Inteligência Artificial

Busca Informada

Prof. Eduardo R. Hruschka

Agenda

- Estratégias de Busca Informada
- Função Heurística
- Busca Local e Otimização

  - Leitura Recomendada:
    - Russell, S., Norvig, P., Artificial Intelligence – A Modern Approach, Second Edition, Prentice Hall - Capítulo 4 **– Informed Search and Exploration**.

Créditos e Agradecimentos

Adaptado das notas de aula de Tom Lenaerts,

Vlaams Interuniversitair Instituut voor Biotechnologie

- **http://aima.cs.berkeley.edu**

- Os slides traduzidos foram gentilmente cedidos pelo Prof. Ricardo J. G. B. Campello (ICMC/USP).

Busca *Best-First*

- Abordagem Geral de Busca Informada:
  - "Informada" = Baseada em conhecimento de domínio.
  - Busca *Best-First*: nodo é selecionado para expansão baseado em uma *função de avaliação f(n)*
- Idéia: Estimar a distância até a meta.
  - Escolhe o nodo que *aparenta* ser o melhor - segundo *f(n)*.
- Implementação:
  - Fila de prioridades, na qual a chave é *f(n)*.
  - Realizações: Busca Gulosa, Busca A*

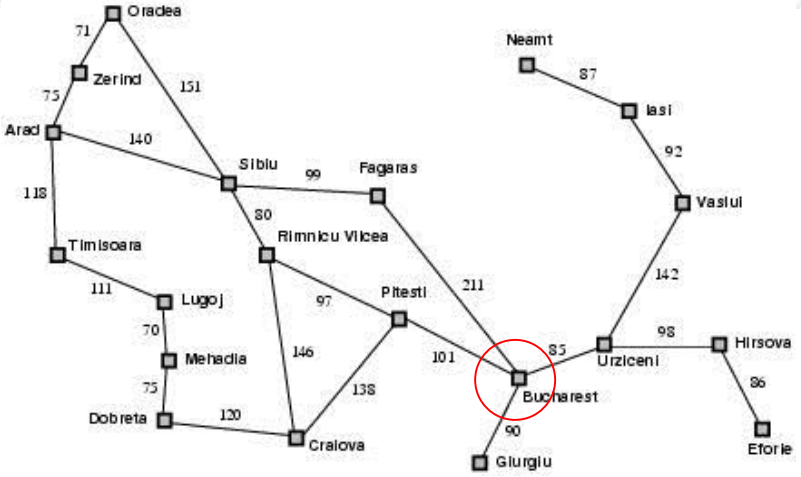
Função Heurística

- *"A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood".*
  - *h(n)* = custo estimado do caminho mais curto de *n* até o nó meta.
  - Condição Básica: Se *n* é a meta então *h(n)* = 0.

Exemplo: Mapa da Romênia

Distâncias em linha reta até Bucareste

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

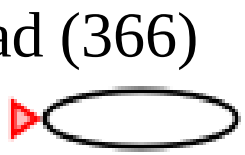


Custos de Passo para o Problema

- h_{SLD} = heurística da distância em linha reta (*straight-line distance*).
- h_{SLD} **Não** pode ser calculada da descrição do problema em si.
- Neste exemplo: $f(n) = h(n)$
  - Expande o nodo "mais perto" da meta.

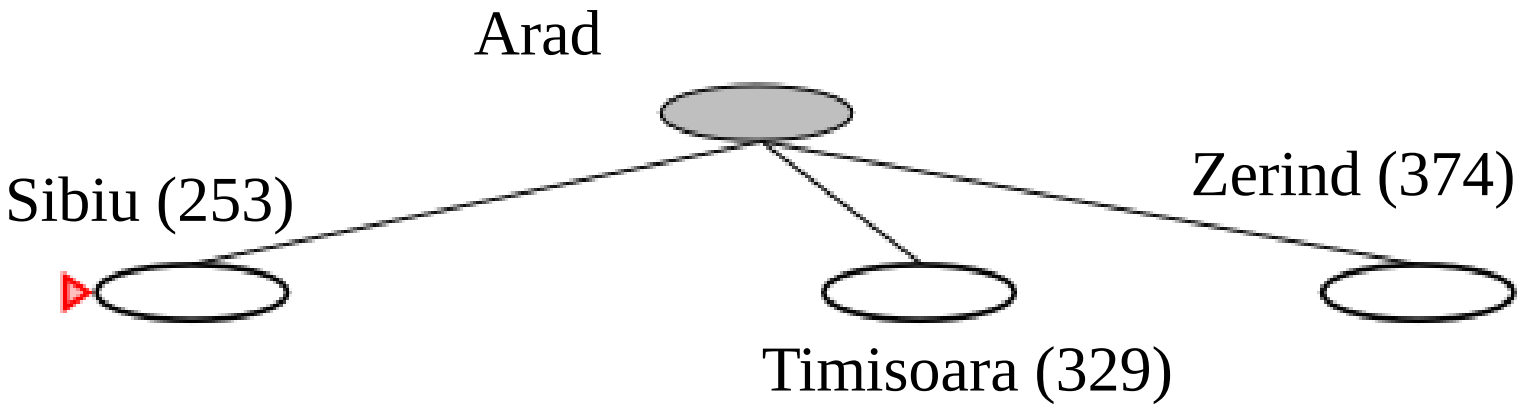
  = **Busca *Best-First* Gulosa**.

Busca Gulosa



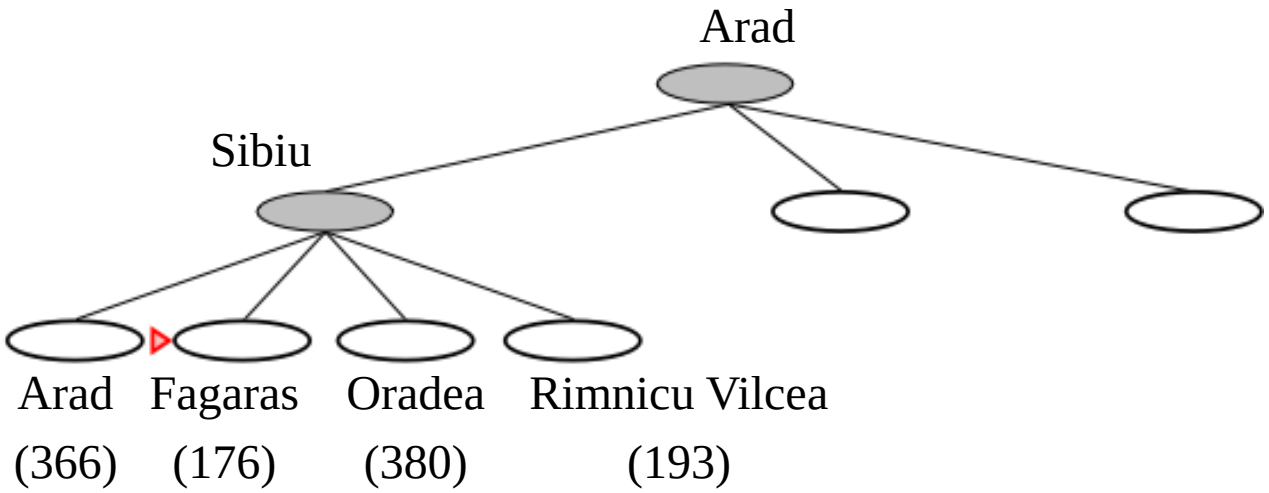
- Utilizar busca gulosa para viajar de Arad até Bucareste.
- Estado Inicial = Arad.

Busca Gulosa



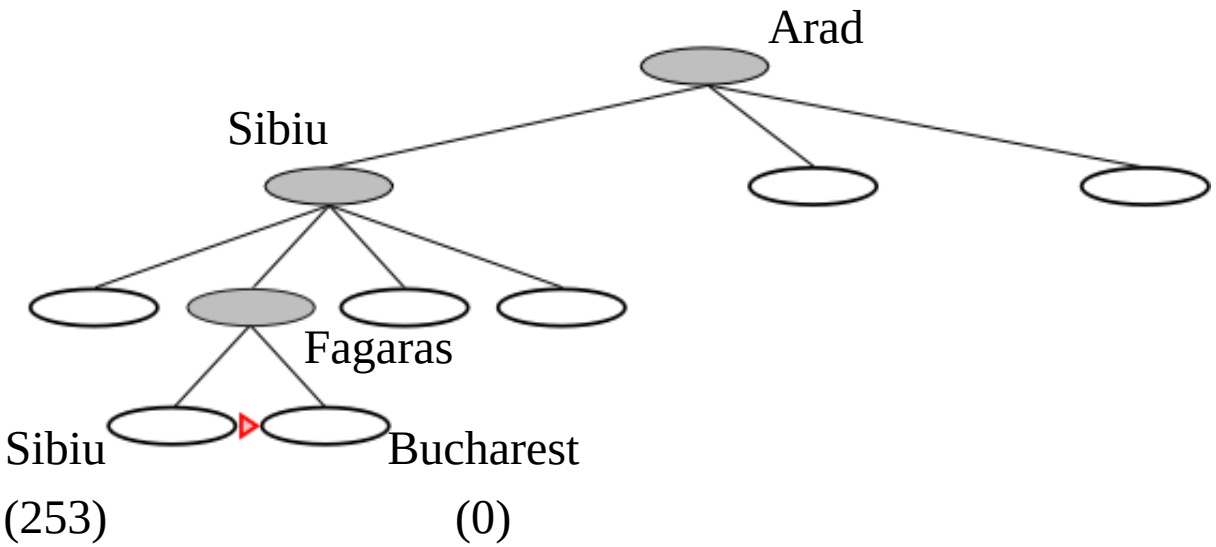
- O primeiro passo de expansão produz:
  - Sibiu, Timisoara e Zerind.
- Busca Gulosa irá selecionar Sibiu.

Busca Gulosa



- Quando Sibiu é expandida obtém-se:
  - Arad, Fagaras, Oradea e Rimnicu Vilcea.
- Busca Gulosa irá selecionar Fagaras.

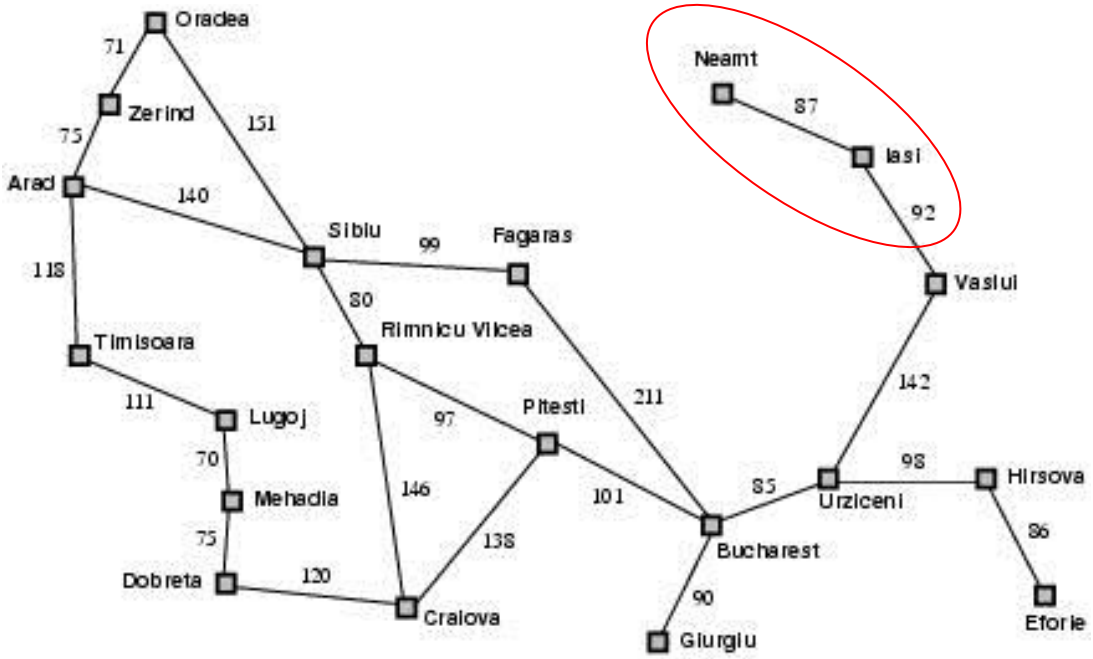
Busca Gulosa



- Expandindo Fagaras obtém-se:
  - Sibiu e Bucareste.
- Meta Alcançada
  - Porém não ótima (veja Arad, Sibiu, Rimnicu Vilcea, Pitesti).

Busca Gulosa

- Completude: NÃO.
  - Assim como busca em profundidade, tenta seguir ao longo de um único caminho até a meta, podendo nunca retornar devido a estados repetidos.
  - Minimizar *h(n)* pode produzir laços infinitos:
    - E.g., de Iasi para Fagaras fica-se preso indefinidamente entre Iasi e Neamt.



Busca A*

- Idéia: considerar também o custo de alcançar cada nodo candidato *n* a partir do nodo (estado) inicial.
- Função de Avaliação $f(n) = g(n) + h(n)$
  - $g(n)$ Custo (já conhecido) de alcançar o nodo *n*.
  - $h(n)$ Custo estimado do nodo *n* até o nodo meta.
  - $f(n)$ Custo estimado total do caminho através de *n* até a meta.

OBS. $g(n) = 0 \Rightarrow$ Busca Gulosa

$h(n) = 0 \Rightarrow$ Busca Não Informada de Custo Uniforme

Busca A*

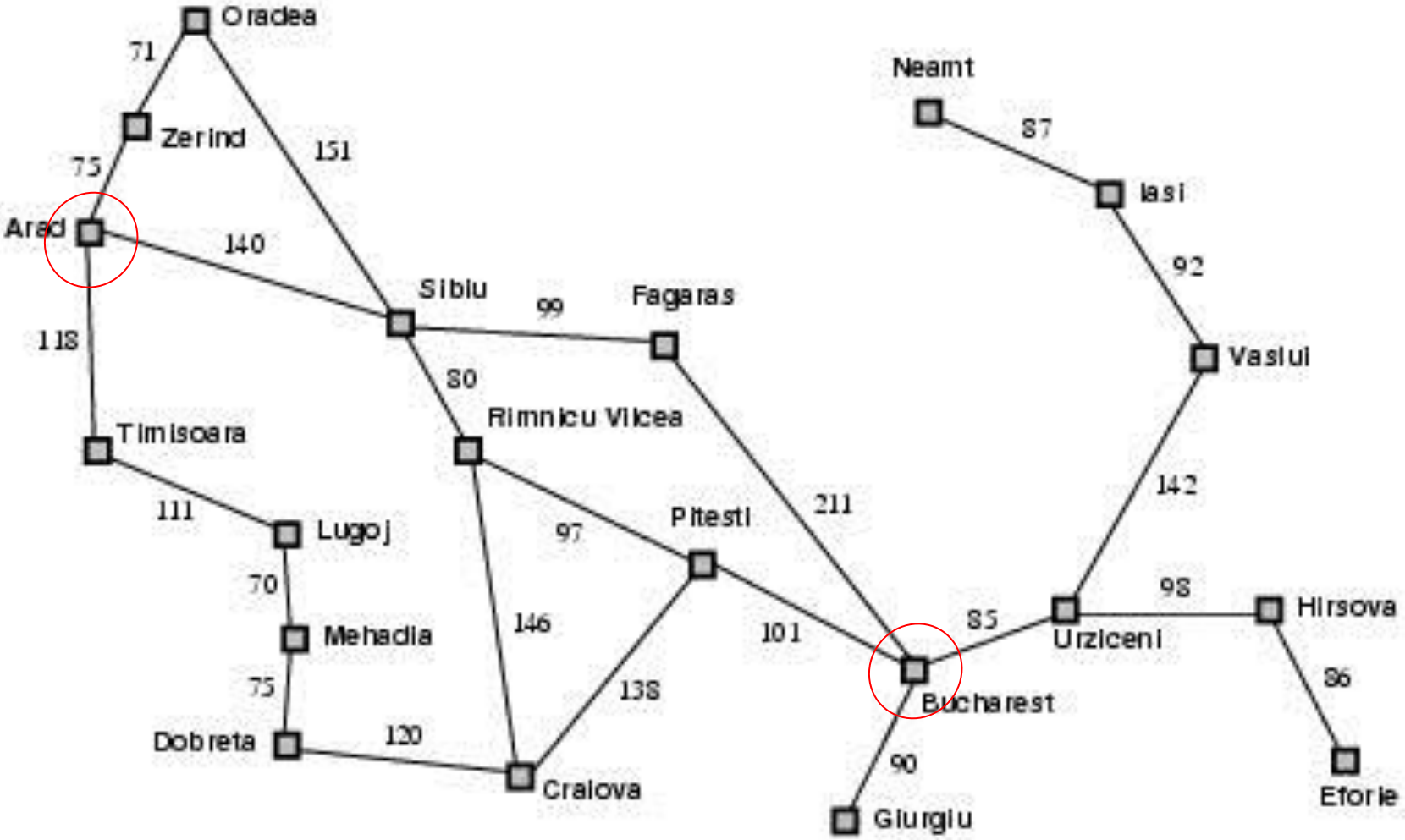
- Busca A* utiliza uma heurística **admissível**
  - *Nunca superestima* o custo para alcançar a meta.
  - Ou seja, heurísticas admissíveis são otimistas.

  - Formalmente:

    1. $h(n) \leq h^*(n)$, $h^*(n)$ é o custo verdadeiro de n até *a meta*.
    2. Dado $h(n) \geq 0$, tem-se $h(G) = 0$ para qualquer meta G.

    E.g., $h_{SLD}(n)$ nunca superestima a distância real via estrada.

Busca A*

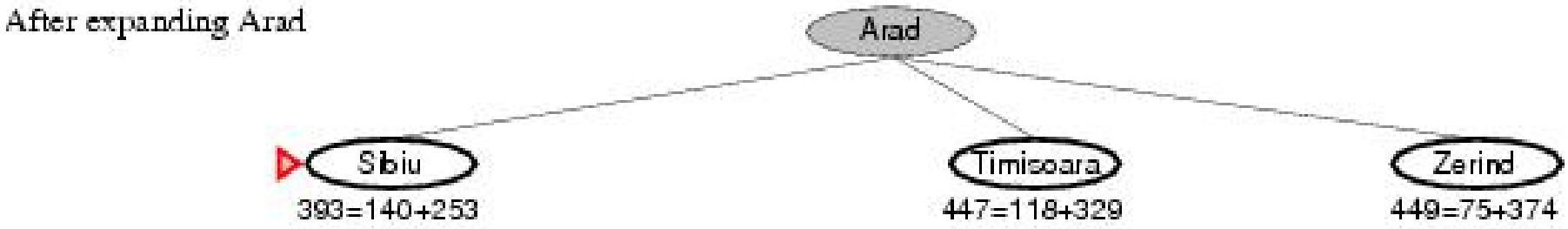

Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Dobreta
120
Craiova
90
Giurgiu
Neamt
87
Iasi
92
Vaslui
142
Eforie

Busca A*

(a) The initial state

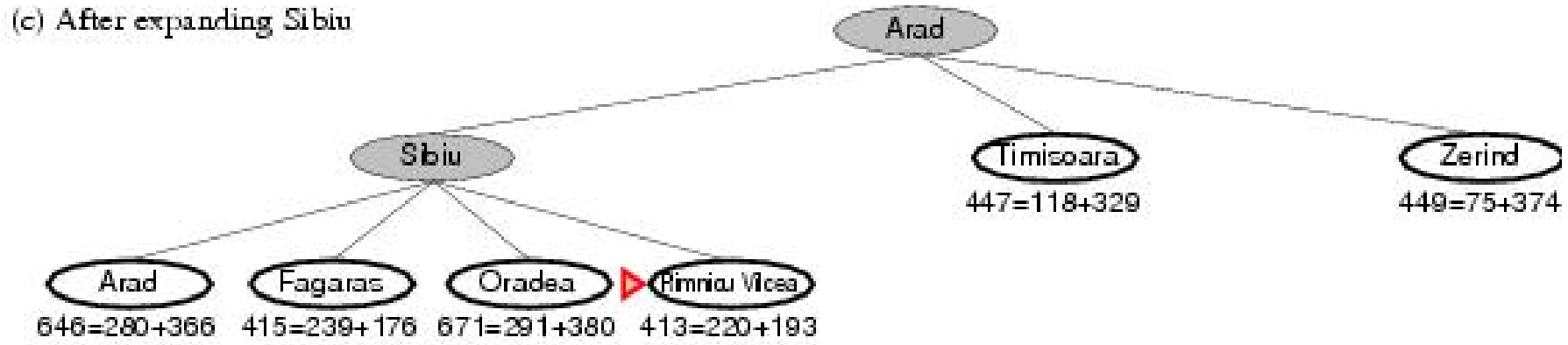
- Encontrar Bucareste partindo de Arad:
  - f(Arad) = 0 + h(Arad) = 0 + 366 = 366

Busca A*



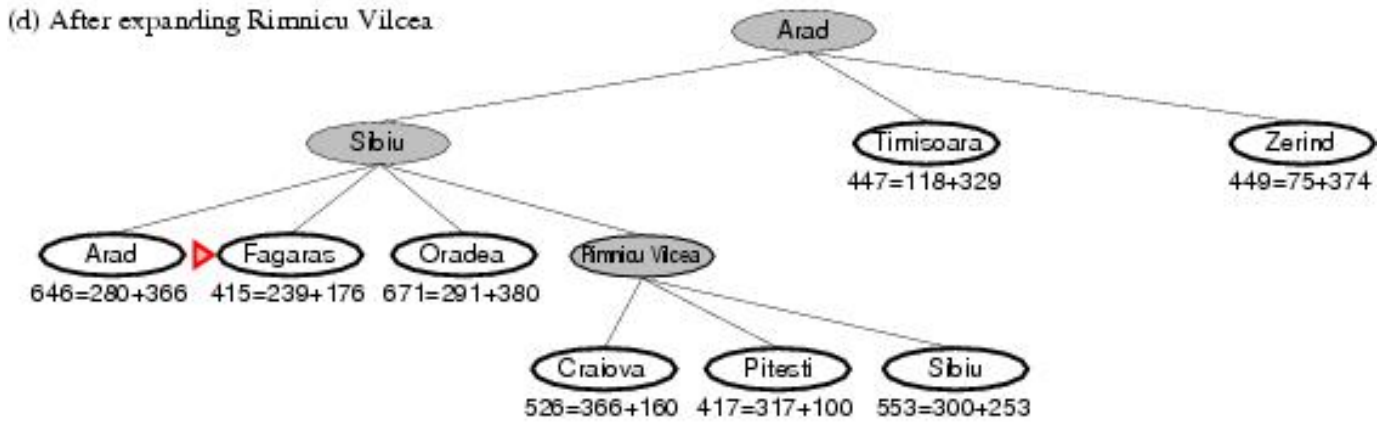
- Expandindo Arad e determinando *f(n)* para cada nó:
  - f(Sibiu) = c(Arad,Sibiu) + h(Sibiu) = 140 + 253 = 393.
  - f(Timisoara) = c(Arad,Timisoara) + h(Timisoara) = 118 + 329 = 447.
  - f(Zerind) = c(Arad,Zerind) + h(Zerind) = 75 + 374 = 449.
- Melhor escolha é Sibiu.

Busca A*



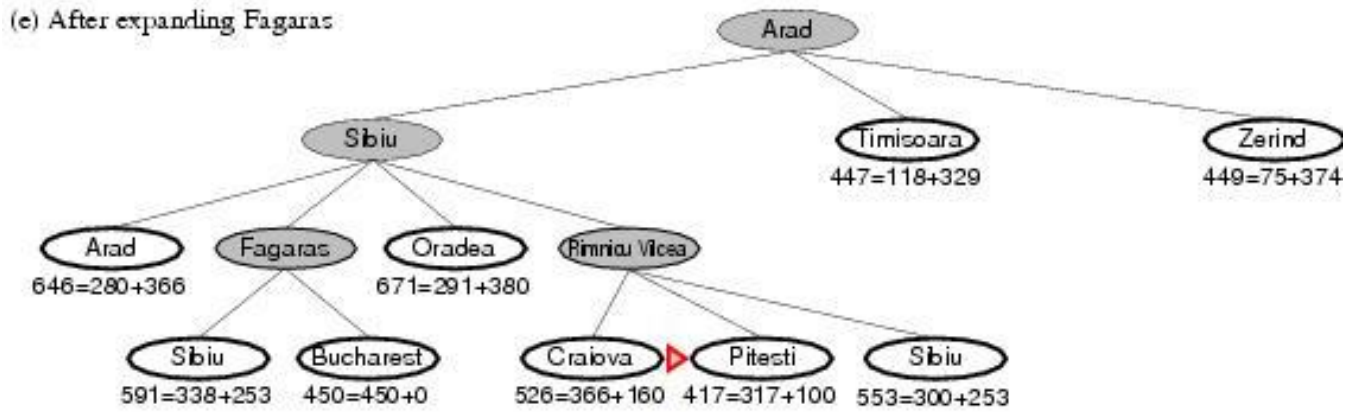
- Expandindo Sibiu e determinando *f(n)* para cada nodo:
  - f(Arad) = g(Arad) + h(Arad) = 280 + 366 = 646.
  - f(Fagaras) = g(Fagaras) + h(Fagaras) = 239 + 176 = 415.
  - f(Oradea) = g(Oradea) + h(Oradea) = 291 + 380 = 671.
  - f(Rimnicu Vilcea) = g(Rimnicu Vilcea) + h(Rimnicu Vilcea) = 220 + 193 = 413.
- Melhor escolha é Rimnicu Vilcea.

Busca A*



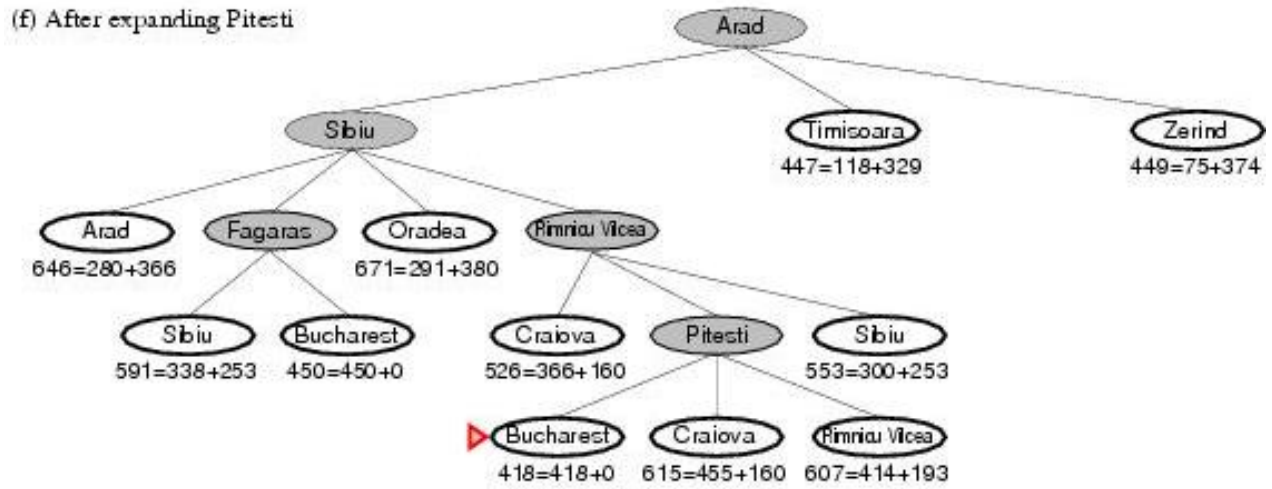
- Expandindo Rimnicu Vilcea:
  - f(Craiova) = g(Craiova) + h(Craiova) = 366+160 = 526.
  - f(Pitesti) = g(Pitesti) + h(Pitesti) = 317 + 100 = 417.
  - f(Sibiu) = g(Sibiu) + h(Sibiu) = 300 + 253 = 553.
- Melhor escolha é Fagaras.

Busca A*



- Expandindo Fagaras:
  - f(Sibiu) = g(Sibiu) + h(Sibiu) = 338 + 253 = 591.
  - f(Bucareste) = g(Bucareste) + h(Bucareste) = 450 + 0 = 450.
- Melhor escolha é Pitesti.

Busca A*



- Expandindo Pitesti:
  - *f(Bucareste) = g(Bucareste) + h(Bucareste) = 418 + 0 = 418.*
- Melhor escolha é Bucareste.
  - Solução ótima (dado que *h(n)* é **admissível**).

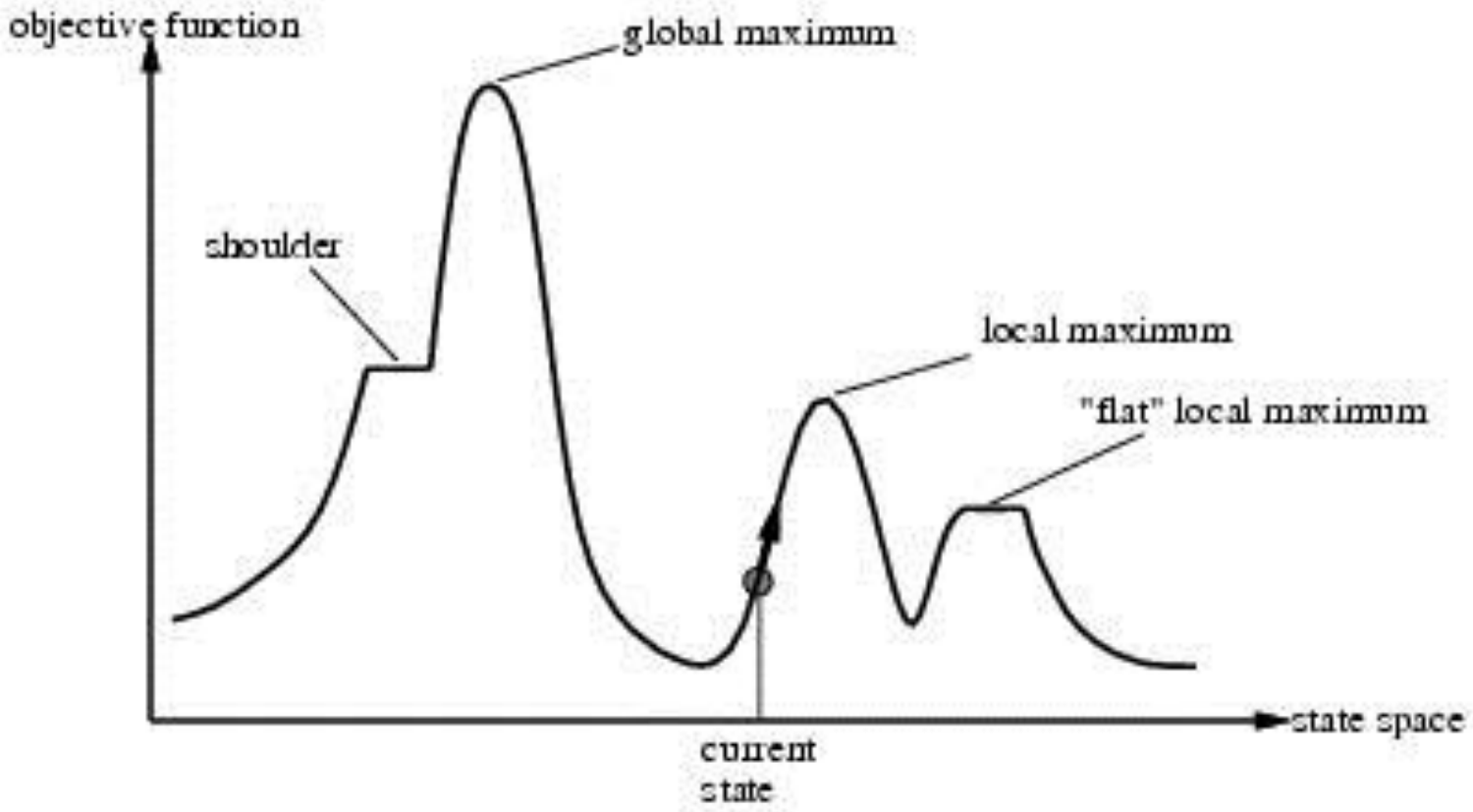
Busca Local e Otimização

- Anteriormente: exploração sistemática do espaço de busca.
  - Caminho para a meta é a solução para o problema.
- Em muitos problemas o caminho é irrelevante.
  - E.g, encontrar máximos/mínimos de funções multimodais.
- Algoritmos de natureza distinta podem ser utilizados:
  - Busca Local.

Busca Local e Otimização

- Busca Local = armazenamento apenas do estado corrente e movimento para estados vizinhos.
- Vantagens:
  - Utiliza pouca memória (usualmente uma quantidade constante).
  - Frequentemente encontra boas soluções em espaço de estados muito grandes ou mesmo infinitos, nos quais estratégias sistemáticas são inadequadas ou inviáveis.

Busca Local e Otimização



Hill-Climbing

- Política de busca local cujos movimentos se dão impreterivelmente na direção de valores crescentes (ou não decrescentes) da função objetivo.
  - Termina quando um pico (mínimo ou máximo) é alcançado.
- Move-se apenas para “estados vizinhos” ao corrente.
- O conceito de estado vizinho depende do problema.
- *Hill Climbing* é considerada uma *estratégia gulosa*.

Hill-Climbing

- Diferentes versões:
  - Original (maior passo): move-se para o estado vizinho que proporciona a maior melhoria (redução ou aumento) no valor da função objetivo.
  - *First-Choice*: versão estocástica onde se move para algum estado vizinho sorteado aleatoriamente e que esteja entre aqueles (se existirem) que melhoram o valor corrente da função objetivo.
    - Interrompe a busca após um limite pré-especificado de tentativas de movimento frustradas.
    - Interessante em problemas nos quais a quantidade de vizinhos é grande.
  - *Random-Restart*: Múltiplas buscas *hill-climbing* a partir de diferentes estados iniciais gerados aleatoriamente.
    - Interessante em problemas com superfícies de otimização complexas (muitos mínimos ou máximos locais não satisfatórios).

Hill-Climbing

Algoritmo Básico (versão maior passo)

Inicialização: Faça índice de iterações t = 0 e escolha estado inicial $\mathbf{x}_t = \mathbf{x}_0$.

Ótimo local: Pare se nenhuma ação (movimento) do conjunto *G* de possíveis ações a partir do estado corrente leva a um estado melhor.

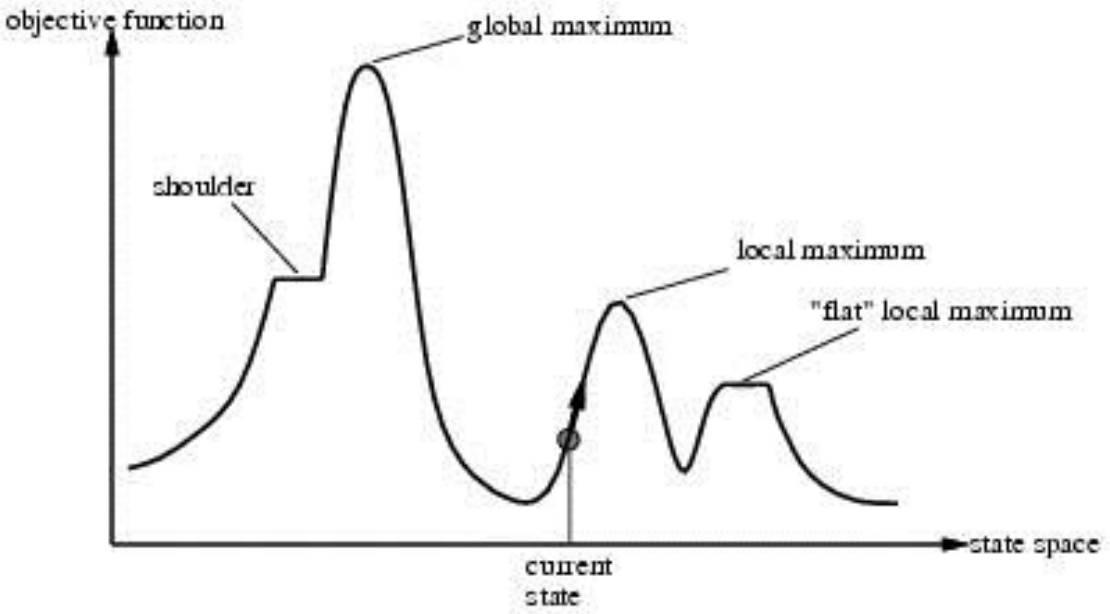
Movimento: Escolha a ação $g \in G$ que leve ao estado com melhor função objetivo (melhor do que o estado corrente).

Atualização: Aplique *g* sobre o estado corrente $\mathbf{x}_t$ obtendo assim $\mathbf{x}_{t+1}$.

Incremento: Faça t = t+1 e retorne ao passo "Ótimo local".

Hill-Climbing

Principais Dificuldades:



- Ótimos locais: picos mínimos/máximos de abrangência local.
- Platôs = áreas do espaço de estados onde a função objetivo é constante (plana).

Busca Tabu

- Algoritmo capaz de escapar de ótimos locais permitindo movimentos “ruins”.
  - Idéia:
    - permitir movimentos que pioram o valor da FO;
    - proibir por um dado número de iterações movimentos que possam retornar a busca a um estado anterior.

Busca Tabu

Algoritmo Básico

Inicialização: Faça t = 0 e escolha um estado inicial $\mathbf{x}_t = \mathbf{x}_0$ e um limite de iterações t_{max}. Crie uma lista (vazia) para ações tabu.

Limite de Parada: Pare se $t = t_{max}$ e tome como solução o estado com melhor função objetivo dentre os t_{max} estados explorados.

Movimento: Procure aleatoriamente por uma ação $g \in G$ não tabu que leve a um estado vizinho com função objetivo melhor do que a corrente. Caso não seja encontrada tal ação dentro de um limite de tentativas, selecione qualquer ação não tabu (e.g. a menos pior).

Atualização: Aplique a ação g sobre o estado corrente $\mathbf{x}_t$ obtendo assim $\mathbf{x}_{t+1}$.

Lista Tabu: Retire ações que permaneceram na lista por um no. de iterações e insira ações relacionadas a g.

Incremento: Faça t = t+1 e retorne ao passo "Limite de Parada".

Simulated Annealing

- Assim como a busca tabu, algoritmo capaz de escapar de ótimos locais permitindo movimentos “ruins”.
  - Idéia: decrescer gradualmente a freqüência (e eventualmente a amplitude) desses movimentos.
  - Funciona se o decrescimento for lento o suficiente.
  - Origem: metalurgia (materiais temperados).

  → Há várias (outras) metaheurísticas, tais como AEs (próxima aula)...