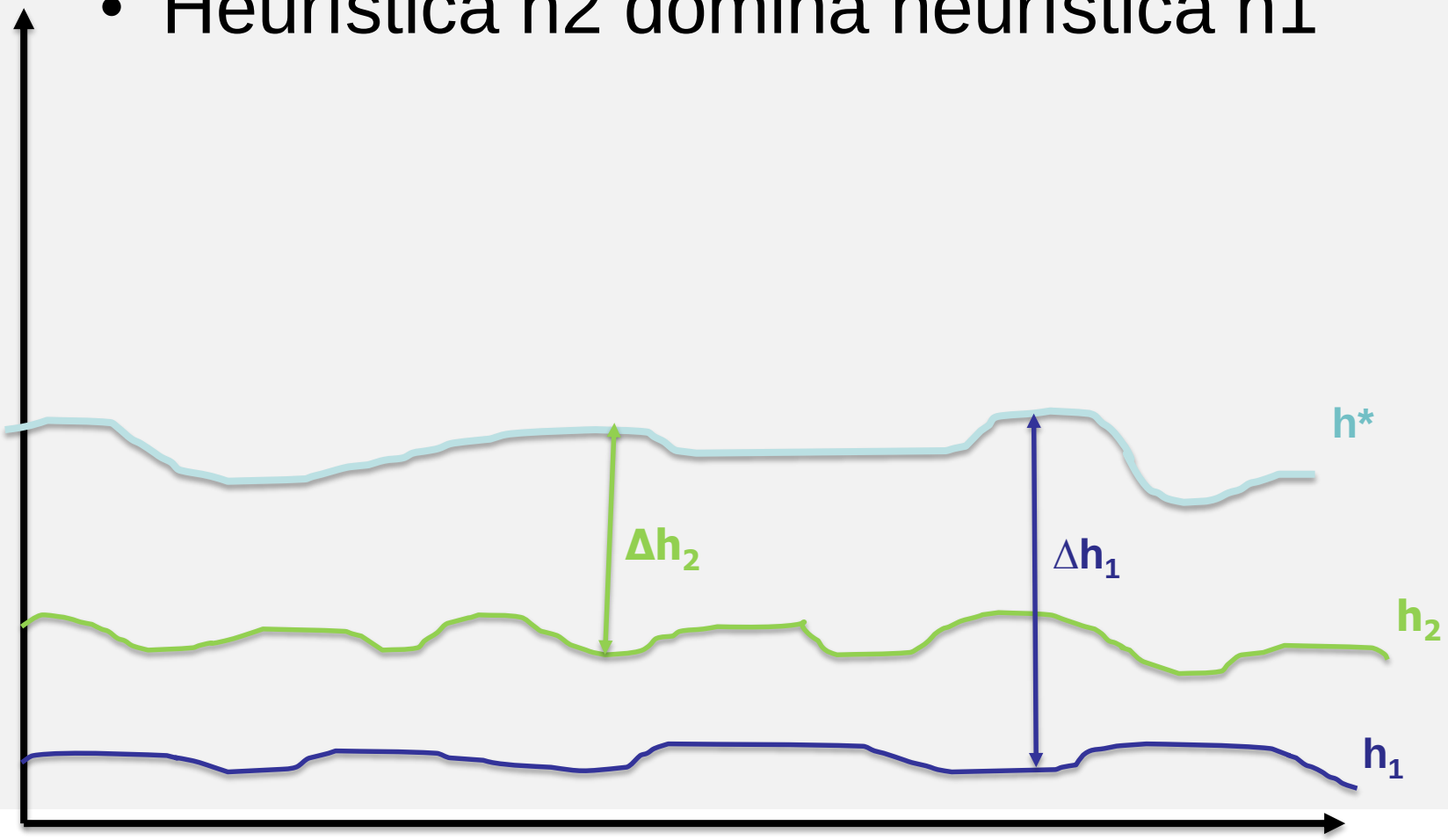
TÉCNICO LISBOA

# Comparar funções heurísticas

| | Search Cost (nodes generated) | | | Effective Branching Factor b* | | |
|---|---|---|---|---|---|---|
| ***d*** | **IDS** | **A*(h1)** | **A*(h2)** | **IDS** | **A*(h1)** | **A*(h2)** |
| 2 | 10 | 6 | 6 | 2.45 | 1.79 | 1.79 |
| 4 | 112 | 13 | 12 | 2.87 | 1.48 | 1.45 |
| 6 | 680 | 20 | 18 | 2.73 | 1.34 | 1.30 |
| 8 | 6384 | 39 | 25 | 2.80 | 1.33 | 1.24 |
| 10 | 47127 | 93 | 39 | 2.79 | 1.38 | 1.22 |
| 12 | 3644035 | 227 | 73 | 2.78 | 1.42 | 1.24 |
| 14 | - | 539 | 113 | | 1.44 | 1.23 |
| 16 | - | 1301 | 211 | | 1.45 | 1.25 |
| 18 | - | 3056 | 363 | | 1.46 | 1.26 |
| 20 | - | 7276 | 676 | | 1.47 | 1.27 |
| 22 | - | 18094 | 1219 | | 1.48 | 1.28 |

# Comparar funções heurísticas

- Heurística h2 domina heurística h1



TÉCNICO LISBOA

# 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)
    - Excepto eventualmente para alguns nós com f(n) = C*

- $h_2$ é melhor para a procura
  - Expande menos nós porque não é tão optimista / tem um erro menor
  - Custo da procura (média do nº de nós expandidos):

*prof=12* Profundidade Iterativa = 3,644,035 nós
$A^*(h_1)$ = 227 nós
$A^*(h_2)$ = 73 nós

*prof=24* Profundidade Iterativa = muitos nós...
$A^*(h_1)$ = 39,135 nós
$A^*(h_2)$ = 1,641 nós

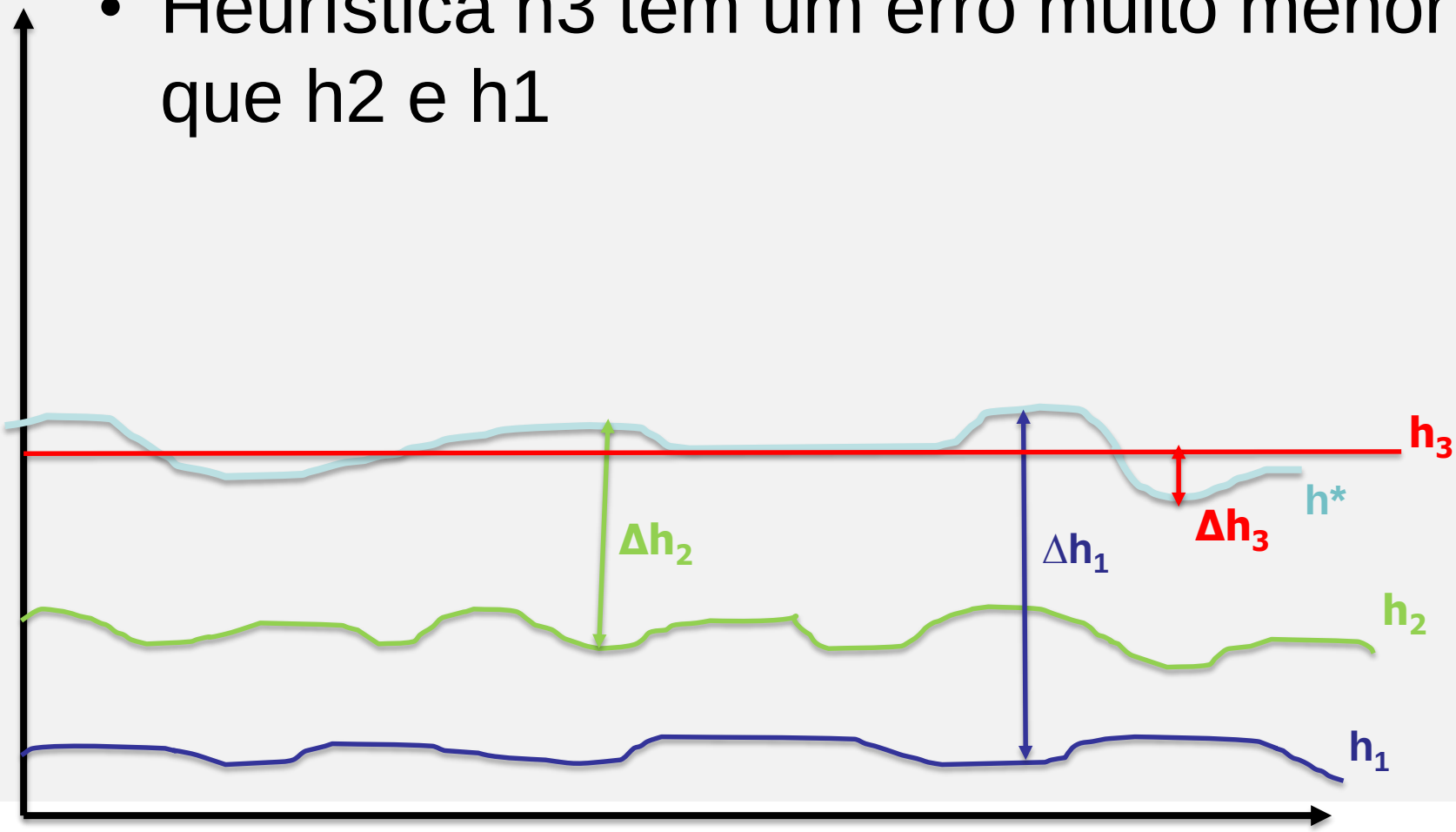
TÉCNICO LISBOA

# Heurísticas não admissíveis

- Por vezes, se a solução óptima 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 óptima

# Heurísticas não admissíveis

- Heurística h3 tem um erro muito menor que h2 e h1



TÉCNICO LISBOA

# Heurísticas Admissíveis: como inventá-las?

- Heurísticas h1 e h2 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 acções 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 h1 retorna o custo exacto da melhor solução para o problema: retirar as 8 peças do puzzle e voltar a coloca-las.
- Um problema com menos restrições nas acções consideradas é chamado problema relaxado

TÉCNICO LISBOA

# Heurísticas Admissíveis: como inventá-las?

- O custo de uma solução óptima para um problema relaxado corresponde a uma heurística admissível e consistente para o problema original
- Exemplo do 8-puzzle:
  - Restrições sobre as acções:
  - Uma peça pode mover-se de uma posição A para B se:
    - A é verticalmente ou horizontalmente adjacente a B (1) **e se**
    - B está vazio (2)
  - 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 → ??

TÉCNICO LISBOA

# Heurísticas Admissíveis: como inventá-las?

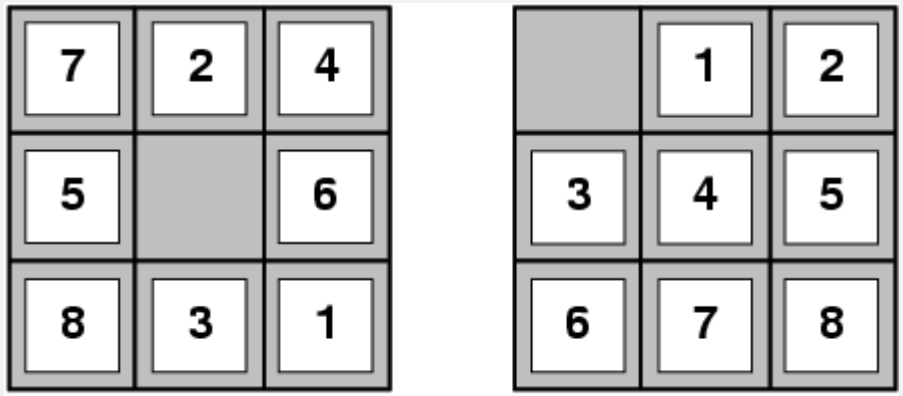
- Problema relaxado:
  - Uma peça pode mover-se de A para B se B está vazio
  - Ideia para nova heurística h3

- Ideia para heurística problema relaxado
  - Para cada peça
    - Se peça está na posição final
      - Custo 0
    - Se posição final da peça está vazia actualmente
      - 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?

- h3(start) = 2 + 2 + 1 + 2 + 2 + 2 + 2 + 2 = 15

custo 7 custo 2 custo 4 custo 5 custo 6 custo 8 custo 3 custo 1



TÉCNICO LISBOA

# 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-de-procura (tempo que leva a calcular )

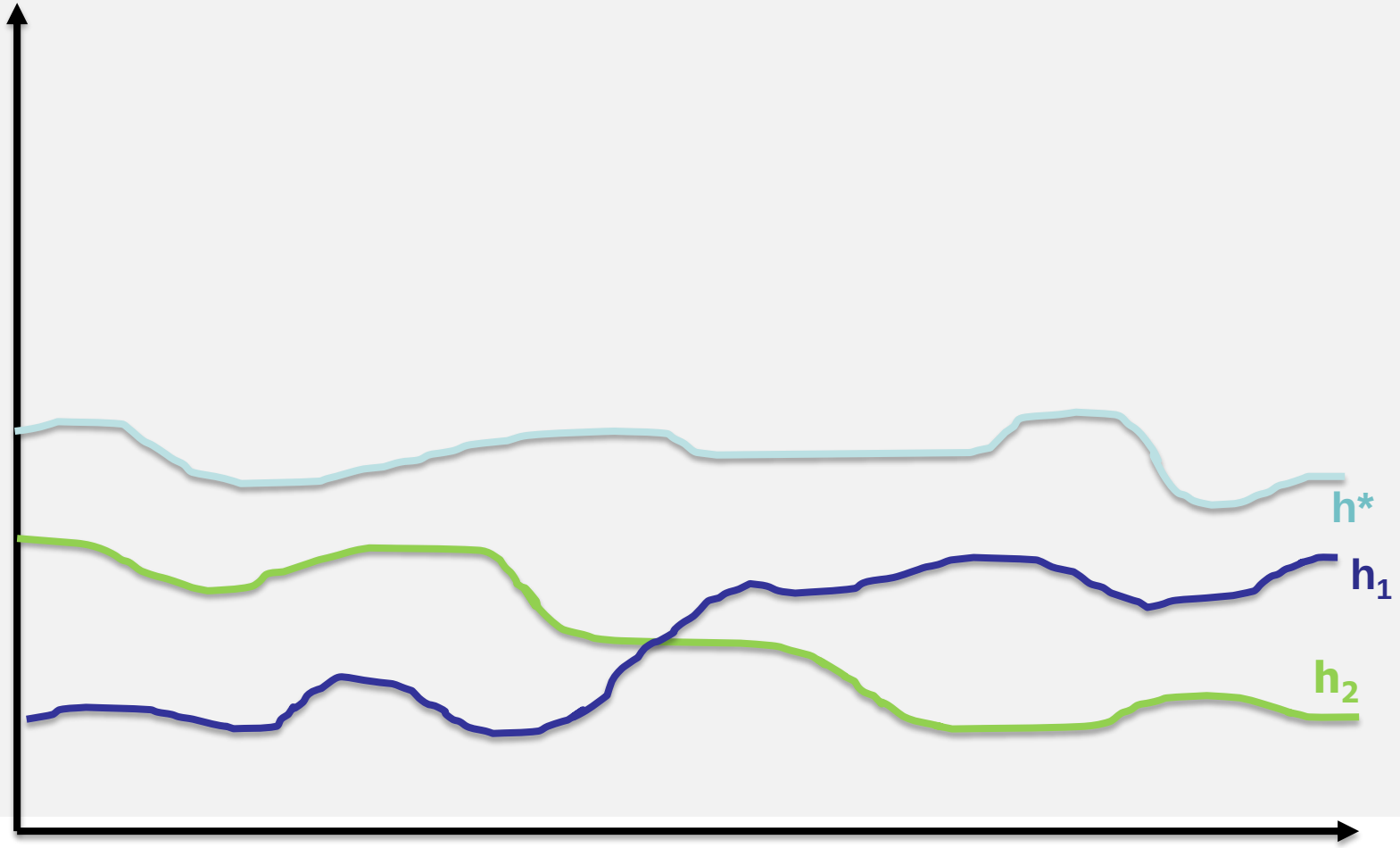
TÉCNICO LISBOA

# Heurísticas Admissíveis: como inventá-las?

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

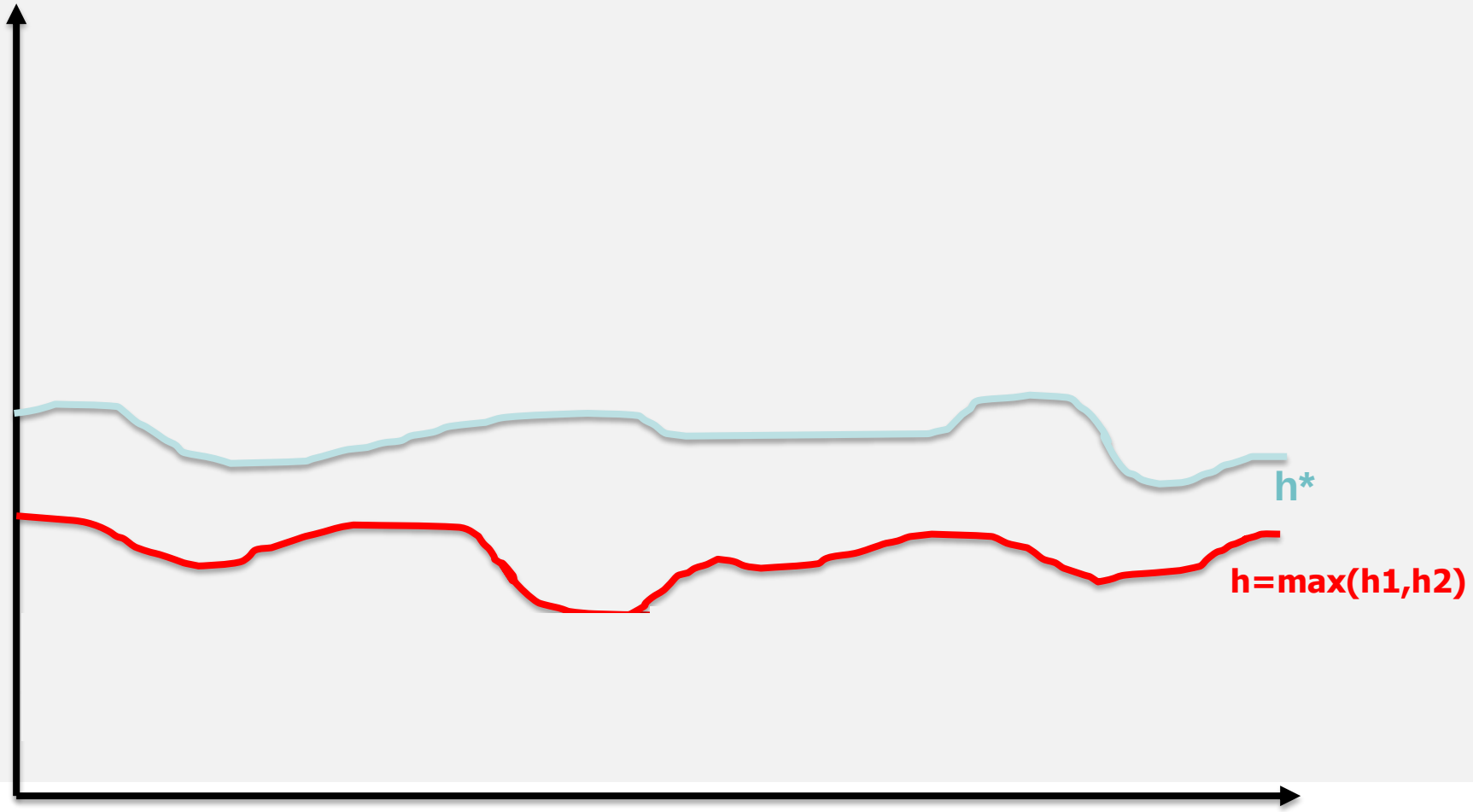
TÉCNICO LISBOA

# Heurísticas Admissíveis:
## como inventá-las?



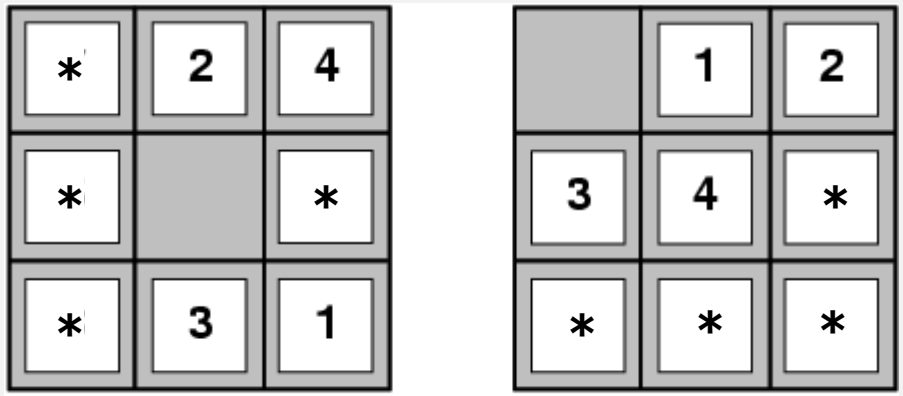
TÉCNICO LISBOA

# Heurísticas Admissíveis:
## como inventá-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




Start State Goal State

TÉCNICO LISBOA

# Heurísticas Admissíveis: como inventá-las?

- Bases de dados de padrões
  - Guardar custos exactos 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 actual do estado na base de dados
  - Base de dados construída ao fazer uma procura regressiva a partir do objectivo
    - e guardando na b.d. o custo de cada estado gerado
    - Custo de geração da b.d. amortizado ao longo de muitas procuras

TÉCNICO LISBOA

# Heurísticas Admissíveis: como inventá-las?

- Bases de dados de padrões
  - Podemos construir b.d para outros subproblemas
    - 5,6,7,8
    - 2,3,4,5
  - Podemos combinar as heurísticas de cada subproblema fazendo o máximo valor delas
  - E não podemos somar? (1,2,3,4) + (5,6,7,8)
    - Se os subproblemas não forem independentes não, pq perdemos admissibilidade
      - Movimentos partilhados entre os 2 subproblemas
    - 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

TÉCNICO LISBOA

# Heurísticas Admissíveis: como inventá-las?

- Aprender através da experiência
  - Resolver muitas instâncias diferentes do 8-puzzle
  - Cada solução óptima contem vários exemplos que podem ser usados para aprender h(n)
  - Um exemplo consiste num estado do caminho da solução e respectivo custo da solução a partir daquele ponto.
  - Aprendizagem usando características “features” do estado
    - $x_1(n)$ – numero de peças fora da posição desejada
    - $X_2(n)$ – numero de pares de peças adjacentes que não são adjacentes no estado final

# Heurísticas Admissíveis:
## como inventá-las?

- Heurística – combinação linear das várias features
  - $h(n) = c_1x_1(n) + c_2x_2(n)$
- Coeficientes c1 e c2 aprendidos e ajustados de modo a melhor se adaptarem aos exemplos usados
- Não se garante admissibilidade.