Busca com informação e exploração

Aula 3 - Cap. 4

Fundamentos da IA

Mestrado - FEI - 2006

Busca informada (ou heurística) -- busca pela melhor escolha (best-first search)

- Utiliza conhecimento sobre o domínio para encontrar soluções mais eficientes do que no caso de busca cega.
- Busca pela melhor escolha: expande o nó que possui ***função de avaliação*** mais baixa;

Heureca! *(pretérito de Heurisko)*

- Função de avaliação (f(n)): mede o custo de um nó até o objetivo.
- Função heurística (*h(n)*): custo estimado do caminho mais econômico do nó n até o nó objetivo.

Heurística = capacidade de resolver problemas

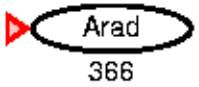
Busca gulosa (pela melhor escolha)

- Expande o nó mais próximo da meta, supondo que isso levará rapidamente a uma solução;
- Portanto, f(n) = h(n)
- Exemplo: encontrar uma rota na Romênia usando da ***heurística da distância em linha reta*** (h_{DLR})

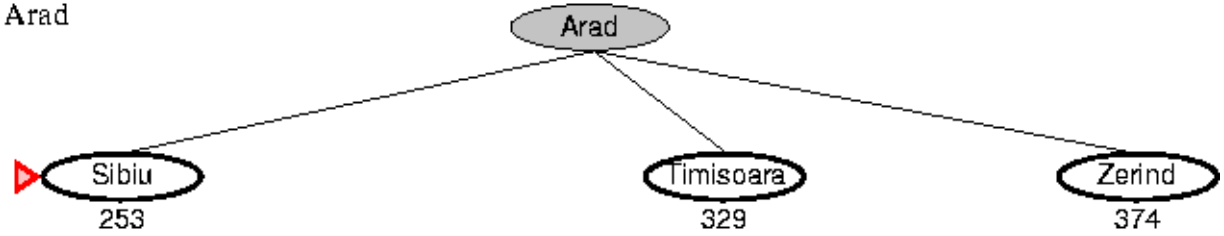
Busca gulosa: exemplo

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	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

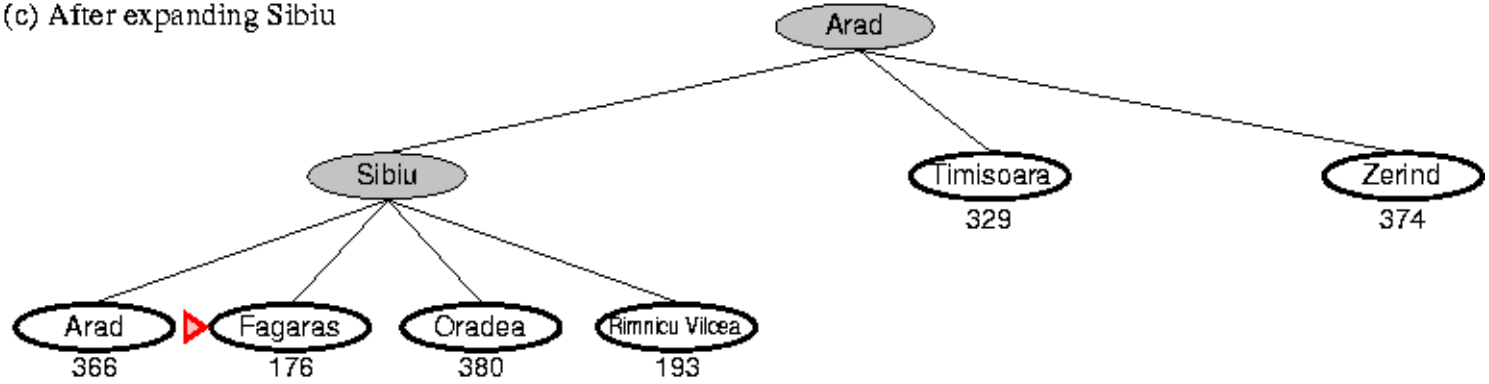
(a) The initial state



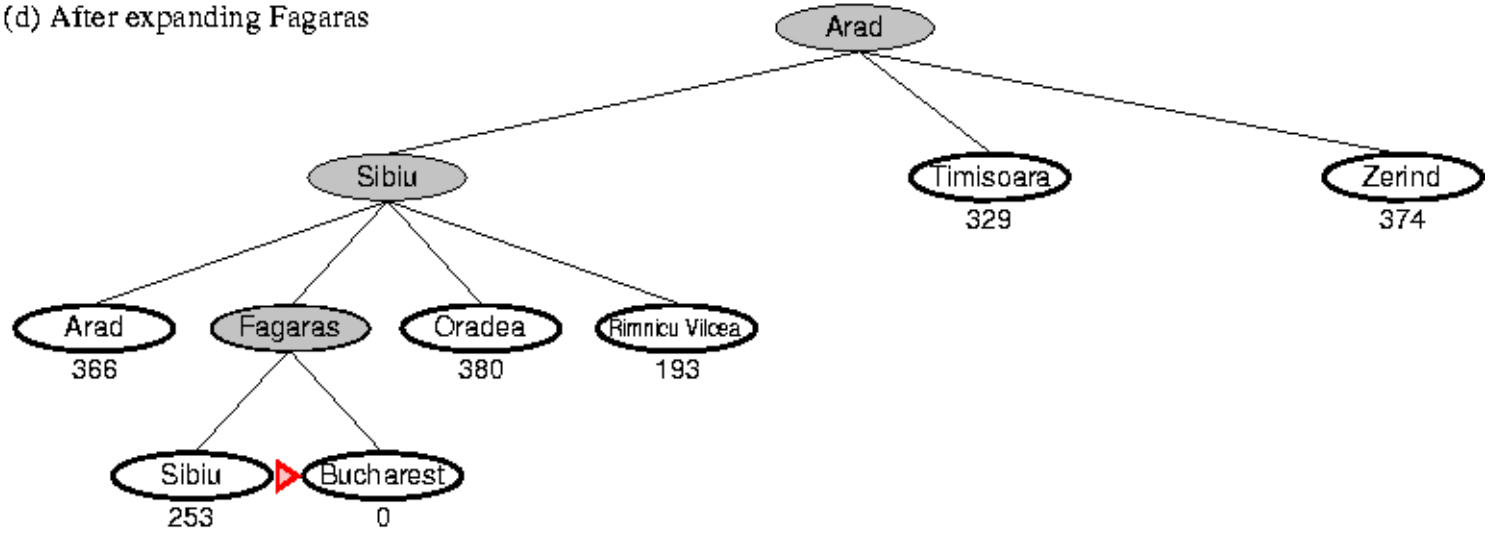
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



Busca gulosa

- Não é ótima, pois segue o melhor passo **considerando somente o momento atual**.
  - pode haver um caminho melhor seguindo algumas opções piores em alguns pontos da árvore de busca.
- Minimizar h(n) é suscetível a falsos inícios.

Busca gulosa

- Guloso: em cada passo, tenta chegar mais perto do objetivo.
- Não é completa: pode seguir caminhos infinitos e nunca voltar (como depth-first search!).

Busca A*

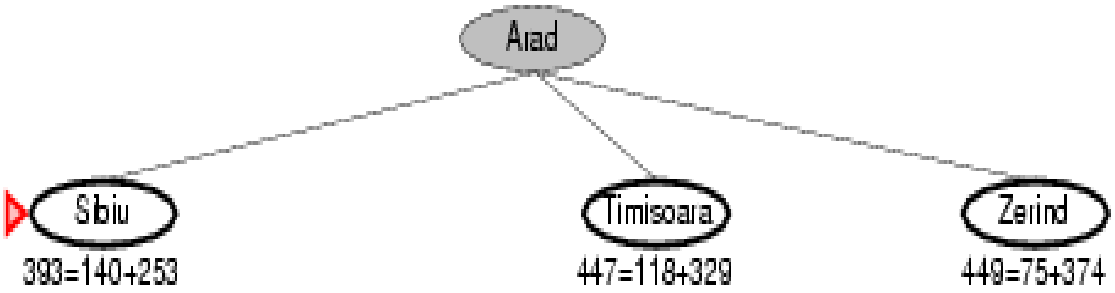
- Avalia nós combinando o custo para alcançar cada nó (*g(n)*) e o custo estimado para ir deste nó até o objetivo (*h(n)*):

$$f(n) = g(n) + h(n)$$

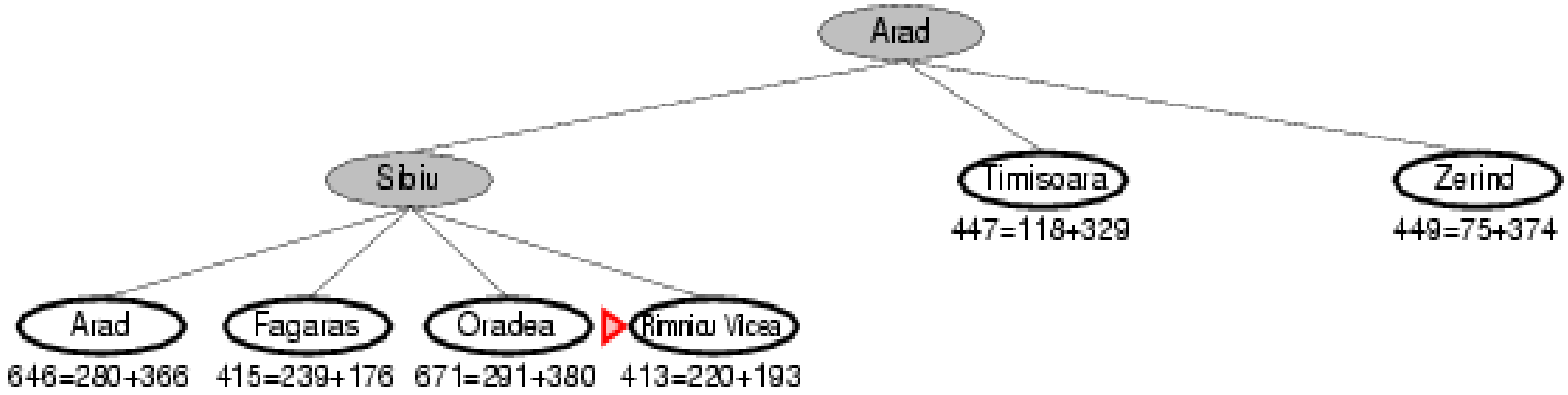
- Para a solução de custo mais baixo, seguir os estados de menor valor de g(n) + h(n).

Busca A*: exemplo

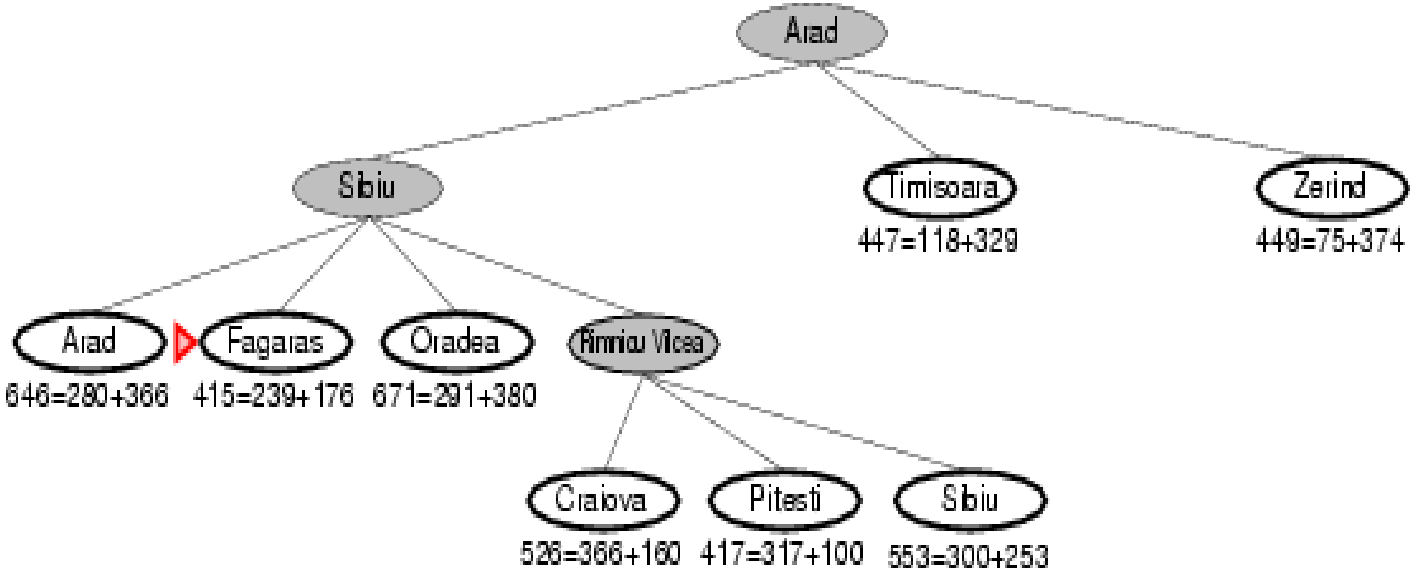
Busca A*: exemplo



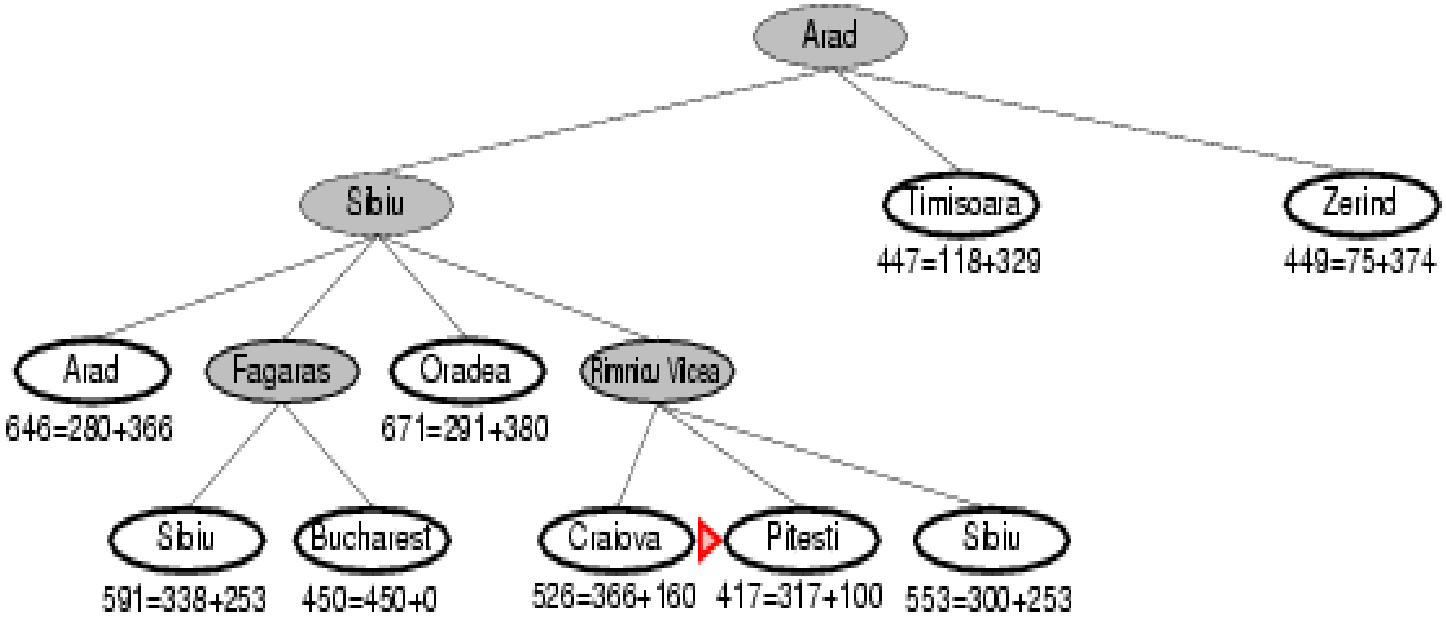
Busca A*: exemplo



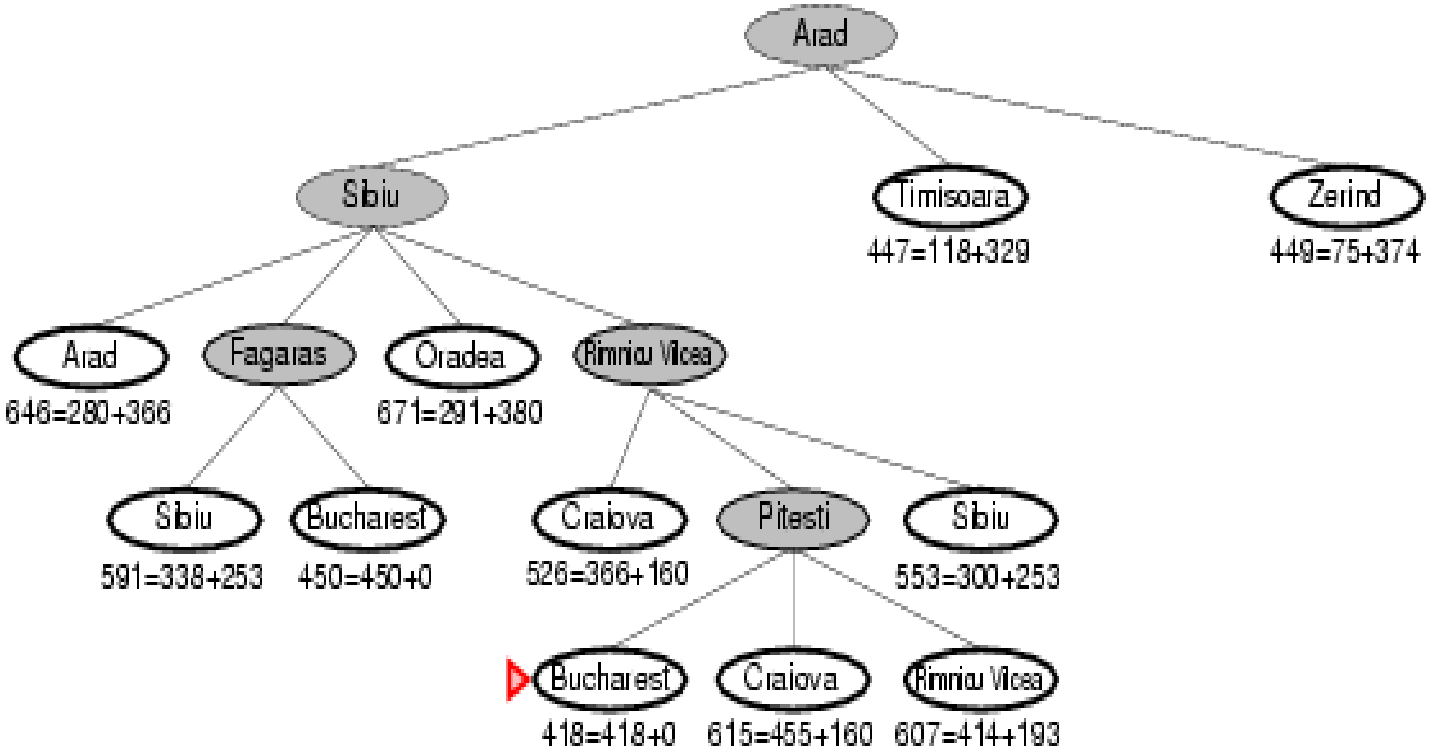
Busca A*: exemplo



Busca A*: exemplo



Busca A*: exemplo



- Perceberam que A* tomou um rumo diverso do algoritmo guloso??
- A* tomou o caminho ótimo..

Busca A*: ótima e completa??

- SIM, se a ***heurística for admissível*** - i.e. desde que a função h(n) nunca *superestime* o custo para alcançar um objetivo;
  - ex. a distância em linha reta.
- assim, f(n) nunca irá superestimar o custo verdadeiro de uma solução, pois g(n) é o valor exato.

Prova-usando busca em árvore...

- Assuma um **nó objetivo não-ótimo** G2, e seja C* o custo da solução ótima. Então, como G2 não é ótimo e h(G2) = 0, sabemos que:

  f(G2) = g(G2) + h(G2) = g(G2) > C*

- Considere qqr nó de borda **n** que esteja num caminho de solução ótimo. Se h(n) ***não superestimar o custo de completar o caminho de solução***, então: **f(n) = g(n) + h(n) ≤ C*.**

- Logo, se $f(n) \leq C^* < f(G2)$, G2 não será expandido e A* deve retornar uma solução ótima.
- Isso vale para busca em árvore, para outras estruturas de busca pode não valer (ex. busca com checagem de repetição -- busca em grafo -- pois um caminho ótimo pode ser deletado, se não for expandido primeiro!)

- Uma solução: assegurar que o caminho ótimo para qualquer estado repetido é sempre o primeiro a ser seguido -- como na busca com custo uniforme!
- Isso ocorre se h(n) for **consistente (ou monotônico).**

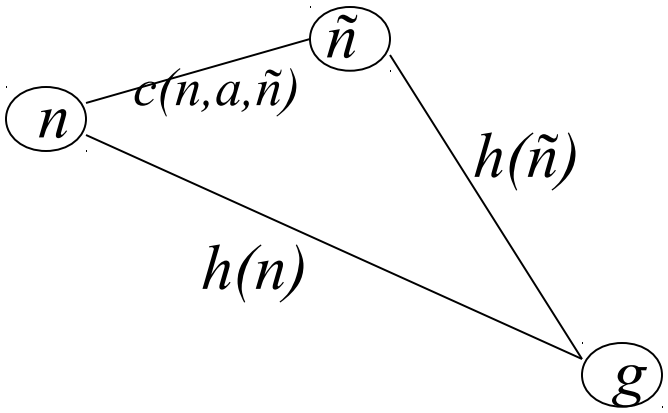
Heurística consistente:

- **H(n) é consistente se o custo estimado para alcançar o objetivo a partir de n for menor do que o custo do passo para se chegar a ñ (ñ∈ {sucessor(n)}) somado a h(ñ).**

Heurística consistente:

- Para todo nó *n* e todo sucessor *ñ* de *n* *(dada uma ação a)*:

$$h(n) \leq c(n, a, \tilde{n}) + h(\tilde{n})$$



- Toda heurística admissível é também consistente (ex. 4.7)
- Se uma h(n) é consistente então os valores de f(n) ao longo de qualquer caminho são não-decrescentes.

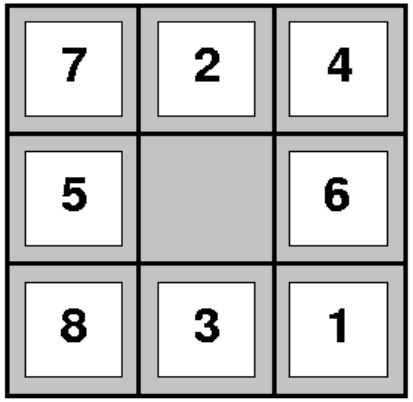
A* para heurísticas consistentes

- Completo
- Otimamente eficiente:
  - i.e. nenhum outro algoritmo tem garantia de expandir um número de nós menor que A*. Isso porque qqr algoritmo que não expande todos os nós com f(n) < C* corre o risco de omitir uma solução ótima.

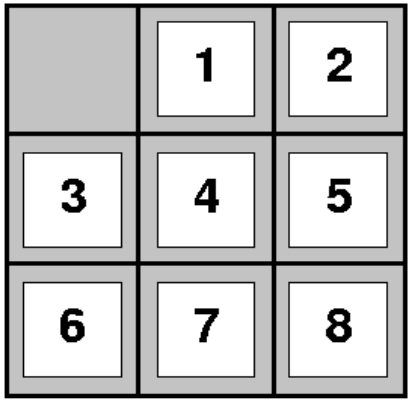
A* nem tudo o que queremos...

- Na maioria dos problemas o número de nós gerados ainda é exponencial em relação ao comprimento da solução.
- Todos os nós gerados devem ser mantidos na memória... A* esgota a memória rapidamente.

Funções heurísticas estudo de caso: *quebra-cabeças de 8 peças*



Start State



Goal State

h1 = 8

h2 = 18

- h1 = o número de blocos em posições erradas. (adimissível, pois cada bloco deve ser movido ao menos uma vez)
- h2 = a soma das distâncias dos blocos de suas posições objetivos. (adim., o resultado de qqr movimento é deslocar o bloco para uma posição + prox. do objetivo) -- distância de manhattan

Qualidade de uma heurística: fator de ramificação efetiva

- para um problema em que A* gera *N* nós e cuja profundidade de solução seja ***d***, *o* fator de ramificação efetiva **b*** é o fator de ramificação que uma ***árvore uniforme*** de profundidade ***d*** precisa ter para conter **N+1** nós. Assim:

$$N+1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

Extra!! Extra!!

	Custo da busca			Fator de ramificação efetiva		
d	BAI	$A^*(h_1)$	$A^*(h_2)$	BAI	$A^*(h_1)$	$A^*(h_2)$
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
24	–	39135	1641	–	1,48	1,26

Heurística dominante

- *h2* é melhor que *h1* e muito melhor que a busca por aprofundamento iterativo.
- *h2* é sempre melhor que *h1*, pois

$$\forall n \; h2(n) \geq h1(n)$$

- I.e. h2 **domina** h1

Dominância é eficiência

- A* com h2 nunca expandirá mais nós que A* com h1, pois o fator de ramificação de h2 será mais próximo de 1 do que o de h1
  - menor nós serão expandidos por h2!
- Logo: é sempre melhor usar uma heurística consistente com valores maiores que outras

Como criar heurísticas admissíveis?

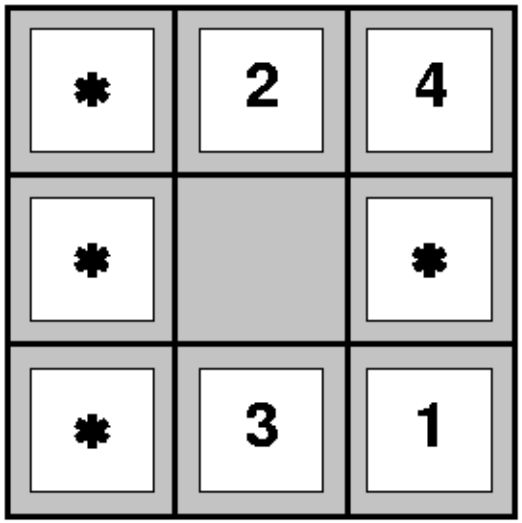
1- A solução de uma simplificação de um problema (problema relaxado) é uma heurística para o problema original.

- **Admissível:** a solução do problema relaxado não vai superestimar a do problema original
- É **consistente** para o problema original se for consistente para o relaxado.

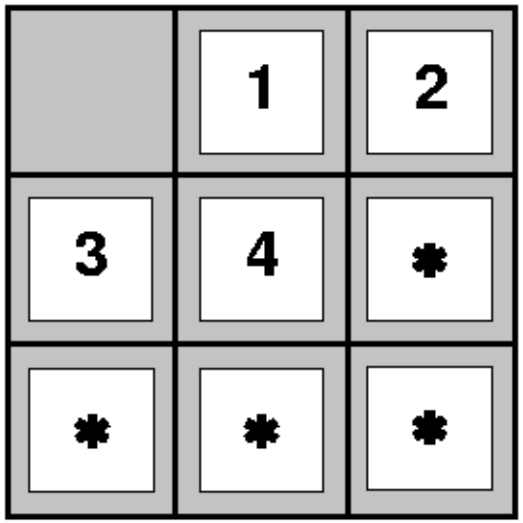
Exemplo

- h1 daria a solução ótima para um quebra-cabeças em que as peças pudessem se deslocar para qualquer lugar;
- h2 daria a solução ótima se um bloco pudesse se mover um quadrado por vez em qqr direção.

2- Custo da solução de um subproblema do problema original



Start State



Goal State

Medir o custo da solução exata sem se preocupar com os *

Limite inferior sobre o custo do problema completo

3- banco de dados de padrões:
armazenar os custos de soluções exatas para toda instância possível de subproblema (ex. toda configuração possível dos 4 blocos do espaço na fig. anterior).

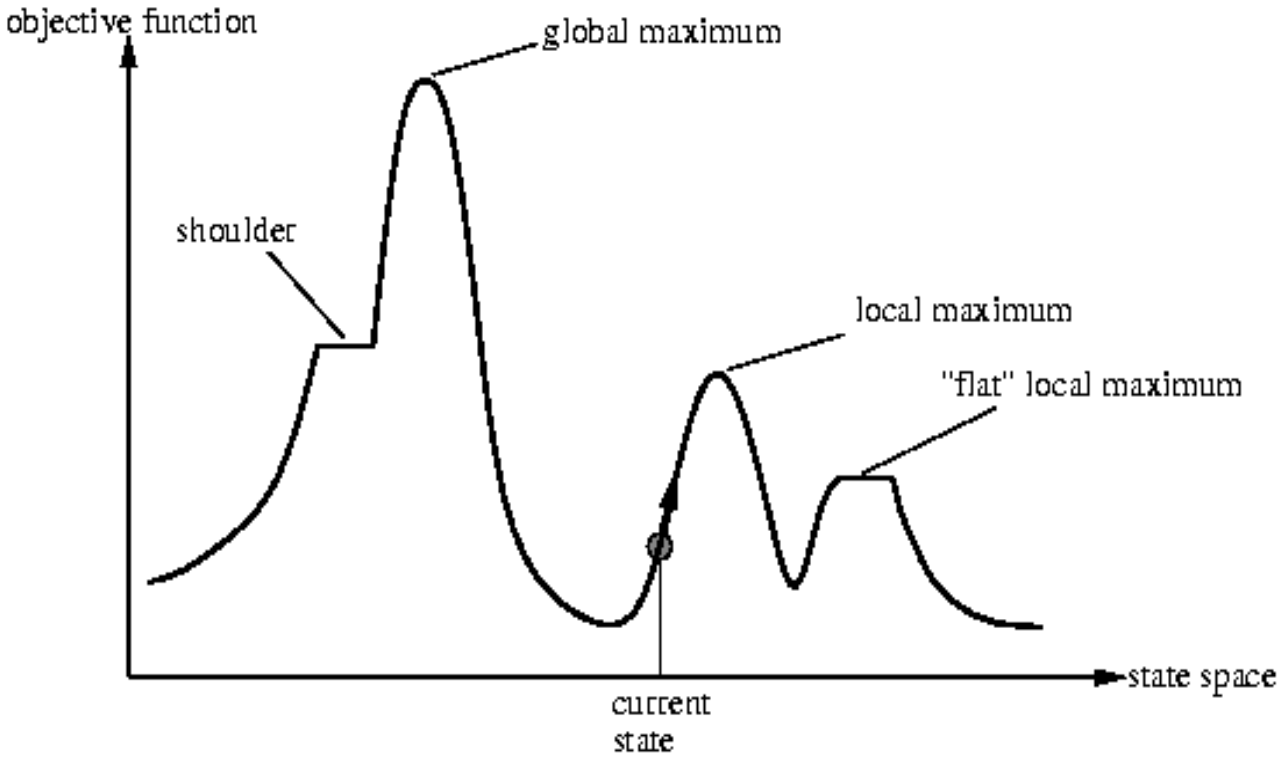
Algoritmos de busca local

Algoritmos de busca local

- Não se interessam pelo caminho de solução, mas com a configuração final (ex. 8 queens);
- Operam com um único estado corrente
- Usam pouca memória
- podem encontrar soluções **razoáveis** onde algoritmos sistemáticos se perdem

Topologia de espaço de estados

- elevação é o custo: encontrar o **mínimo global;**
- elevação é função objetivo: **máximo global.**



Busca de subida de encosta

- O algoritmo consiste em um laço repetitivo que percorre o espaço de estados no sentido do valor crescente (decrescente);
- Termina quando encontra um pico (vale) em que nenhum vizinho tem valor mais alto;

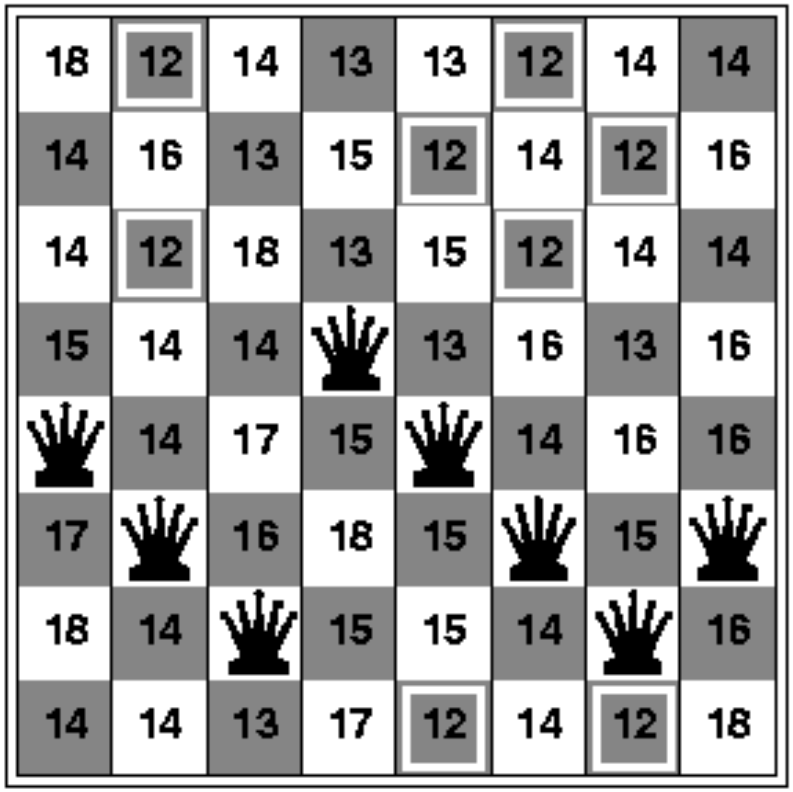
Busca de subida de encosta

- Não mantém uma árvore, o nó atual só registra o estado atual e o valor da função objetivo;
- Não examina antecipadamente valores de estados além dos vizinhos imediatos do estado corrente!

Busca de subida de encosta

- “ é como tentar alcançar o cume do Everest em meio a um nevoeiro durante uma crise de amnésia”

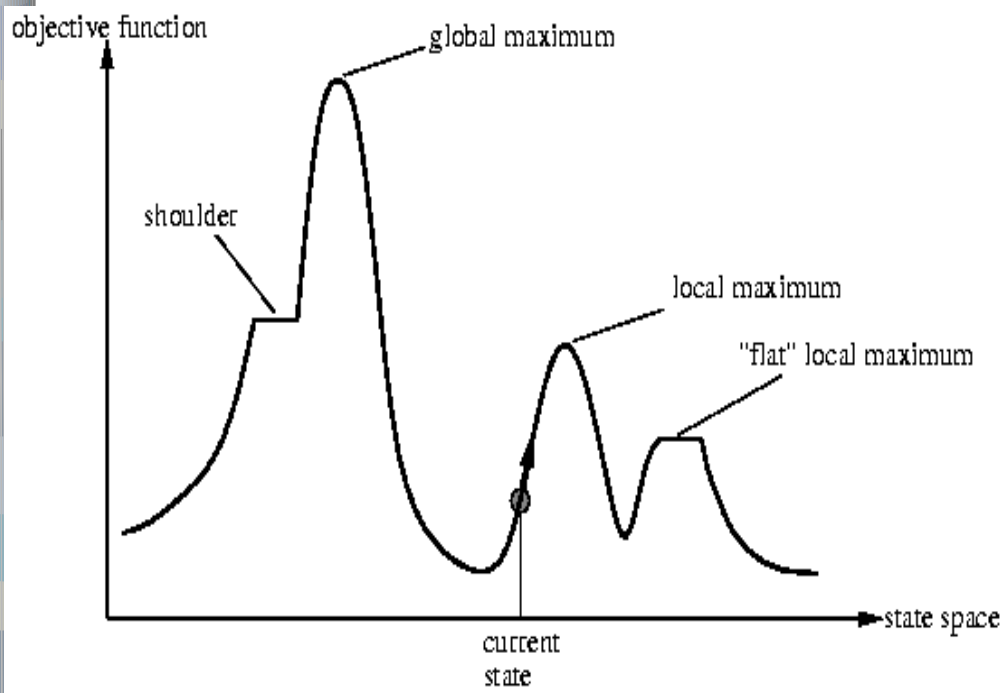
Busca de subida de encosta-sucessores

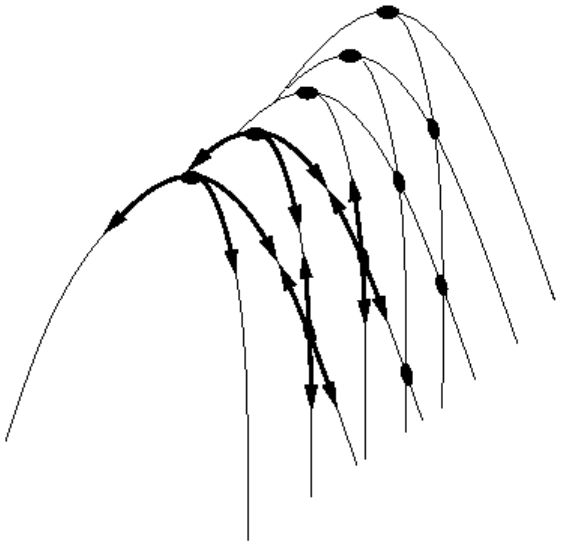


h=17 e valor de h para cada sucessor possível obtido pela **movimentação de uma** rainha dentro de sua coluna

Subida de encosta: problemas

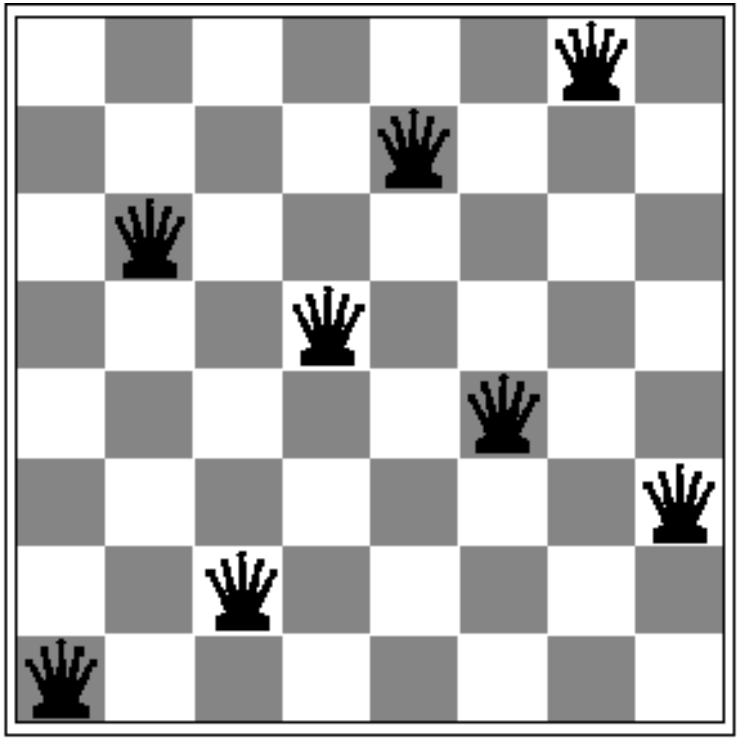
- Máximos locais, picos, platôs e planícies





Subida de encosta: problemas

- Ex. máximo local em 8 queens



h=1, mas qqr movimento de qqr rainha só piora a situação...

Subida de encosta: soluções

- Movimento lateral para evitar platôs
  - pode ocorrer repetição infinita se for um máximo local plano que não seja planície;
  - Impor um limite sobre o número de movimentos laterais;

Subida de encosta: soluções

- Subida de encosta com reinício aleatório
  - Conduz vária buscas subida de encosta a partir de diversos estados iniciais gerados ao acaso, para quando encontra um objetivo.
  - É completa, pois irá acabar gerando um estado objetivo como estado inicial. Porém ineficiente...

Busca de têmpera simulada (simulated annealing)

- Combina a subida de encosta com um percurso aleatório resultando em eficiência e completeza.
- Subida de encosta dando uma "chacoalhada" nos estados sucessores;
  - i.e. estados sucessores com avaliação pior que o estado corrente podem ser escolhidos com uma certa probabilidade;
  - esta probabilidade diminui com o tempo

Conclusão

- Buscas locais são amplamente utilizadas em projetos de circuitos integrados, layout de instalações industriais, otimização de redes de telecomunicações etc..
- O estudo de heurísticas ainda é muito intenso havendo até um *journal* dedicado a isso: *Journal of Heuristics* (Kluwer)
- LEIAM O CAP. 4 do Russell (até p. 114)