Métodos de Busca Informada Part III

Busca limitada pela memória (Memory Bounded Search)

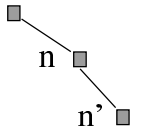
- Apesar da existência de algoritmos de busca informada, alguns problemas são realmente difíceis por natureza
- Dois algoritmos designados a economizar memória.
  - IDA* extensão da Busca em Profundidade Iterativa (IDS) que usa informação heurística
  - SMA*, é similar ao A*, mas restringe o tamanho da lista de nós para caber na memória disponível.

3

A*

- A função de avaliação deve ser monotônica!
- Funções com heurísticas admissíveis, em geral, são monotônicas
- Pode existir uma heurística não monotônica: *f(n)* decresce. Para eliminar problemas desse tipo devemos usar o custo do pai ao invés do calculado para o seu sucessor, ou seja:

  $f(n') = max(f(n), g(n') + h(n'))$ (equação do ***pathmax***)



g(n) = 3, h(n) = 4, f(n) = 7

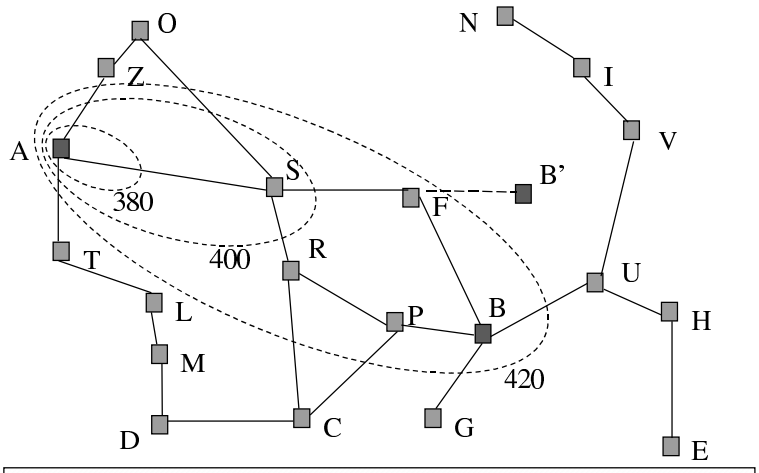
g(n') = 4, h(n') = 2, f(n') = 6

4

Contorno no espaço de estados

- Garantindo que *f* nunca decresce (monotonicidade), podemos desenhar **contornos** no espaço de busca

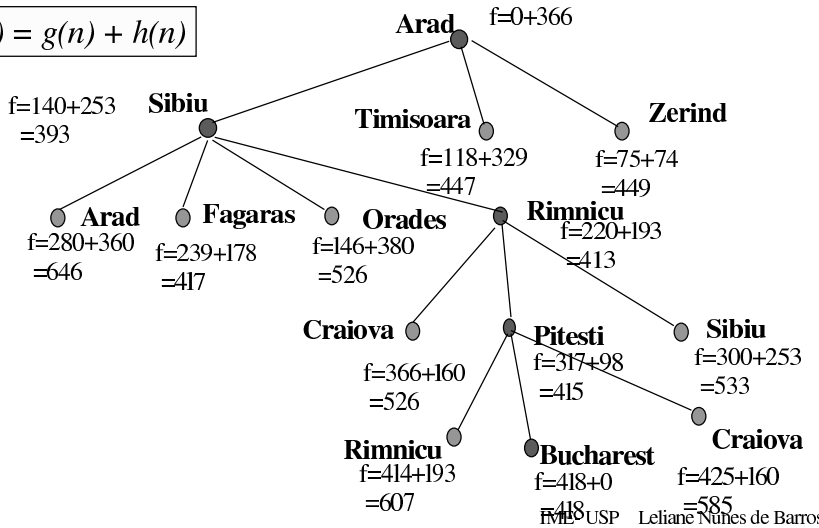
Contornos no espaço de estados



Note como os contornos se "esticam" em direção ao gol.

Espaço de busca

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



7

A*

- Para f^* sendo o custo do caminho da solução ótima, A* expande todos os nós com $f(n) < f^*$.
- A* pode expandir alguns nós dentro do contorno do goal antes de selecionar o estado goal ==> a primeira solução encontrada é a ótima
- nenhum outro algoritmo ótimo garante expandir menos nós do que A*

8

Busca em Profundidade Iterativa A* (IDA*)

- IDS reduz requisitos de memória.
- Em IDA* cada iteração é DFS, como na busca IDS.
- Ao invés de adotar valores crescentes de profundidade limite, IDA* adota valores de função de avaliação limite.
- a cada iteração expande-se todos os nós dentro de um **contorno** no espaço de estados.

9

O algoritmo IDA*

Função IDA* (*problema*) **devolve** *a sequência da solução*

Entradas: *problema*

Variáveis locais: *f-limite* (limite atual de *f*)

raiz (um nó)

raiz ⟵ Faça-nó (Estado-Inicial(*problema*))

f-limite ⟵ *f(raiz)*

laço faça

solução, *f-limite* ⟵ DFS-contorno(*raiz*, *f-limite*)

Se *solução* não nula **então devolve** *solução*

Se *f-limite* = ∞ **então devolve** falhou; fim

10

Função DFS-contorno

Função DFS-contorno(*nó*, *f-limite*) **devolve** *a solução e um novo f-limite*

Entradas: *nó, f-limite* (limite corrente)

Variáveis locais: *next-f* (próximo valor de *f-limite*, inicialmente ∞

Se *f[nó]* > *f-limite* **então devolve** nulo, *f(nó)* */*mudou de contorno*/*

Se Testa-goal(*nó*) **então devolve** *nó*, *f-limite* */*achou colução */*

Para cada nó *s* **em** Sucessores(*nó*) **faça**

solução, *new-f* ⟵ DFS-contorno(*s*, *f-limite*)

Se *solução* não nula **então devolve** *solução, f-limite*

Senão *next-f* ⟵ Min(*new-f, next-f*); **fim**

devolve nulo, *next-f* */*busca em contorno maior*/*

11

IDA*

- completo e ótimo como A*
- por usar DFS, requer espaço proporcional ao caminho mais longo a ser explorado
- complexidade de tempo depende do cálculo da heurística
- Tipicamente *f* cresce 2 ou 3 vezes
- IDA* guarda menos nós que A* e encontra a solução ótima mais rapidamente (não usa lista de prioridades)

12

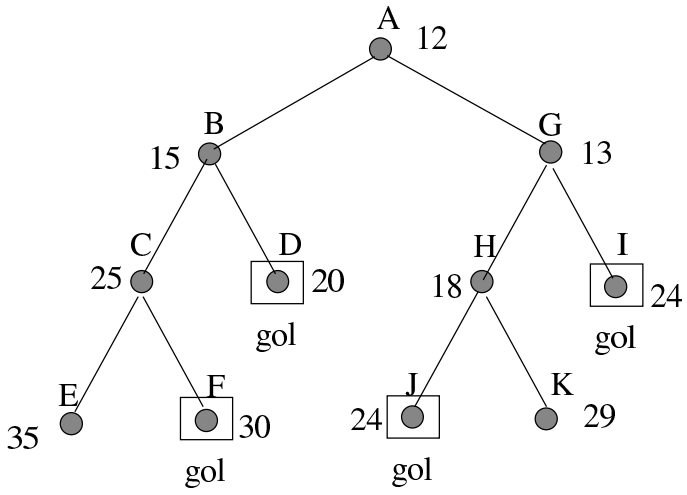
SMA* - **S**implified **M**emory-**B**ounded **A***

- somente usa a memória disponível para busca (quanto mais memória disponível, melhor a eficiência)
- evita explorar nós repetidos
- é completo se houver memória para guardar o caminho da solução de menor profundidade
- é otimo se houver memória para guardar o caminho da solução ótima
- melhor desempenho que A* (se existir suficiente memória disponível)

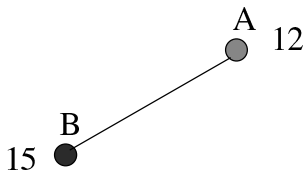
SMA* - estratégia

- para gerar um sucessor sem existir espaço na memória: joga fora nós com alto *f-cost* (*nós esquecidos*)
- guarda nos nós ancestrais informação sobre a qualidade do melhor caminho da sub-árvore esquecida
- somente regenera a sub-árvore quando todos os outros caminhos se mostraram piores que o caminho esquecido

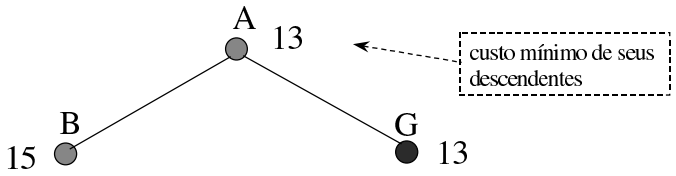
SMA* - com memória de três nós



SMA* - progresso

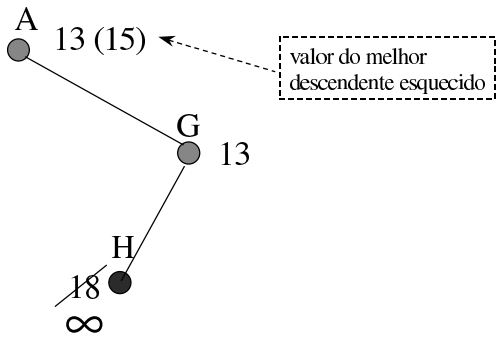


SMA* - progresso



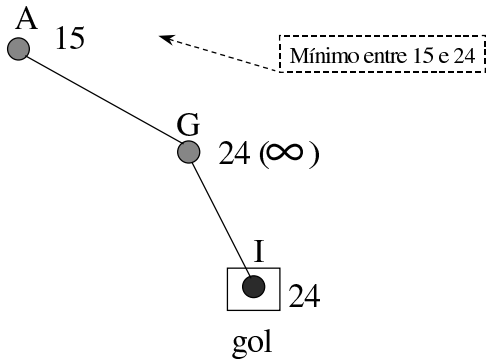
17

SMA* - progresso

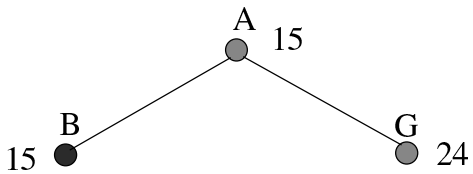


18

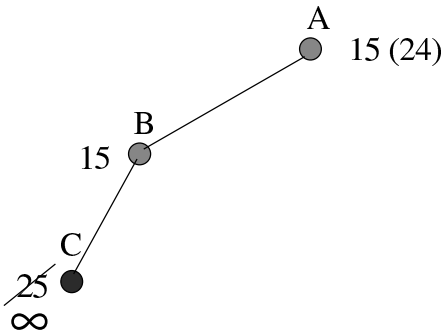
SMA* - progresso



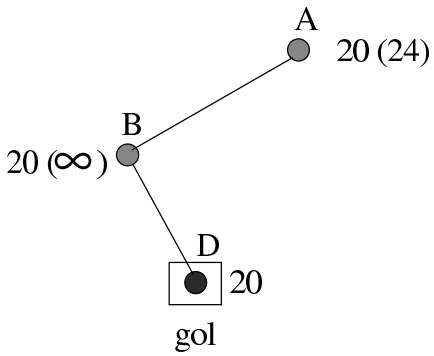
SMA* - progresso



SMA* - progresso



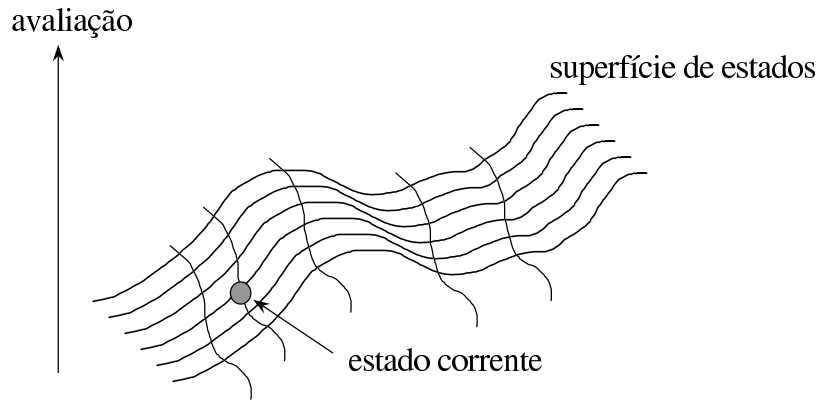
SMA* - progresso



Algoritmos de melhora iterativa

- somente quando o estado do mundo contém toda a informação necessária para a solução (exemplo: n-rainhas)
- busca:
  - começa com uma configuração completa
  - realiza modificações para melhorar a qualidade da solução

Algoritmos de melhora iterativa



topologia geográfica

Algoritmos de melhora iterativa (Iterative Improvement Methods - IIM)

- Guardam somente um estado na memória
- Caminham sobre uma superfície topológica de estados tentando encontrar o pico mais alto, onde estão as soluções ótimas
- Só olham na vizinhança do estado corrente
- Podem resolver problemas difíceis e reais
- Dois tipos principais:
  - ***Hill climbing***
  - ***Simulated annealing***

Hill-climbing search

- inicia com uma descrição completa do estado e faz modificações para melhorar sua qualidade
- move continuamente em direção a estados de maior valor. Maximiza a função (*Valor*) que avalia melhores estados
- não mantém a árvore de busca ==> nós contém informações sobre o estado e sua avaliação (*Valor*)
- não faz *backtracking*

Hill-climbing: algoritmo

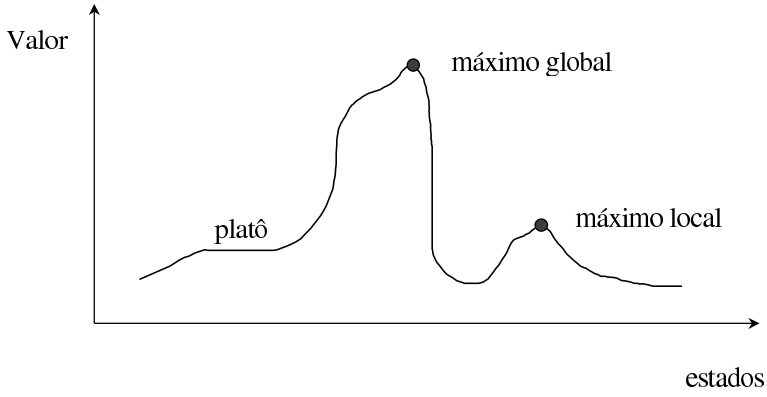
Função Hill-Climbing (*problema*) **devolve** *um estado solução*
  Entradas: *problema*
  Variáveis locais: *estado-corrente*
    próximo-estado
  estado-corrente ← Faça-nó (Estado-Inicial[*problema*])
  laço faça
    próximo-estado ← um sucessor de maior valor que o *estado-corrente*
    if Valor[*próximo-estado*] < Valor[*estado-corrente*]
      então devolve *estado-corrente*
    estado-corrente ← *próximo-estado*
  end

Hill-climbing

- Best First:
  - expande nós com *f-custo* mínimo
- Hill-climbing
  - expande nós sucessores do último nó expandido com *Valor* máximo, isto é, uma vez que foi decidido expandir um nó, somente considere seus sucessores e nunca tente um caminho alternativo.

Hill-climbing

Problema: dependendo do estado inicial: HC pode parar num máximo local



Random-restart Hill-climbing

- execute uma série de buscas hill-climbing a partir de um estado inicial aleatório
- guarda o melhor resultado
- pode-se usar um número fixo de iterações ou continuar até não conseguir superar o melhor resultado guardado com um certo número de iterações
- sucesso depende da forma da superfície (caminho da solução for monotônico)

Simulated annealing

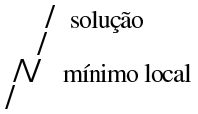
- evita os problemas de Hill-climbing
- ao invés de começar de novo permite que a busca deça o "morro" alguns paços (piora para depois melhorar)
- pulo no espaço de estados não totalmente aleatório
- decresce gradualmente o tamanho do pulo e a frequência até alcançar o melhor estado
- abordagem probabilística para se atingir um ponto de equilíbrio em um sistema:
  - *Annealing*: processo de resfriamento de metais (envolve Temperatura, Tempo, Energia).

Simulated annealing

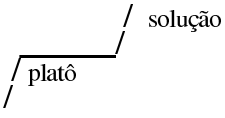
- T: temperatura
- Valor: energia total dos átomos no material
- Escala: relaciona temperatura e tempo (razão com que a temperatura decresce)
- Probabilidade de pular para um estado pior decresce com a temperatura:
  - Distribuição de Boltzman: $p(x) = \alpha e^{E(x)/kT}$
- Dois fatores para controlar a exploração:
  - divergência: *o quanto pior é o próximo-estado?*
  - cobertura: *quanto tempo já foi gasto na busca?*

Hill-climbing: problemas

- **Máximo local**: em oposição ao máximo global



- **Platô**: área do espaço de estado onde a função de avaliação é constante (busca aleatória)



Aplicação de IIM em CSP
(Métodos de Reparo Heurístico)

- estado inicial: atribuir valores para todas as variáveis
- operador de modificação: modificar o valor de uma variável
- heurística: seleciona estados com o número **mínimo de conflitos** (resolve 1M-rainhas com menos de 50 passos)
- resolve grandes problemas em pouco tempo (escalonamento de fábrica, problemas grandes de otimização)
- Telescópio espacial Hubble - escalonamento de uma semana de observações: de 3 semanas para 10 min