Algoritmos de Busca Informada

Problemas de Satisfação de Restrições

Sumário

Algoritmos de Busca Informada:

- ♢ Buscas *best-first* e A*
- ♢ Buscas IDA* e SMA* (CT215)
- ♢ Busca *Hill-climbing* e *Simulated annealing*

Problemas de Satisfação de Restrições:

- ♢ Exemplos de PSR
- ♢ Busca genérica aplicada à PSRs
- ♢ *Backtracking*
- ♢ Verificação *forward*
- ♢ Heurísticas para PSRs

Busca best-first

Idéia: use uma *função de avaliação* para cada nó – estimativa de "desejabilidade"

⇒ Expanda nó mais "desejável"

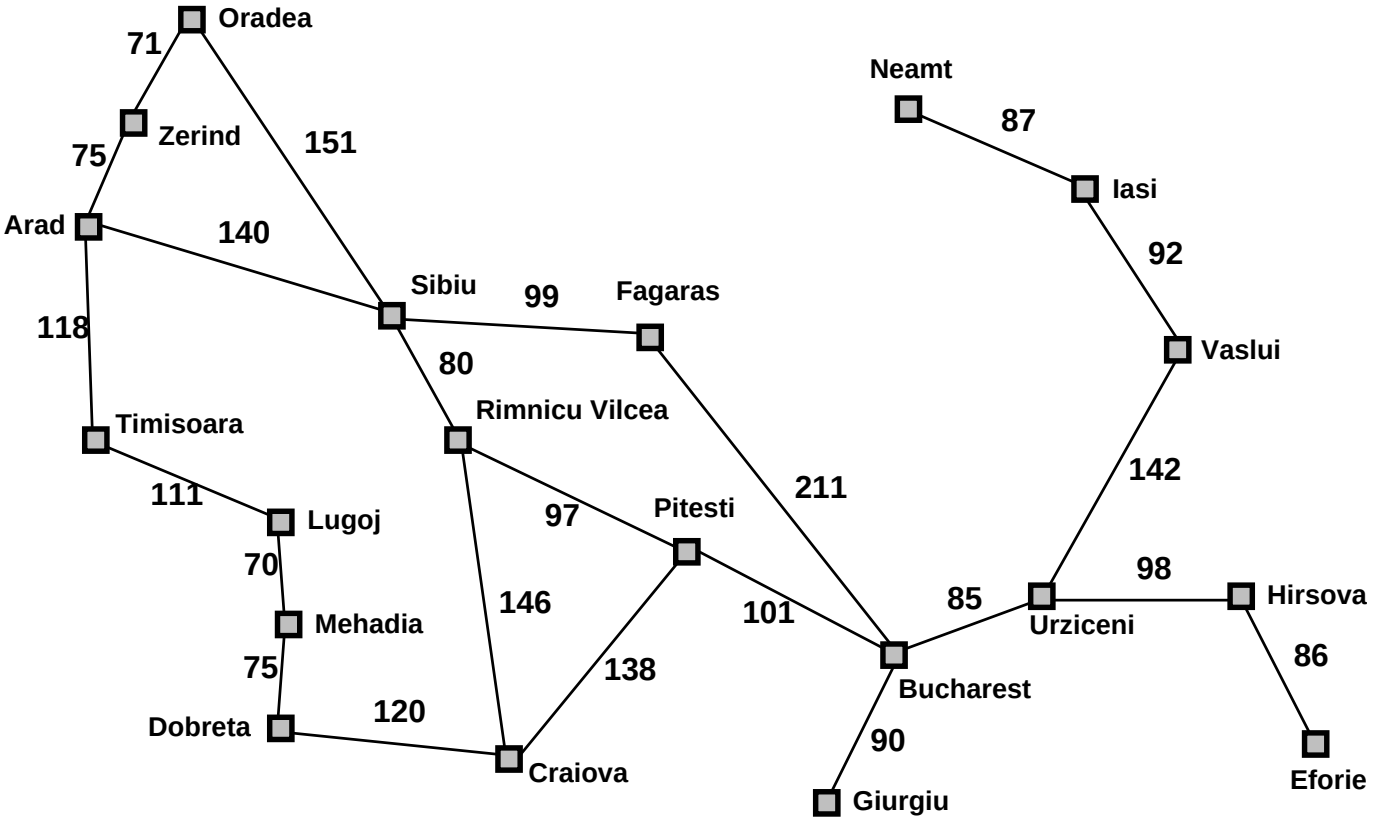
Implementação :
QUEUEINGFN = insira sucessores em ordem decrescente de "desejabilidade"

Casos especiais:

♢ Busca gulosa (*greedy*)

♢ Busca A*

Romênia com custos de passo em km



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

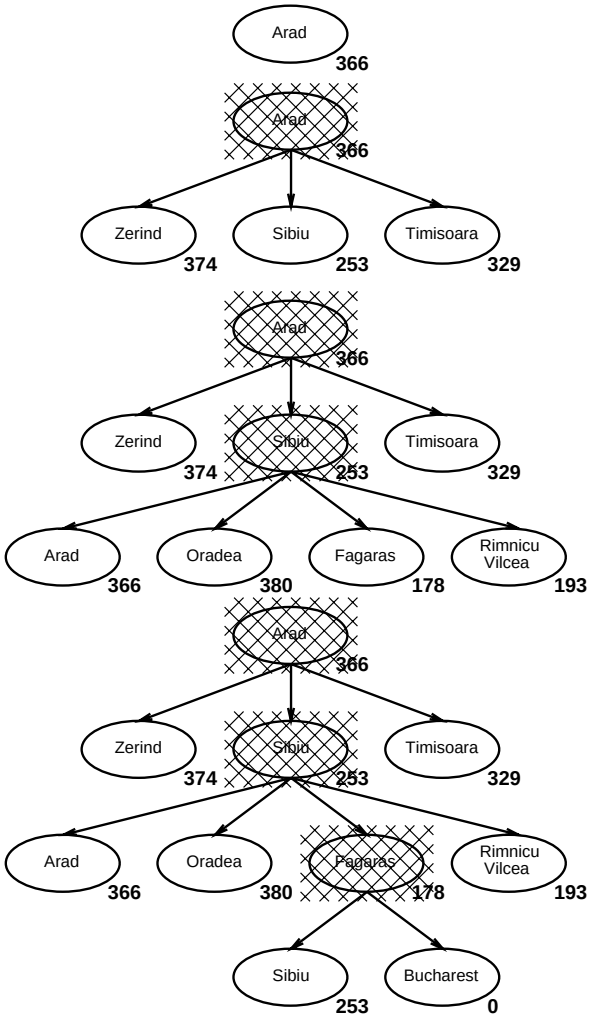
Busca Gulosa

Função de avaliação (desejabilidade) $h(n)$: uso estimativa de custo $\hat{h}(n)$ desde n até objetivo (heurística)

Por exemplo, $\hat{h}_{\mathrm{SLD}}(n) =$ distância em linha reta de n para Bucareste

A busca *greedy* expande o nó que **parece** ser mais próximo ao objetivo

Exemplo de busca gulosa



Propriedades da busca gulosa

Completa?? Não–pode ficar presa em ciclos
Com verificação de estados repetidos, é completa em espaços finitos.

Tempo?? $O(b^m)$, mas uma boa heurística pode levar à uma melhoria dramática.

Espaço?? $O(b^m)$—mantém todos os nós em memória.

Ótima?? Não.

Busca A*

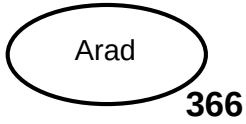
Idéia: evite expandir caminhos que já ficaram caros.

Para cada nó n, definamos uma função de avaliação $f(n) = g(n) + h(n)$, onde:
$g(n)$= custo real da trajetória de custo mínimo para alcançar n
$h(n)$= custo real mínimo desde n até um estado-objetivo
$f(n)$= custo real mínimo do caminho para objetivo via n

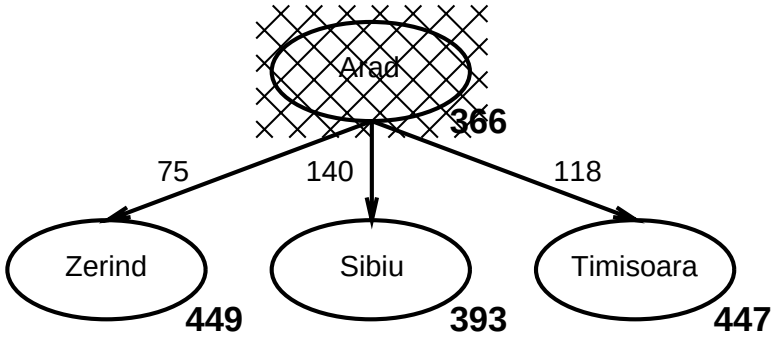
Para uma expansão arbitrária, $h(n)$ deve ser estimado (usando-se uma heurística), e, a princípio, o custo da trajetória até n não é necessariamente o menor possível. Portanto, temos $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$, onde:
$\hat{g}(n)$= custo da trajetória encontrada (até o momento) para alcançar n
$\hat{h}(n)$= custo mínimo estimado desde n até um estado-objetivo.

A* expande o nó com com menor $\hat{f}(n)$

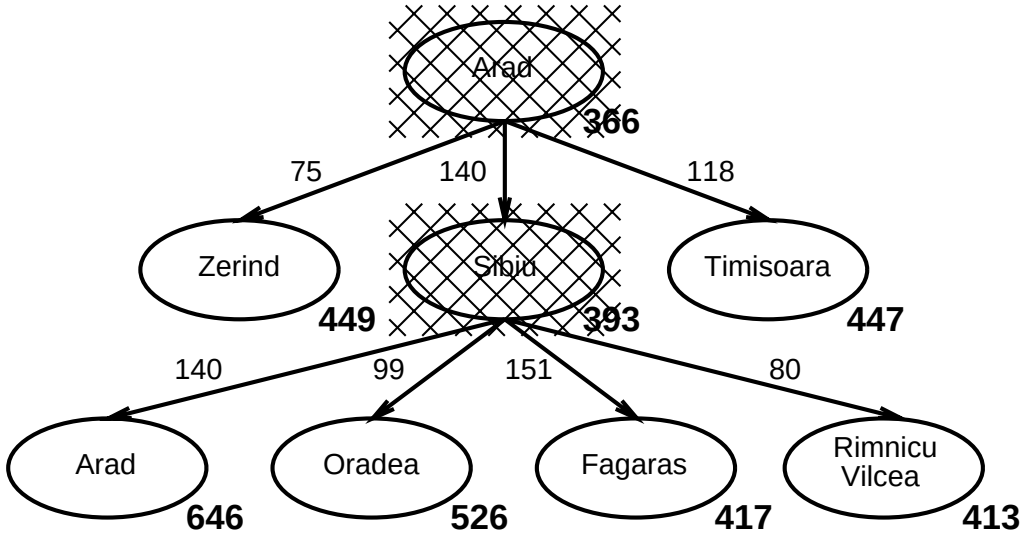
Exemplo de busca A^*



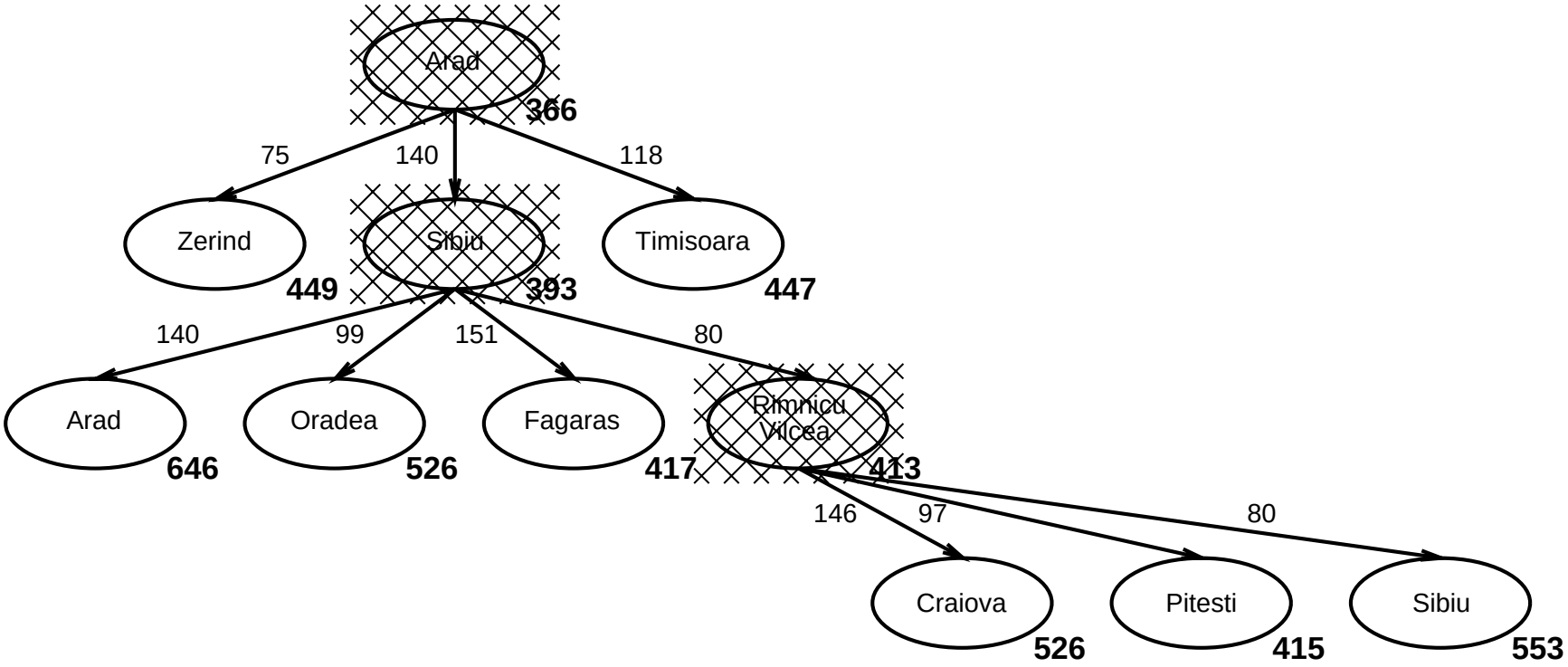
Exemplo de busca A*



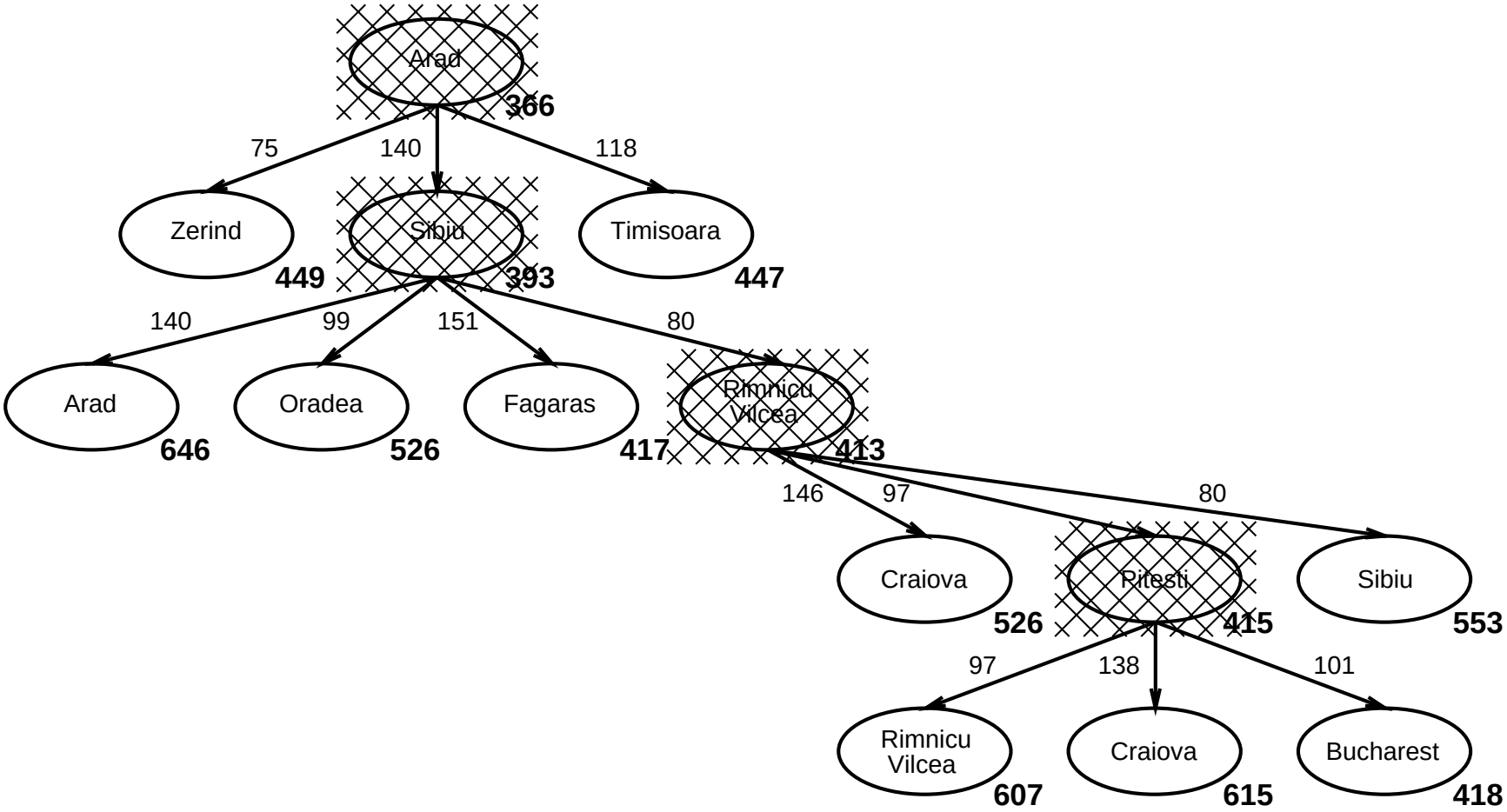
Exemplo de busca A*



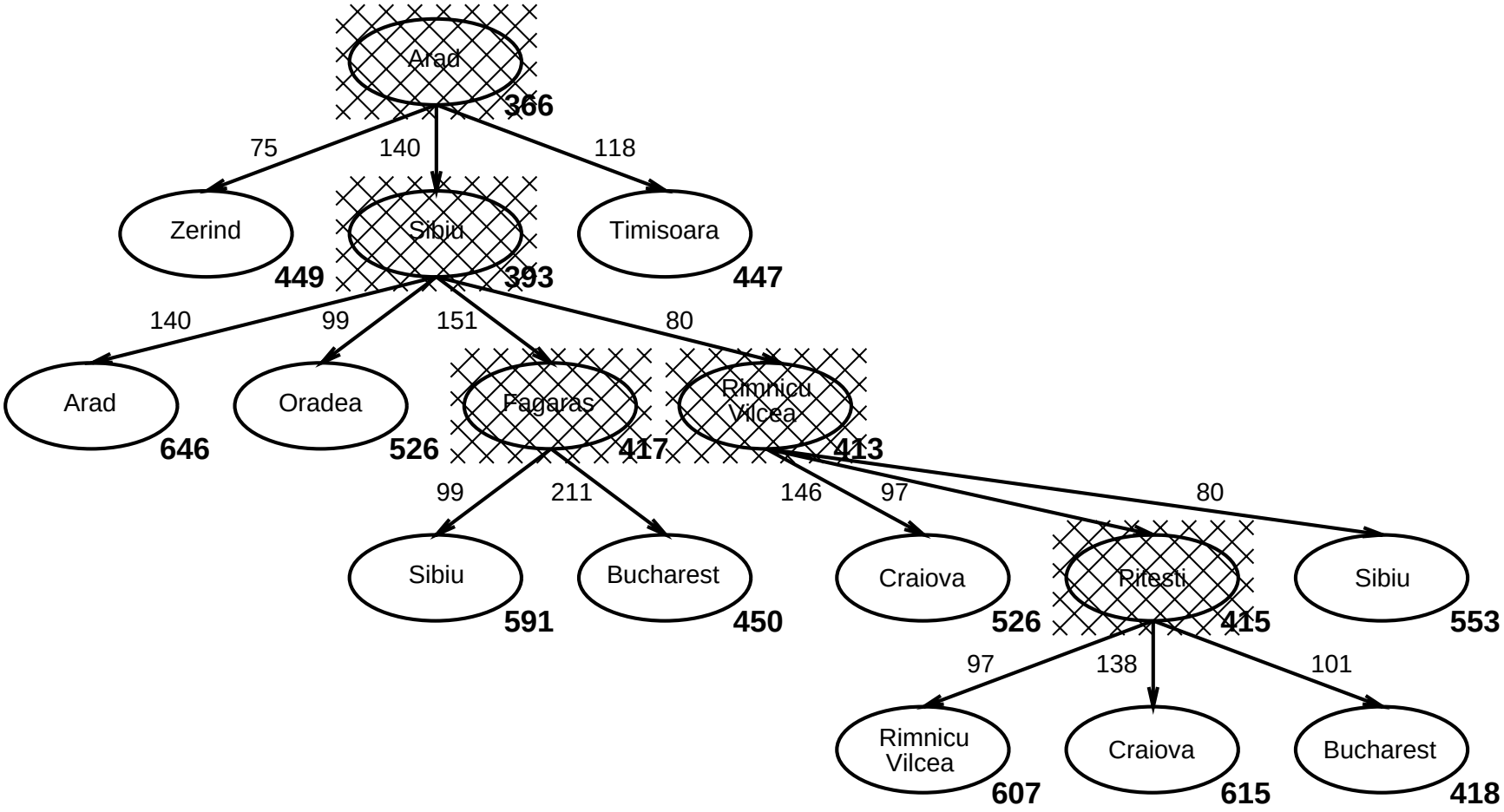
Exemplo de busca A*



Exemplo de busca A*



Exemplo de busca A*



Otimalidade de A*

Se:

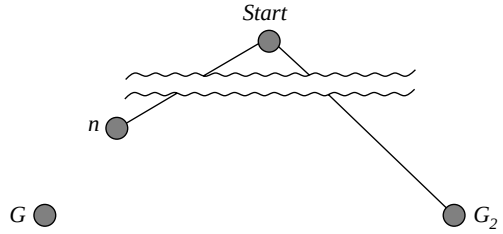
- O fator de ramificação para busca é finito;
- O custo de cada expansão é positivo; e
- A* usa uma heurística *admissível* i.e., $\hat{h}(n) \leq h(n)$; então

Teorema: Busca A* é ótima

Muito bom: em geral, não é muito difícil selecionar uma heurística admissível.
Por exemplo, $\hat{h}_{\mathrm{SLD}}(n)$ nunca superestima a distância real em estradas...

Comportamento de A*

Suponha que um nó sub-ótimo G_2 foi gerado, e está na fila de nós a expandir. Seja n um nó não expandido em um caminho de custo mínimo para um objetivo ótimo G.



$$
\begin{aligned}
\hat{f}(G_2) &= \hat{g}(G_2) \qquad \text{pois } \hat{h}(G_2) = h(G_2) = 0 \\
&> g(G) \qquad \text{pois } G_2 \text{ é subótimo} \\
&= g(n) + h(n) = \hat{g}(n) + h(n) \qquad \text{pois } n \text{ está no caminho ótimo} \\
&\geq \hat{g}(n) + \hat{h}(n) \qquad \text{pois } \hat{h} \text{ é admissível} \\
&= \hat{f}(n)
\end{aligned}
$$

Como $\hat{f}(G_2) > \hat{f}(n)$, A* *nunca* selecionará G_2 para expansão.

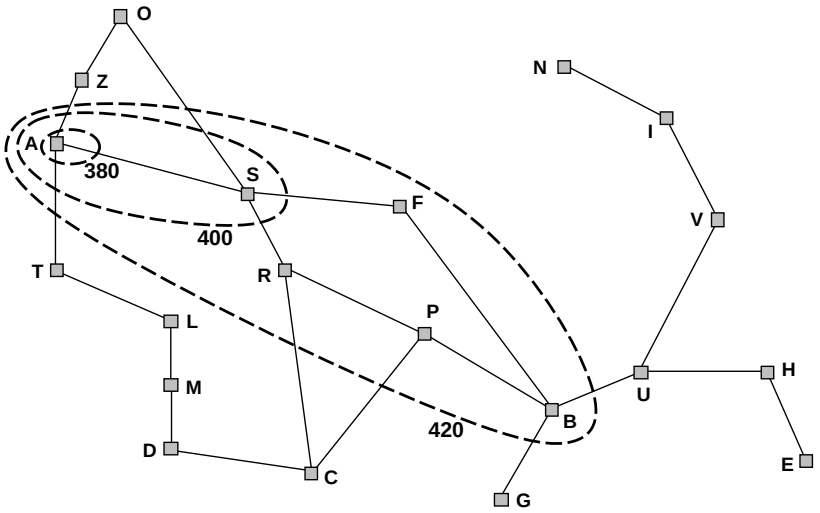
Esta "prova" é só ilustrativa do comportamento de A*: antes, teríamos que mostrar que a) estados-objetivos de fato aparecem na fila; e b) sempre haverá nós provenientes de caminhos mínimos aguardando expansão... Provar otimalidade de A* é um ótimo exercício !!!

Comportamento de A* (mais intuitivo)

Nó inicial n_0: $\hat{f}(n_0) \leq f(n_0)$.
Nó objetivo ótimo G: $\hat{f}(G) = f(G) = f(n_0)$.

Lema: A* expande nós em ordem crescente de valores de $\hat{f}$

Gradualmente soma "contornos-$\hat{f}$" de nós
Contorno i tem todos os nós com $\hat{f} = \hat{f}_i$, onde $\hat{f}_i < \hat{f}_{i+1}$



Isto não quer dizer que um único ramo será continuamente expandido, como em *deph-first*. Mesmo que $\hat{f}$ aumente ao se percorrer o ramo, A* pode ter que abandonar ramos sendo expandidos e reconsiderar outros nós que entraram há mais tempo na pilha (ver exemplo anterior) !!!

Outras Propriedades de A*

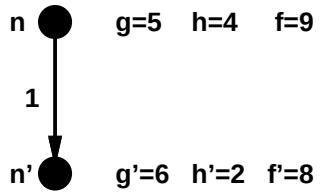
Completa?? Sim, a menos que haja infinitos nós com $f \leq f(G)$

Tempo?? $O(b^m)$, mas uma boa heurística pode levar à uma melhoria dramática.

Espaço?? $O(b^m)$—mantém todos os nós em memória.

Monotonicidade de $\hat{f}$

Para alguma heuristica admissível, $\hat{f}$ pode ***diminuir*** ao longo de um caminho. Por exemplo, suponha que n' seja um sucessor de n:



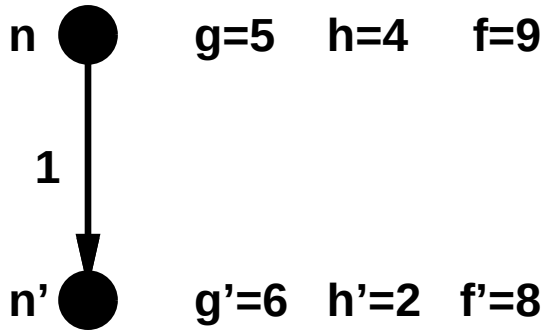
Mas isto joga fora informação!
$\hat{f}(n) = 9 \Rightarrow$ verdadeiro custo de um caminho por n é ≥ 9
Portanto o verdadeiro custo de um caminho por n' também é ≥ 9

Manter $\hat{f}(n)$ não-decrescente ajudaria a aumentar velocidade da busca.

Modificação **Pathmax** para A*:
Em vez de $\hat{f}(n') = \hat{g}(n') + \hat{h}(n')$, usamos $\hat{f}(n') = max(\hat{g}(n') + \hat{h}(n'), \hat{f}(n))$

Com Pathmax, $\hat{f}$ sempre é não-decrescente ao longo de qualquer caminho... Isto diminui complexidade no tempo!!!

Heurísticas Consistentes



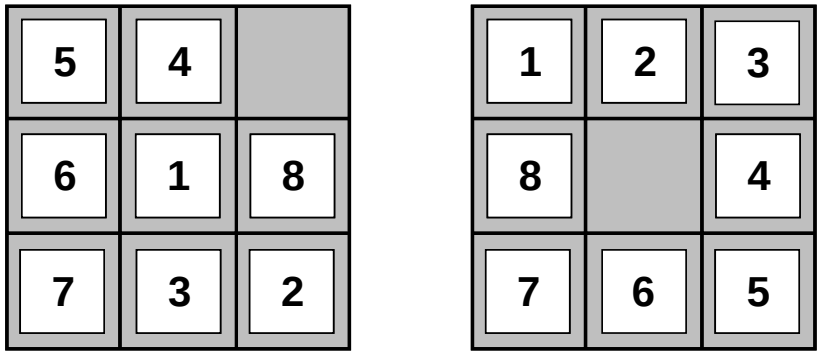
Outra idéia: usar uma heurística que garanta $\hat{h}(n') \geq \hat{h}(n) - c(n, n')$, onde $c(n, n')$ é o custo do ramo (n, n'). Tal heurística é dita $consistente$.

Pode-se provar que isto também garante $\hat{f}$ não-decrescente.

Heurísticas admissíveis

Por exemplo, para o quebra-cabeças da aula anterior:

$h_1(n) =$ número de blocos mal-posicionados
$h_2(n) =$ distância Manhattan total
(i.e., no. de quadrados para a localização desejada de cada bloco)



$h_1(S) =$?? 7
$h_2(S) =$?? 2+3+3+2+4+2+0+2 = 18

Estas heurísticas são consistentes?

Dominância

Se $h_2(n) \geq h_1(n)$ para todo n (ambos admissíveis)
então h_2 **domina** h_1 e é melhor para busca

Custos de busca típicos:

$d = 14$ IDS = 3.473.941 nós
A$^*(h_1) = 539$ nós
A$^*(h_2) = 113$ nós
$d = 14$ IDS = muitos nós
A$^*(h_1) = 39.135$ nós
A$^*(h_2) = 1.641$ nós

Observe o compromisso entre dominância e admissibilidade...

Problemas relaxados

Heuristica admissível pode ser derivada da solução ***exata*** do custo de uma versão ***relaxada*** do problema

Se as regras do quebra-cabeça são relaxadas de forma que um bloco possa mover-se $para\ qualquer\ lugar$, então $h_1(n)$ dá a solução mais curta.

Se as regras são relaxadas de forma que um bloco possa mover-se para $qualquer\ quadrado\ adjacente$, então $h_2(n)$ dá a solução mais curta.

Algoritmos de melhoria iterativa

Em muitos problemas de otimização, a trajetória é $\boldsymbol{irrelevante}$;
o próprio estado-objetivo é a solução

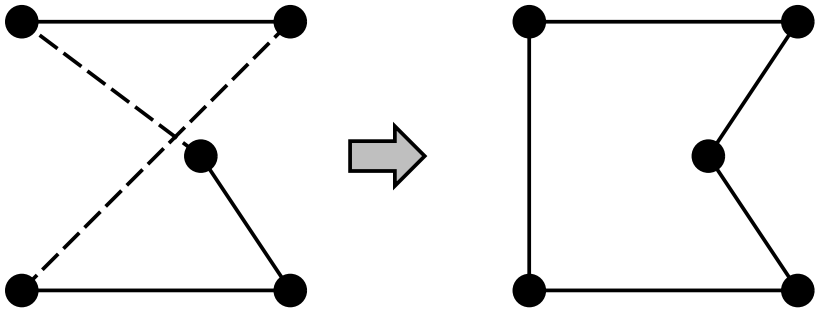
Espaço de estados = conjunto de configurações "completas"
ache configuração $\acute{o}tima$ (e.g. PCV),
ou ache configuração que satisfaz restrições (e.g. prob. das n-rainhas)

Em tais casos, pode-se usar um algoritmo de **melhoria iterativa**;
mantém-se um único estado "atual", tentando-se melhorá-lo

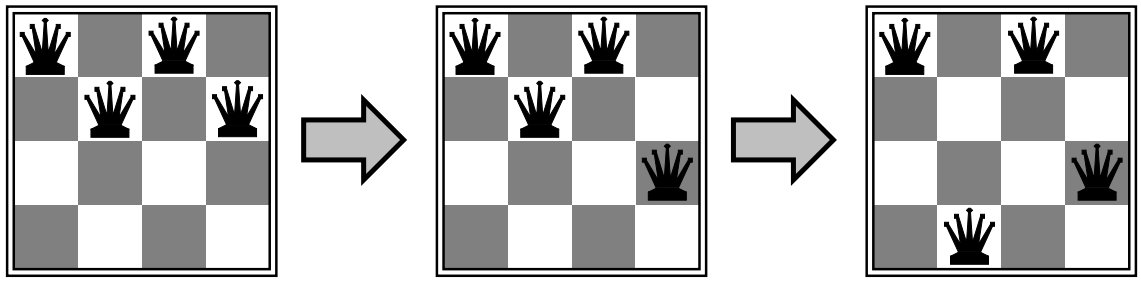
Espaço constante, adequado para busca $online$ ou $offline$

Exemplos

Problema do caixeiro viajante: Ache a menor excursão que visita cada cidade precisamente uma vez



Problema das n-rainhas: Ponha n rainhas em um tabuleiro $n \times n$, sem que duas rainhas fiquem na mesma linha, coluna, ou diagonal



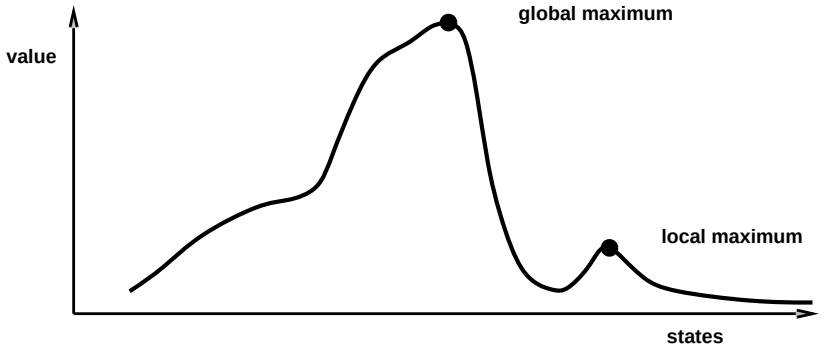
Hill-climbing (ou subida/descida de gradiente)

"É como escalar o Everest com amnésia e em névoa espessa"

function HILL-CLIMBING(*problem*) **returns** a solution state
  inputs: *problem*, a problem
  local variables: *current*, a node
    next, a node

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  loop do
    next ← a highest-valued successor of *current*
    if VALUE[next] < VALUE[current] **then return** *current*
    current ← *next*
  end

Dependendo do estado inicial, pode ficar preso em máximos locais



Simulated annealing

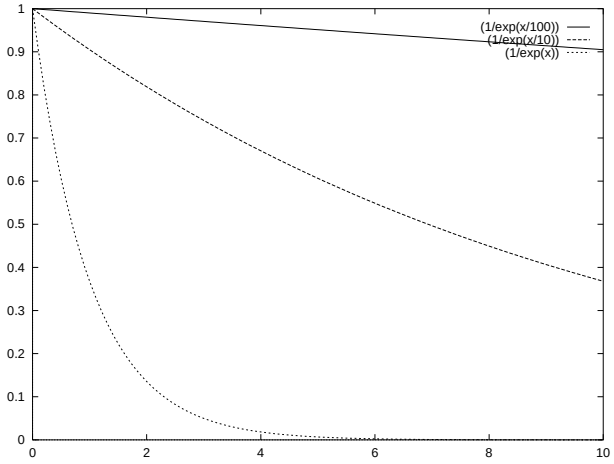
Idéia: fuga de máximos locais permitindo-se alguns movimentos "ruins" mas **gradualmente diminuindo-se o tamanho e frequência destes**

function SIMULATED-ANNEALING(*problem, schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling the probability of downward steps

  current $\leftarrow$ MAKE-NODE(INITIAL-STATE[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow$ *schedule*[t]
    if T=0 **then return** *current*
    next $\leftarrow$ a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** *current* $\leftarrow$ *next*
    else *current* $\leftarrow$ *next* only with probability $e^{\Delta E/T}$

Simulated annealing: Efeito da Temperatura e Propriedades

Observe que $\Delta E < 0$ para os estados "ruins".



A uma "temperatura" fixa T, a probabilidade de ocupação de estados atinge distribuição Boltzman: $p(x) = \alpha e^{\frac{E(x)}{kT}}$

T diminuiu lentamente o bastante $\Longrightarrow$ sempre atinge-se o melhor estado

esta é necessariamente uma garantia interessante??

Inventado por Metropolis em 1953, para modelamento de processos físicos

Extensivamente usado em projetos de VLSI, programação de rotas aéreas, etc.

IDA* e SMA* (CT215)

A*: bom, mas pode ter problemas com memória. Alternativas:

- IDA*: busca *iterative deepening* com limite por estágio definido pelo valor de $\hat{f} = g + \hat{h}$. Completo e ótimo, mas sofre com problemas em que o valor de $\hat{h}$ varia muito com o estado visitado.
- SMA* (Simplified Memory-Bounded A*): variação de IDA* que usa uma memória finita para evitar repetição de caminhos. Completo e ótimo, desde que a memória seja suficiente para armazenar todos os nós do caminho ótimo.

Mais detalhes (incluindo uma ilustração do funcionamento de SMA*: AIMA, págs. 106-111.

CT215: Um aluno sorteado para apresentar SMA*: trs transparências, próxima aula.