Resolvendo problemas com busca

Inteligência Artificial

Aydano Pamponet Machado

Universidade Federal de Alagoas - UFAL
Instituto de Computação – IC
aydano.machado@ic.ufal.br

As estratégias de ***busca exaustiva*** ...

- As estratégias de ***busca exaustiva*** ...

  Encontram soluções para problemas pela geração *sistemática* de novos estados, que são comparados ao objetivo
- ... são *ineficientes* na maioria dos casos
  - São capazes de calcular *apenas* o *custo de caminho* do nó atual ao nó inicial (função *g*), para decidir qual o próximo nó da fronteira a ser expandido ...
  - mas não necessariamente conduz a busca na direção do objetivo (olha só para o passado)

Exemplo: barco perdido

Não se considera o oceano inteiro: correntes marítimas, vento, etc...

Busca heurística (ou informada)

- Estratégias de ***busca heurística*** utilizam *conhecimento específico* do problema na escolha do próximo nó a ser expandido e aplicam uma ***função de avaliação*** a cada nó na fronteira do espaço de estados
  - Essa função estima o *custo de caminho* do nó atual ao objetivo mais próximo utilizando uma ***função heurística***
  - Qual dos nós supostamente é o mais próximo do objetivo?

Busca com informação e exploração

- **Busca heurística (ou informada)**
  - Estima qual o melhor nó da fronteira a ser expandido com base em ***funções heurísticas*** $\Rightarrow$ **conhecimento**
  - **Estratégia de busca**:
    - **Melhor escolha** (*best-first search*),
      - Busca gulosa
      - A*
    - **Busca com limite de memória**
    - **Busca com melhora iterativa**
  - **Direção de busca**: idem à busca cega

Busca heurística (ou informada)

- **Função heurística *h(n)***
  - **Estima** o custo do caminho entre o nó n e o objetivo
  - Depende o problema

  Exemplo: Encontrar a rota mais barata entre Arad e Bucharest

  $h_{dd}(n)$ = distância direta entre o nó *n* e o nó final.
- **Como escolher uma boa função heurística?**
  - Deve ser *admissível* i.e., nunca *superestimar* o custo real da solução
  - Distância direta (h_{dd}) é *admissível* porque o caminho mais curto entre dois pontos é sempre uma linha reta

Busca heurística (ou informada)

- **Busca pela melhor escolha**
  - Busca genérica onde o nó de menor custo “aparente” na fronteira do espaço de estados é expandido primeiro
  - Duas abordagens básicas:
    - Busca Gulosa (*Greedy search*)
    - Algoritmo A*

Busca Gulosa

- Semelhante à **busca em profundidade com *backtracking***

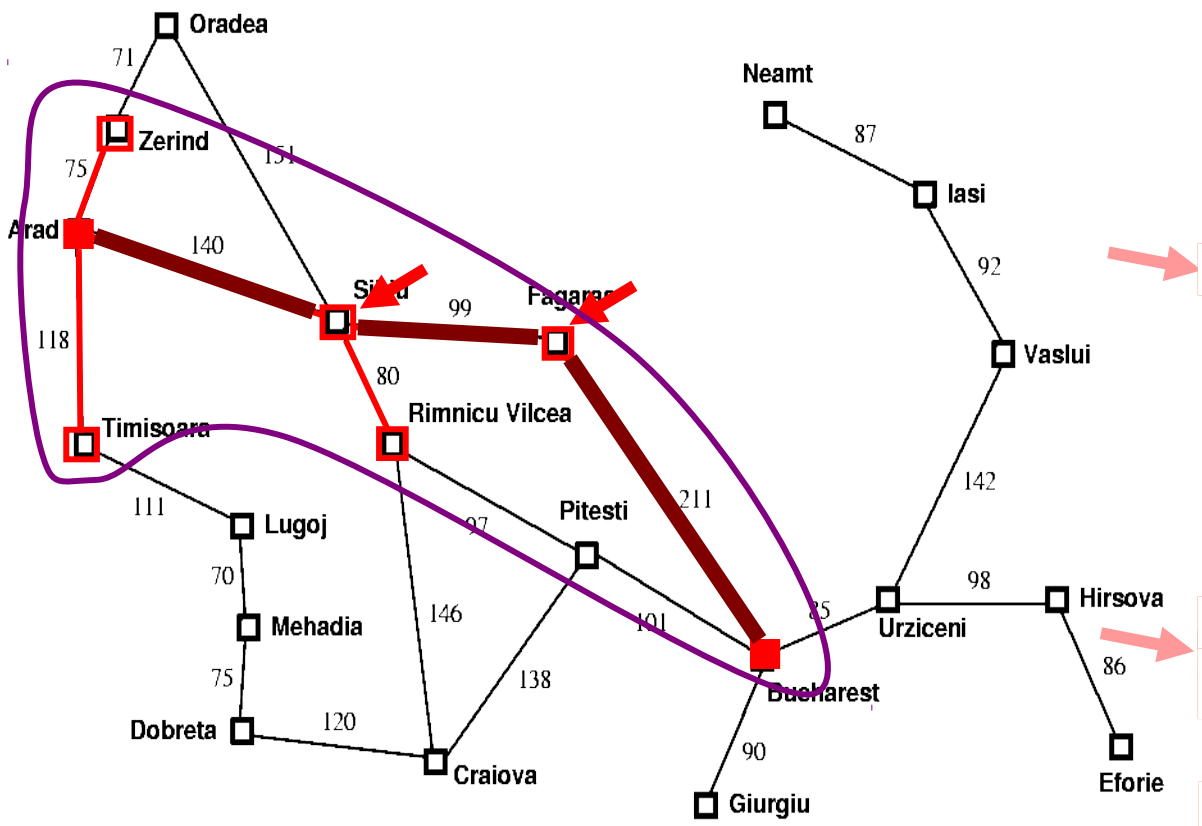
  Tenta expandir o nó mais próximo ao nó final com base na estimativa feita pela função heurística *h*.

- **Custo de busca** é minimizado

  Não expande nós fora do caminho

- Escolhe o caminho mais econômico à primeira vista

Busca Gulosa



Distância em linha reta para 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 (considerações finais)

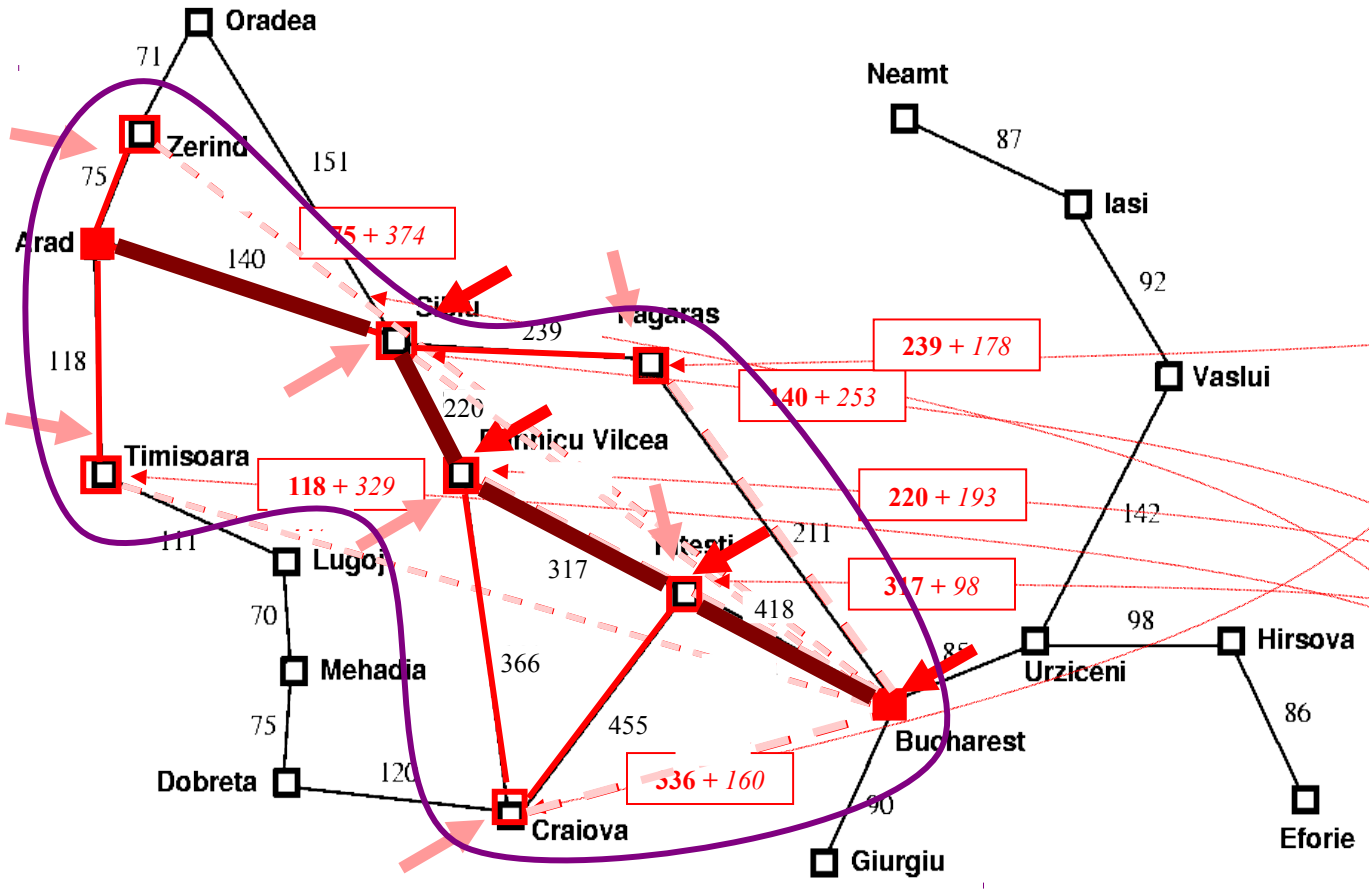
- Não é ótima... (semelhante à busca em profundidade)
  - Só olha para o futuro!
- ... nem é completa
  - Pode entrar em “loop” se não detectar a expansão de estados repetidos
  - Pode tentar desenvolver um caminho infinito
- Custo de tempo e memória: $O(b^d)$
  - Guarda todos os nós expandidos na memória

Algoritmo A*

- Tenta minimizar o custo total da solução combinando:
  - **Busca Gulosa**: Econômica, porém não é completa nem ótima
  - **Busca de Custo Uniforme**: Ineficiente, porém completa e ótima
- **Função de avaliação: f(n) = g(n) + h(n)**
  - g(n) = distância de n ao nó inicial e
  - h(n) = distância estimada de n ao nó final
- A* expande o nó de menor valor de f na fronteira do espaço de estados
  - Olha o futuro sem esquecer do passado!
  - Se h é admissível, f(n) nunca irá superestimar o custo real da melhor solução através de n

    (Neste caso, pode-se encontrar a rota de fato mais curta entre Arad e Bucarest)

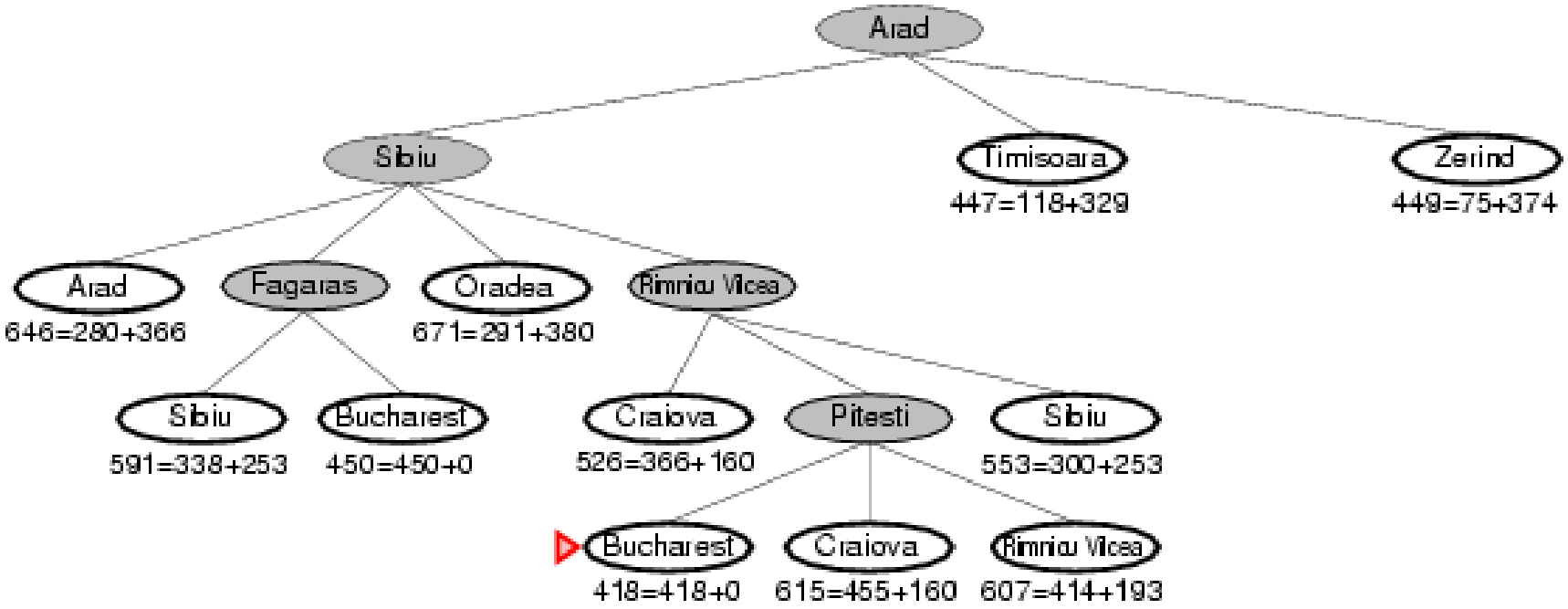
Algoritmo A*



Distância em linha reta para 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

Algoritmo A*


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

Algoritmo A*

- **O algoritmo é monotônico**
  - O custo de cada nó gerado no mesmo caminho nunca diminui
  - Mas pode haver problemas: $f(n') < f(n)$, onde n é o pai de n'

    $f(n) = g(n) + h(n) = 3 + 4 = 7$

    $f(n') = g(n') + h(n') = 4 + 2 = 6$
- Para se garantir a monotonicidade de f:
  - $f(n') = \max(f(n), g(n') + h(n'))$

    ..., uma vez que todo caminho que passa por n' passa também por n' (seu pai),
- Semelhante à busca em largura

  ... mas ao invés de geração (profundidade) da árvore, o que conta é o contorno

  $f = g + h$

Algoritmo A* (considerações finais)

- É *completa* e *ótima*

  Como a busca em largura, ...
- **Custo de memória**: $O(b^d)$

  ... guarda todos os nós expandidos na memória, o que é um problema bem mais grave do que o tempo de busca
- É otimamente eficiente

  Não existe algoritmos expandindo menos nós com a mesma f
- **Custo de tempo**:

  Exponencial com o comprimento da solução, porém boas funções heurísticas diminuem significativamente esse custo

  $|h(n) - h^*(n)| \leq O(\log h^*(n))$ onde h^* custo real

Métodos baseados em limitação de memória

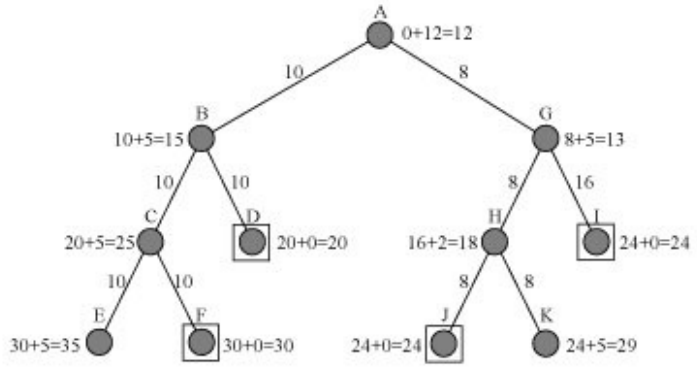
- **IDA*** (Iterative Deepening A*)
  - igual ao aprofundamento iterativo, porém com limite na função de avaliação (f) no lugar da profundidade (d).
  - necessita menos memória do que A*
- **SMA*** (Simplified Memory-Bounded A*)
  - O número de nós guardados em memória é fixado previamente

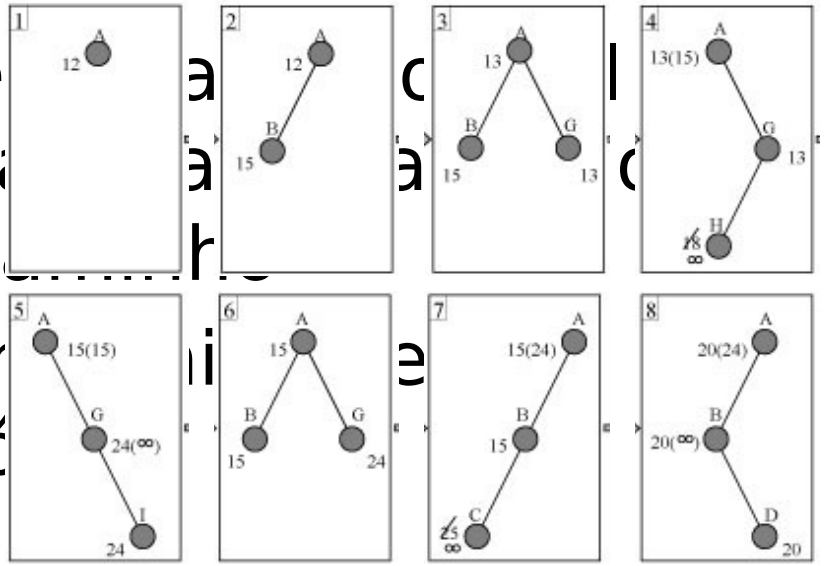
IDA* (Iterative Deepening A*)

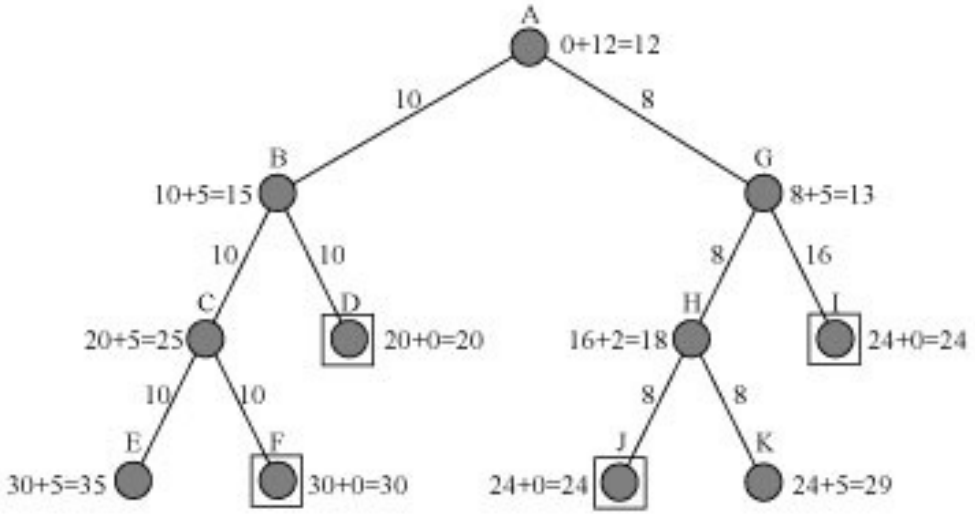
- Se a busca com profundidade iterativa é interessante, por que não incorporá-la no A*?
- Da mesma forma que o A*, IDA* é *completa* e *ótima*
- **Complexidade de espaço**: bf^*/δ, onde f^* é o custo da solução ótima e δ é o menor custo de operador.
- **Complexidade de tempo**: depende fortemente do número de diferentes valores que a função heurística pode assumir
- **Problema**: só funciona bem para problemas simples

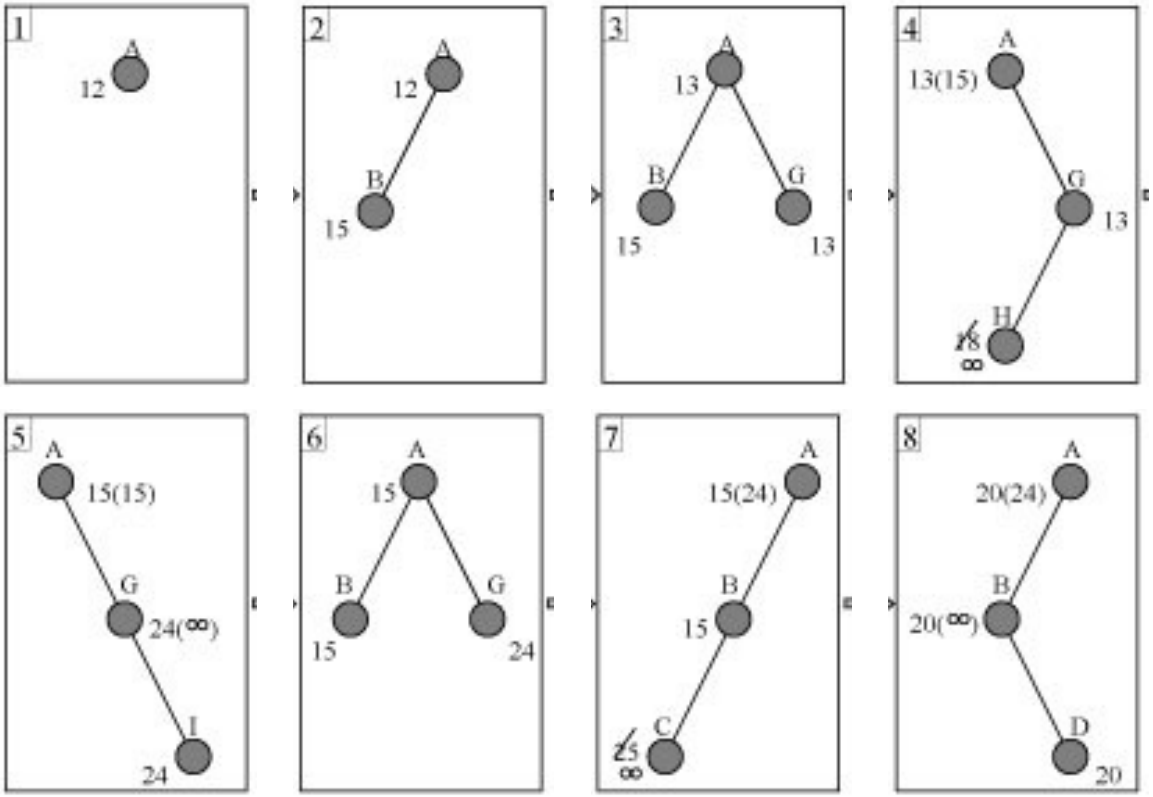
SMA* (Simplified Memory-Bounded A*)

- Características
  - Usa toda a memória
  - Evita estados repeti
    memória permite
  - É completo se a me
    suficiente para arma
    solução de menor c
  - É ótimo se o melhor
    disponível na memó






A
0+12=12
10
8
B
10+5=15
G
8+5=13
10
10
8
16
C
20+5=25
D
20+0=20
H
16+2=18
I
24+0=24
10
10
8
8
E
30+5=35
F
30+0=30
J
24+0=24
K
24+5=29

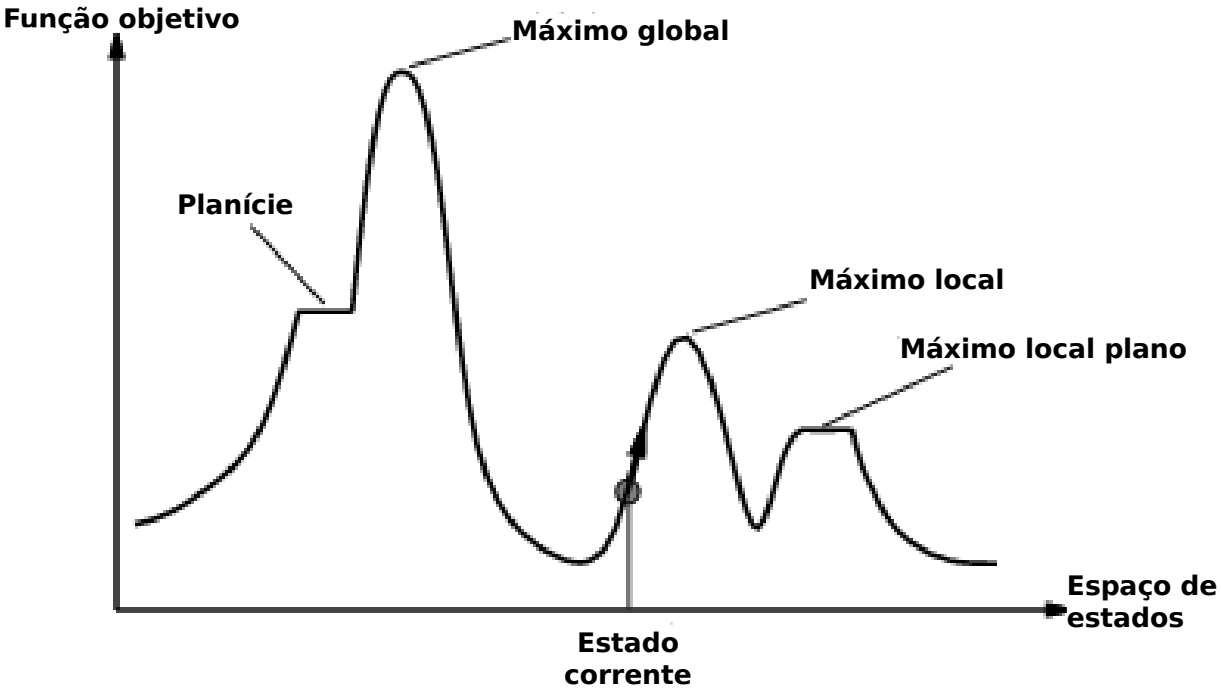

1
A
12
2
A
12
B
15
3
A
13
B
15
G
13
4
A
13(15)
G
13
H
8
5
A
15(15)
G
24(∞)
I
24
6
A
15
B
15
G
24
7
A
15(24)
B
15
C
25
8
A
20(24)
B
20(∞)
D
20

Busca com melhora interativa

- A idéia é começar com o estado inicial (= configuração completa, solução aceitável), e melhorá-lo iterativamente.
- Os estados estão representados sobre uma superfície
  - A altura de qualquer ponto na superfície corresponde à função de avaliação do estado naquele ponto
- O algoritmo se “move” pela superfície em busca de pontos mais altos/baixos (objetivos)
  - O ponto mais alto (máximo global) corresponde à solução ótima
    - Nó onde a função de avaliação atinge seu valor máximo
- Aplicações: problemas de otimização (linha de montagem, ...)

Busca com melhora interativa

- Topologia de espaço de estados unidimensional, no qual a elevação corresponde à função objetivo
- O objetivo é encontrar o máximo global



Busca com melhora interativa

- Guardam apenas o estado atual e não vêem além dos vizinhos imediatos do estado
- Contudo, muitas vezes são os melhores métodos para tratar problemas reais muito complexos

- Duas classes de algoritmos:
  - **Subida da Encosta ou Gradiente ascendente (Hill-Climbing)**

    Só faz modificações que melhoram o estado atual
  - **Têmpera Simulada (Simulated Annealing)**

    Pode fazer modificações que pioram o estado temporariamente para possivelmente melhorá-lo no futuro

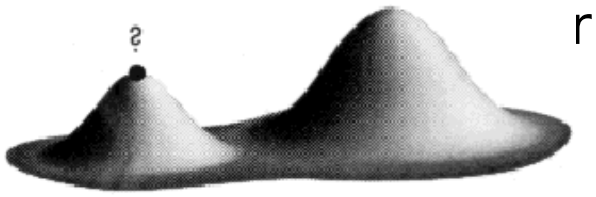
Subida da Encosta

- O algoritmo não mantém uma árvore de busca:
  - Guarda apenas o estado atual e sua avaliação
  - É simplesmente um “loop” que fica se movendo na direção que está crescendo
- Quando existe mais de um “melhor” sucessor para o nó atual, o algoritmo faz uma escolha aleatória
- Isso pode acarretar em 3 problemas, que podem levar ao desvio do caminho à solução:
  - Máximos locais
  - Planícies
  - Encostas

Subida da Encosta

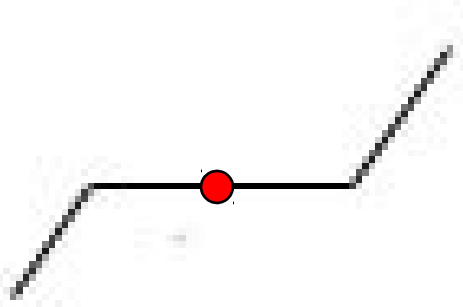
- **Problemas a serem vencidos**
  - **Máximos locais**
    - Em contraste com *máximos globais*, são picos mais baixos do que o pico mais alto no espaço de estados (solução ótima)
    - A função de avaliação leva a um valor máximo para o caminho sendo percorrido: essa função utiliza informação "local"
    - Porém, o nó final pode estar em outro ponto mais "alto"
    - Isto é uma conseqüência das decisões irrevogáveis do método

Exemplo: no Xadrez, eliminar a Rainha do adversário pode le r o jogo



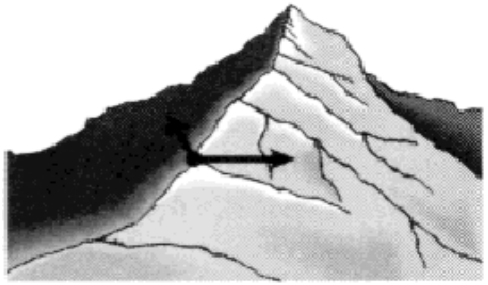
Subida da Encosta

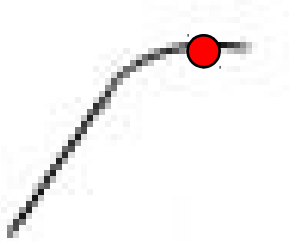
- **Problemas a serem vencidos**
  - **Planícies**
    - Uma região do espaço de estados onde a função de avaliação dá o mesmo resultado ($f(n) = f(\text{filhos}(n))$)
    - O algoritmo pára depois de algumas tentativas



Subida da Encosta

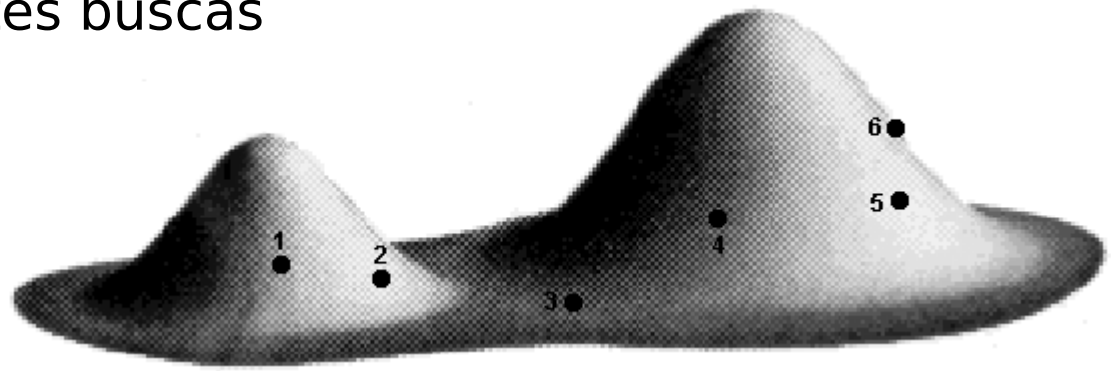
- **Problemas a serem vencidos**
  - **Encostas**
    - Encostas podem ter lados muito íngremes e o algoritmo chega ao topo com facilidade
    - Mas o topo pode se mover em direção ao pico vagarosamente
    - Na ausência de operadores que alterem significativamente o topo, o algoritmo oscila entre dois pontos sem fazer grande progresso





Subida da Encosta com Reinício Aleatório

- Nos casos anteriores, o algoritmo padrão pode chegar a um ponto onde não faz progresso
- O algoritmo com *reinício aleatório* realiza uma série de buscas a partir de estados iniciais gerados aleatoriamente
- Cada busca é executada ...
  - até que um número máximo de iterações estipulado seja atingido, ou
  - até que os resultados encontrados não apresentem melhora significativa
- O algoritmo escolhe o melhor resultado obtido com as diferentes buscas



Subida da Encosta com Reinício Aleatório

- Pode chegar a uma solução ótima quando iterações suficientes forem permitidas
- O sucesso deste método depende muito do formato da superfície do espaço de estados:
  - Se há poucos máximos locais, o reinicio aleatório encontra uma boa solução rapidamente
  - Porém, para problemas NP-completos, o custo de tempo é exponencial

Subida da Encosta

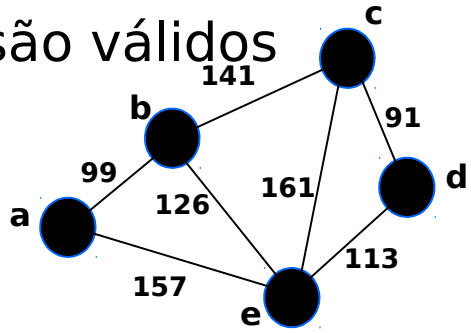
- Algoritmo

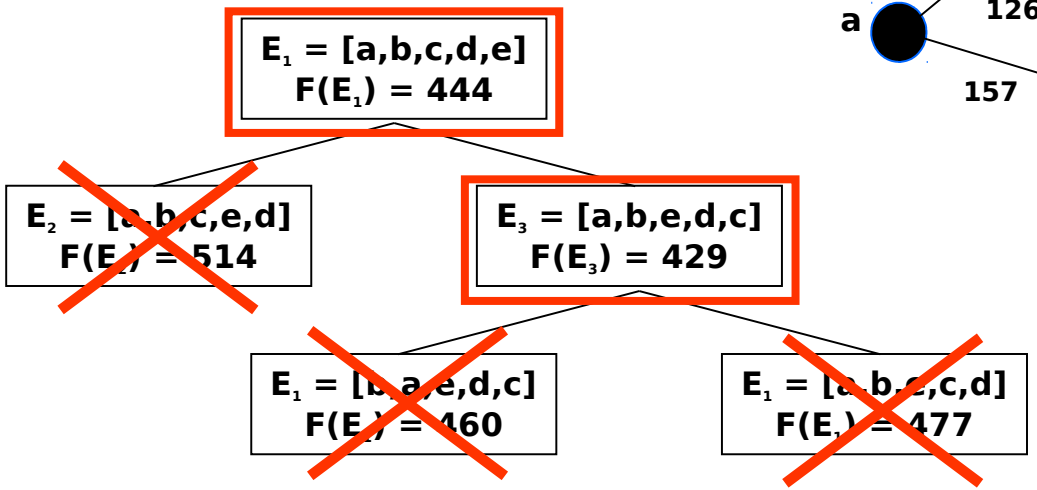
```
função SUBIDA-DA-ENCOSTA(problema) retorna uma solução
    Variáveis locais:
                - corrente, o nó atual
                - vizinho, o próximo nó

    corrente ← CRIAR-NÓ(ESTADO-INICIAL[problema])
    repita
                vizinho ← SUCESSOR-DE-MAIOR-VALOR(corrente)
                Se valor[vizinho] < valor[corrente]
                então retornar ESTADO[corrente]
                corrente ← vizinho
```

Subida da Encosta

- **Exemplo**: cálculo da menor rota com 5 nós
  - **Estado inicial**: [a,b,c,d,e]
  - ***F***: soma das distâncias diretas entre cada nó, na ordem escolhida (admissível)
  - **Operadores**: permutar dois nós quaisquer do caminho
  - **Restrição**: somente caminhos conectados são válidos
  - Estado final: nó onde valor de ***f*** é mínimo





Têmpera simulada

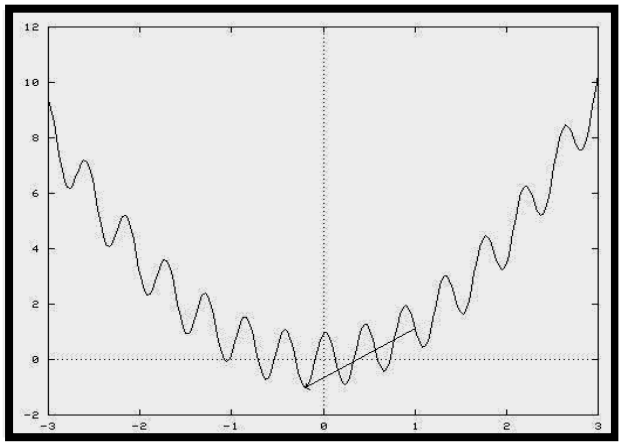
- Analogia com resfriamento/recozimento de vidros ou metais
  - Processo de resfriar um líquido gradualmente até ele se solidificar
- O algoritmo utiliza um **mapeamento de resfriamento** de instantes de tempo (t) em temperaturas (T)
- Semelhante à Subida da Encosta, porém com meios para se escapar de máximos locais
  - Quando a busca fica “preso” num máximo local, o algoritmo não reinicia a busca aleatoriamente
  - Retrocede para escapar desse máximo local

Têmpera simulada

- Nas iterações iniciais, não escolhe necessariamente o "melhor" passo, mas sim um movimento aleatório:
  - Se a situação melhorar, esse movimento será sempre escolhido posteriormente
  - caso contrário, associa a esse movimento uma probabilidade de escolha menor do que 1.
- Essa probabilidade depende de dois parâmetros, e decresce exponencialmente com a piora causada pelo movimento, $e^{\Delta E/T}$, onde:

  $\Delta \mathbf{E}$ = *Valor*[*próximo-nó*] - *Valor*[*nó*

  T = *Temperatura*



Têmpera simulada

- Com o tempo, passa a funcionar como Subida da Encosta
- O algoritmo é ótimo e completo se o mapeamento de resfriamento tiver muitas entradas com variações suaves
  - Isto é, se o mapeamento diminui T suficientemente devagar no tempo, o algoritmo vai encontrar um máximo global ótimo

Algoritmos Genéticos

- Estados sucessores são gerados a partir de dois estados, em vez de gerados pela modificação de um único estado
- Analogia com a reprodução sexuada e a seleção natural

(a) População: conjunto de *k* estados (*indivíduos*) gerados aleatoriamente

(b) Função de *fitness*: retorna a avaliação do estado

(c) Seleção: Pares são escolhidos aleatoriamente para reprodução, de acordo com as probabilidades definidas pela função de *fitness*

(d) Cruzamento (*crossover*): escolhe-se ao acaso um ponto de *crossover*, gerando-se os descendentes

(e) Mutação: Cada posição está sujeita a uma mutação aleatória