- Sumário:
  - Busca informada
    - Funções heurísticas
    - Algoritmos de busca local

Objectivos

- Explicar as abordagens para o desenho sistemático de heurísticas admissíveis
- Adquirir a noção de busca local
- Descrever alguns algoritmos de busca local: subida de encosta, busca em feixe

Busca informada (heurística)

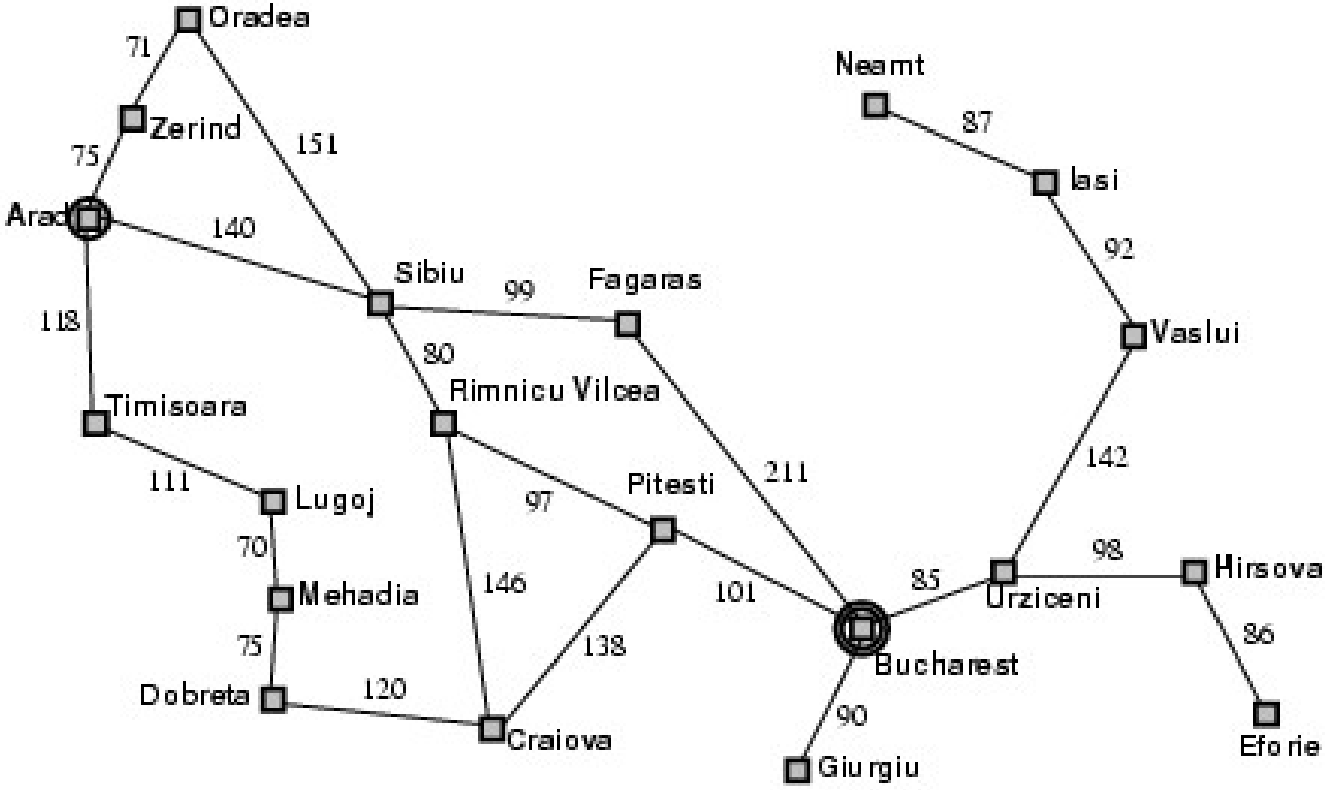
- Utiliza conhecimento específico sobre o problema para guiar o processo de busca, para além da informação contida na definição do problema
- Permite:
  - Encontrar soluções mais rápido
  - Encontrar soluções mesmo na presença de limitações de tempo
  - Geralmente encontram melhores soluções

Heurística admissível

- Uma heurística *h(n)* é admissível se para qualquer nó *n*, *h(n)* ≤ *h*(n)*, onde *h*(n)* é o custo verdadeiro de alcançar o estado objectivo a partir de *n*
  - Nunca sobrestima o custo de alcançar o objectivo (é optimista)

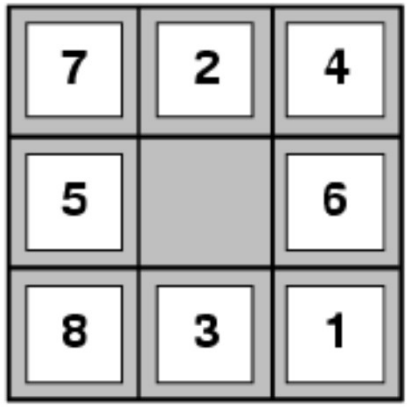
Heurística admissível: exemplo (I)

- $h_{DLR}(n)$: distância em linha recta nunca é maior que distância pela estrada

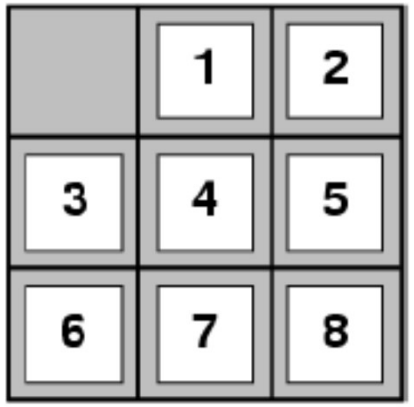


Heurística admissível: exemplo (II)

- Puzzle de 8 peças
  - $h_1(n)$: número de peças fora da posição
    - $h_1(ini) = 8$
  - $h_2(n)$: soma das distâncias de cada peça até à posição final
    - $h_2(ini) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$



Start State



Goal State

Qualidade das heurísticas

- Se pode caracterizar através do **factor de ramificação efectivo (*b**)**
- Suponhamos que para um problema dado:
  - O algoritmo A* gera *N* nós e
  - A profundidade da solução é *d*,
- ***b**** é o factor de ramificação que uma árvore uniforme teria para conter N + 1 nós

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

Qualidade das heurísticas: experiência (I)

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	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	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	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

- Boas funções heurísticas têm o valor de b* próximo de 1

Qualidade das heurísticas: experiência (II)

- h_2 é melhor que h_1 e muito melhor que IDS
- h_2 é sempre melhor que h_1
- Para qualquer nó *n* se cumpre que $h_2(n) \geq h_1(n)$
  - h_2 **domina** a h_1
  - Heurística dominante expande sempre menor quantidade de nós
- Caso existam muitas funções heurísticas para o mesmo problema, e nenhuma delas domine as outras, usa-se uma *heurística composta*:
  - $h(n) = max(h_1(n), h_2(n),\ldots,h_m(n))$
  - Assim definida, *h* é *admissível* e *domina* cada função h_i individualmente

Como criar uma função heurística

Fonte imagem: http://st.depositphotos.com/1654249/1946/i/950/depositphotos_19467373-3d-man-sitting-with-red-question-mark.jpg

Criação de funções heurísticas admissíveis

- Existem diferentes aproximações à criação de heurísticas admissíveis
  - Simplificação das condições do problema original
  - Utilização do custo de subproblemas do problema original

Criação de funções heurísticas: simplificação (I)

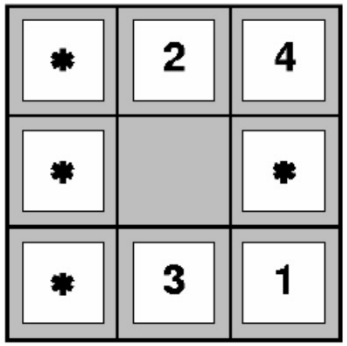
- O custo de uma solução óptima para a simplificação de um problema (*problema relaxado*) constitui uma heurística para o problema original
  - **Admissível**: a solução do problema relaxado não vai sobrestimar a do problema original.
  - É **consistente** para o problema original se for consistente para o relaxado
- Um problema relaxado é um problema que contém menores restricções às acções relativamente ao problema original

Criação de funções heurísticas: simplificação (II)

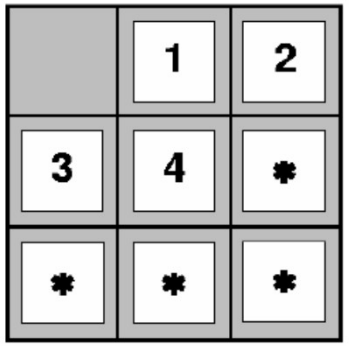
- No puzzle de 8 peças
  - h_1 daria a solução óptima para um problema "relaxado" em que as peças pudessem se deslocar para qualquer lugar
  - h_2 daria a solução óptima para um problema "relaxado" em que as peças pudessem se mover um quadrado por vez em qualquer direção

Criação de funções heurísticas: subproblemas (I)

- O custo da solução de um subproblema de um problema dado constitui uma heurística para o problema original
  - Calcular o custo da solução exacta sem nos preocuparmos com os *



Start State



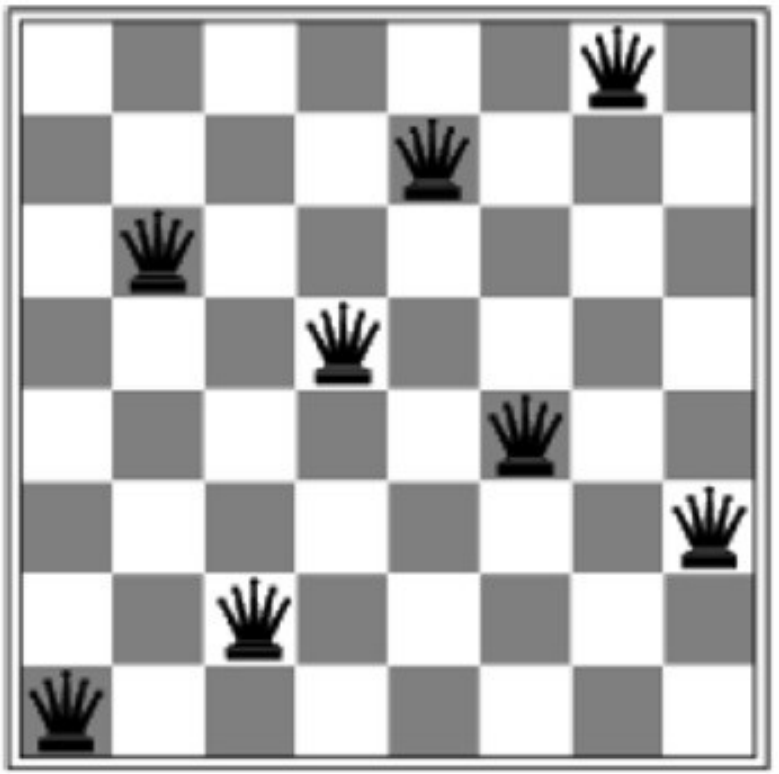
Goal State

Criação de funções heurísticas: subproblemas (II)

- Com os **custos das soluções de cada configuração** possível do subproblema se constrói uma **base de dados de padrões**
- A heurística para cada estado encontrado durante a busca se determina localizando a correspondente configuração do subproblema na base de dados

Algoritmos de busca local (I)

- Em muitos problemas a trajectória até ao objectivo não é relevante
- No problema das 8 rainhas o que interessa é a configuração final e não a ordem em que elas são acrescentadas



Algoritmos de busca local (II)

- Vários problemas do género
  - Desenho de circuitos integrados, agendamento de tarefas, roteamento de veículos…

Algoritmos de busca local (III)

- Funcionam utilizando-se apenas um estado actual, em vez de trajectórias
- Geralmente só é possível movimentar-se para os vizinhos do estado actual
- As trajectórias seguidas durante a busca não são guardadas

Algoritmos de busca local (IV)

- Apesar de não ser sistemáticos:
  - Utilizam pouca memória
  - Geralmente encontram soluções razoáveis em grandes ou infinitos (contínuos) espaços de estados
  - São úteis para resolver problemas de optimização
    - O objectivo é determinar a melhor configuração de acordo a uma função objectivo

Subida de encosta (*hill climbing)* (I)

- Consiste num ciclo que percorre continuamente o espaço de estados no sentido do valor crescente (ou decrescente) de uma função objectivo
- Termina quando encontra um pico (ou vale) no qual nenhum vizinho tem valor mais alto (ou mais baixo)

Subida de encosta (*hill climbing*) (II)

- Na estrutura de dados do nó actual se guarda apenas o estado e o correspondente valor da função objectivo
- Não examina antecipadamente valores de estados além dos valores dos vizinhos imediatos do estado atual
- "É como subir o Everest em meio a um nevoeiro durante uma crise de amnésia"

Subida de encosta (*hill climbing*) (III)

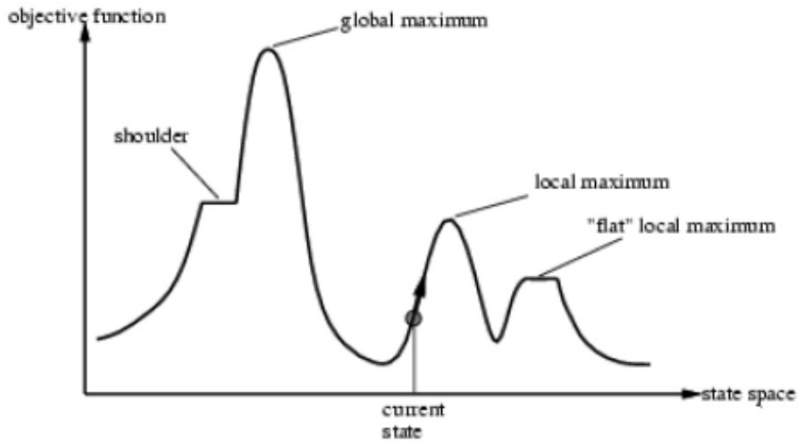
```
Função Hill-Climbing(Problema) retorna um estado que é
   o máximo local
Inicio
  EstadoActual ← FazNó(Problema[EstadoInicial])
  loop do
    Vizinho ← SucessorDeMaiorValor(EstadoActual)
    se Vizinho[Valor] for menor ou igual
  EstadoActual[Valor] então
       retorna EstadoAtual
    EstadoActual ← Vizinho
Fim
```

Subida de encosta: exemplo

- Cada estado se compõe por 8 rainhas no tabuleiro (1 por coluna) → formulação de estado completo
- Função sucessor → retorna todos os possíveis estados gerados pelo movimento de uma rainha para outra posição na mesma coluna (56 sucessores)
- Heurística → número de pares de rainhas atacando-se

Subida de encosta: problemas

- Com frequência se atinge pontos a partir do qual o algoritmo não é capaz de progredir, devido a:
  - Máximos (mínimos) locais
  - Planaltos



Subida de encosta: variantes

- Subida de encosta com movimentos laterais
- Subida de encosta com reinícios aleatórios
- Subida de encosta estocástica
- Subida de encosta com primeira escolha…

Busca em feixe local (*local beam search*)

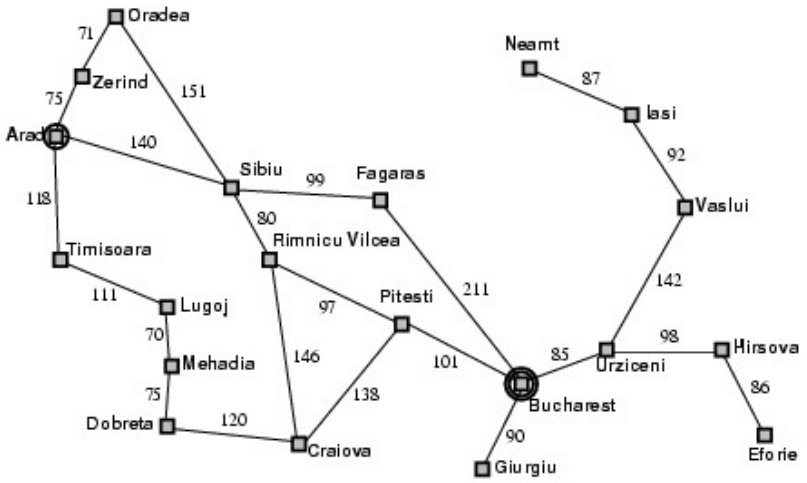
- Segue um conjunto de *k* estados em vez de um único
- Inicia com *k* estados gerados aleatoriamente
- Em cada passo todos os sucessores dos *k* estados são gerados
- Se qualquer deles for o objectivo o algoritmo termina
- Caso contrário, selecciona os *k* melhores sucessores da lista completa e recomeça

Outros

- Recozimento simulado (*simulated annealing*)
- Algoritmos genéticos
- Busca tabu…

Tarefa

Representar a operacão da busca A* aplicada ao problema de ir até Bucareste a partir de Lugoj usando a heurística da distância em linha recta. Mostre a sequência de g e h para cada nó.



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
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374
Hirsova	151	Urziceni	80

Bibliografia

- Russell & Norvig, pg. 105 – 113
- Costa & Simões, pg. 108 – 116
- Palma Méndez & Marín Morales, pg. 353 – 362, 369 – 373