Métodos de Busca

Maria Luize Pinheiro e Jerusa Marchi

Universidade Federal de Santa Catarina

Introdução

*In which we see how an agent can find a sequence of actions that achieves its goals, when no single action **will** do (p.59).*

- Agente de resolução de problemas.
  - Age por meio de descoberta de sequência de ações que o levam para o estado final desejado.
- Definição de problema:
  - Estado inicial
  - Função Sucessora: Possíveis ações disponíveis ao agente.
  - Espaço de Estados
  - Caminho
  - Teste de meta (*goal test*)
  - Custo de caminho
  - Custo de passo
  - Solução ótima

Tipos de Problemas

- *Toy Problem*: Utilizado para exercitar diversos métodos de resolução de problemas, com descrição concisa e exata.
  - Sliding-block puzzles (NP-completos)
  - Problema das 8 rainhas
- *Real-World Problem*
  - Problema de localização de rotas
  - Circuito Hamiltoniano (problema do caixeiro viajante) (NP-difícil)
  - VLSI layout
  - Navegação de agentes (robôs)
  - Sequenciamento automático de montagem
  - *Protein design*
  - Pesquisa na internet

Busca em Espaço de Estados

Árvore de Busca

- Gerada pelo estado inicial e a função de sucessão (espaço de estados).
- Formado por nodos. Estes são caracterizados com:
  - Estado
  - Nodo pai
  - Ação
  - Custo (*path-cost*)
  - Profundidade
- Estado inicial é o nodo raíz da árvore.
- Expansão dos nodos segue uma **estratégia de busca**.
- Node $\neq$ estado.

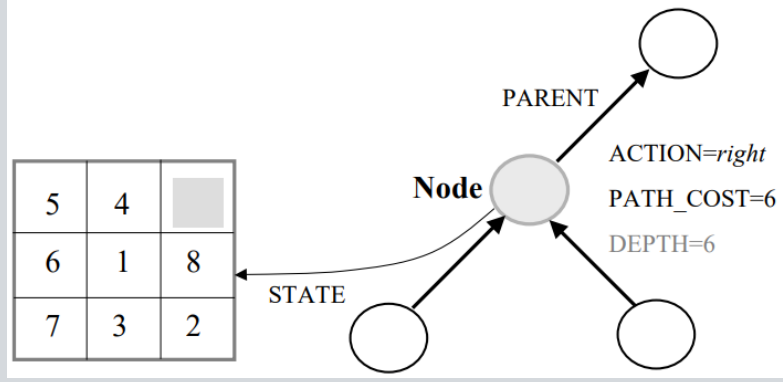


Figura 1: Estrutura de um nodo de uma árvore de busca.

Busca em Espaço de Estados

Árvore de Busca

- As soluções podem ser um **caminho**, do estado inicial até um estado objetivo, ou um **estado** em si.
  - Caminho: o mecanismo de busca é livre para escolher qualquer caminho dentro da árvore de busca.
  - Estado: Jogos - a cada passo, soluções parciais (melhor jogada) são indicadas e uma nova árvore de busca deve ser construída a partir do movimento do oponente.

Mensurando desempenho de algoritmos de busca

- Completeza: Obtem-se uma solução quando houver uma?
- Otimalidade: Encontra-se a solução ideal?
- Complexidade temporal: Tempo para encontrar uma solução?
- Complexidade espacial: Memória necessária para realizar a busca?

Obs: **b** como *branching factor*; **d**, a profundidade do nodo objetivo mais raso; e **m**, o comprimento máximo de qualquer caminho no espaço de estados.

Métodos de Busca

- Busca Cega (BC): Sem informações específicas sobre os estados (geração de sucessores e identificação de estados objetivos).
  - Breadth-first search (BFS)
  - Depth-first search (DFS)
  - Busca Bidirecional
- Busca Heurística (BH): Utiliza funções de avaliação no processo de busca.
  - Busca Gulosa (Greedy best-first search)
  - Algoritmo A*
  - Algoritmo IDA*
  - Busca Local: Utilizado em problemas de otimização.
    - Subida de Encosta (Hill-climbing search)
    - Busca Tabu

BC: Busca em Largura (BFS)

- O nó raiz é expandido primeiro, em seguinda seus sucessores, e após os sucessores dos sucessores.
- Expansão dos nodos ocorre, de forma completa, em casa nível (profundidade) da árvore de busca.
- Utiliza fila first-in-first-out (FIFO).
  - Nodos superficiais são expandidos antes dos mais profundos.

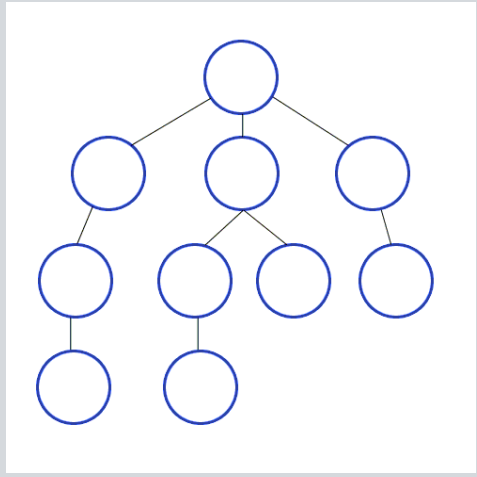


Figura 2: "BFS" by Mre is licensed under CC BY-SA 3.0.

BC: Busca em Largura (BFS) - Características

- Positivas
  - Completeza: Se o nodo objetivo existir em uma profundidade d, BDF o encontrará.
  - Otimalidade: BFS é ótimo se o custo do caminho for uma função não decrescente relacionada à profundidade do nodo.
- Negativas
  - Complexidade Temporal: Considerando um espaço de estados em que cada nodo possui b sucessores, e que a solução esta na profundidade d: $b + b^2 + ... + b^d + (b^{d+1} - b) = O(b^{d+1})$.
  - Complexidade Espacial: Todos os nodos expandido devem permanecer na memória ($O(b^{d+1})$).
    - Complexidade espacial equivale à complexidade temporal (mais um nodo para a raiz).

Exponential-complexity search problems cannot be solved by uninformed methods for any but the smallest instances (p.74).

BC: Busca em Profundidade (DFS)

- A expansão sempre ocorre até o nodo mais profundo da *fringe* atual da árvore de busca.
- Os nodos expandidos são retirados do *fringe* atual, e a busca segue para o próximo nodo mais raso ainda inexplorado.
- Utiliza fila last-in-first-out (LIFO).
  - Alternativa: Implementação com função recursiva.

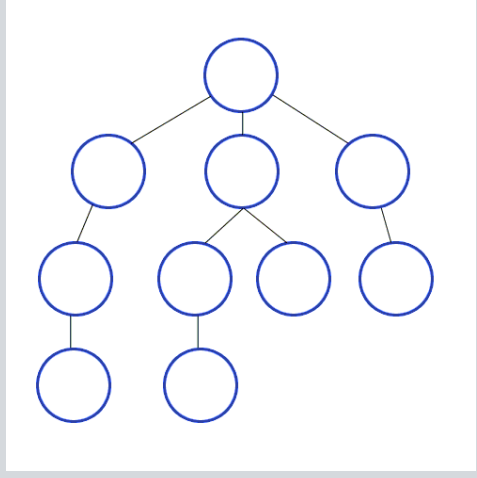


Figura 3: "DFS" by Mre is licensed under CC BY-SA 3.0.

BC: DFS e a Recursividade

Algorithm 1 Implementação recursiva de depth-limited search

```
function DEPTH-LIMITED-SEARCH(problem,limit) return solution or failure/cutoff
    return RECURSIVE-DLS(Make-Node(Initial-State[problem]),problem,problem,limit)
function RECURSIVE-DLS(successor, problem, limit) return solution,failure/cutoff
    cutoff_occurred? ← false
    if Goal-Test[problem](State[node]) then return SOLUTION(node)
    else if Depth[node] = limit then return cutoff
    else for all successor in EXPAND(node, problem) do
        result ← RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff_occured? ← true
        else if result ≠ failure then return result
    if cutoff_occured? then return cutoff else return failure
```

BC: Busca em Profundidade (DFS) - Características

- Positivas
  - Complexidade Espacial: Simples, uma vez que se armazena apenas um caminho da raís até a folha, junto com os nodos irmãos inexpandidos para cada nodo no caminho ($O(bm)$).
- Negativas
  - Complexidade Temporal: Considerando um espaço de estados em que cada nodo possui b sucessores, e que a máxima profundidade é m: $O(b^m)$.
  - Completeza: Não é completo. Se a busca ocorrer em uma subárvore com profundidade ilimitada e sem soluções, esta nunca terminaria.
  - Otimalidade: Não é ótimo. Pode ficar presa e percorrer um caminho infinito, quando uma caminho diferente levaria a uma solução mais próxima da raiz da árvore de pesquisa.

BC: Busca Bidirecional

- Execução de duas buscas simultâneas, uma partindo do estado inicial e outra do estado objetivo, até um caminho ser formado.
  - Estado em comum é detectado entre as buscas.
  - Figura 4: A soma das áreas dos círculos menores é menor do que a de um circulo centrado no início, alcançando o nodo objetivo.
- Implementação feita por meio da verificação, pelas buscas, do nodo a ser explorado. Caso o nodo verificado faça parte da *fringe* de uma das árvore busca, uma solução foi descoberta.

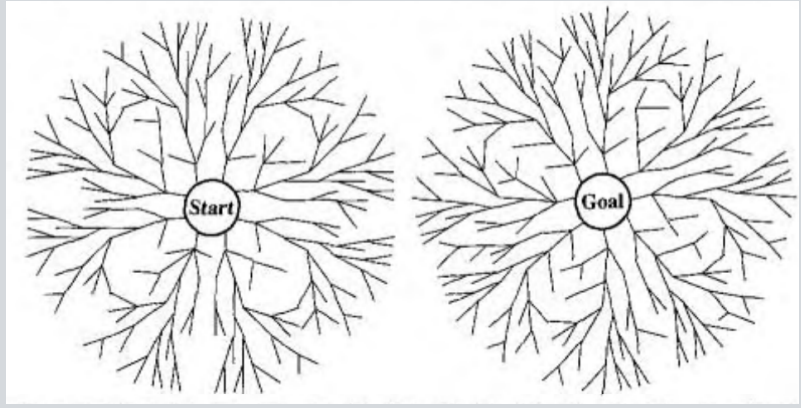


Figura 4: Ilustração da Busca Bidirecional

BC: Busca Bidirecional - Características

- Positivas
  - Complexidade Temporal: $O(b^{d/2})$, e uma tabela *hash* para checagem de fechamento de caminho entre as buscas pode ser feito em tempo contante.
  - Complexidade Espacial: Pelo menos uma das duas árvores de busca geradas deve ser mantida em memória para checagem de nodos em comum, logo estima-se $O(b^{d/2})$.
- Negativas
  - O algoritmo é completo e ótimo (considerando custo de caminho e passo uniformes) se ambas as estratégias de busca forem BFS.
  - Necessário armazenar informação sobre os antecessores dos nodos.
  - Grande dificuldade quando a solução hipotética representa implicitamente um grande conjunto de possíveis soluções.
    - Condição de *checkmate* em um jogo de xadrez.

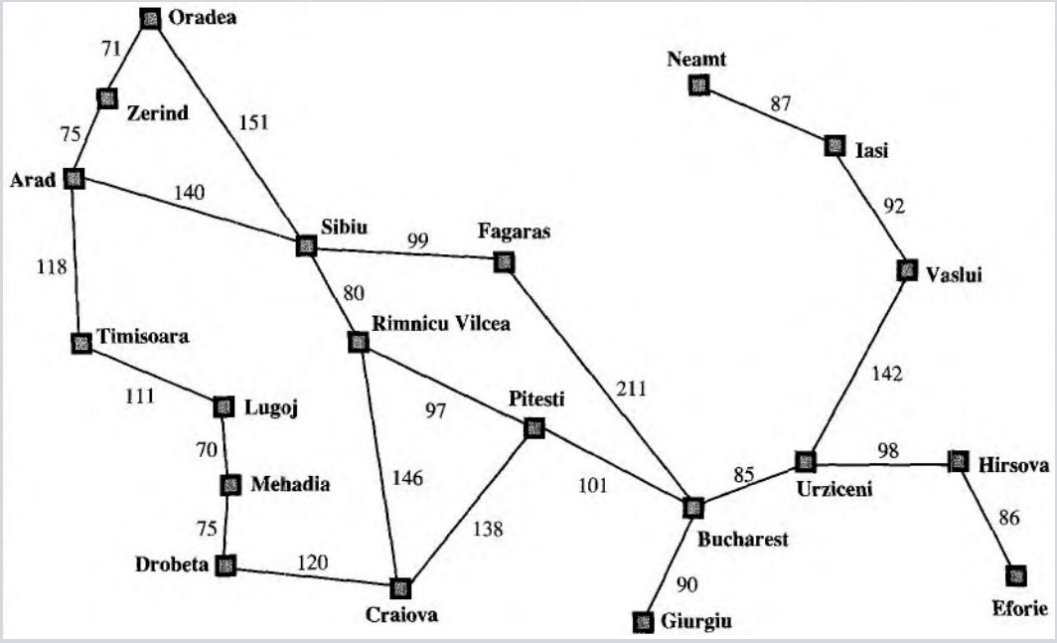
Busca Heurística (*Heuristic or Informed Search*)

- É uma instância de árvore de busca.
  - Expansão de nodos baseado em uma função de avaliação $f(n)$, ao qual mensura distância até o nodo objetivo, a ser minimizada.
- Sem garantia de solução ótima, mas de solução em tempo razoável.
- Utilização de conhecimento específico do problema além da definição do problema em si.
- Implementação de função heurística de estimativa.
  - $h(n) =$ custo estimado do caminho mais barato do nodo n até o nodo objetivo.
  - São a forma comum de adição de conhecimento específico do problema ao algoritmo de busca.

BH: Busca Gulosa

- Tenta expandir o nodo mais próximo do nodo objetivo, direcionando o processo à solução.
- Avaliação dos nodos por meio da função heurística: $f(n) = h(n)$.
- A minimização de $h(n)$ é suscetível a falsos inícios.
  - Pode expandir nodos desnecessários.
  - Verificação de estados repetidos.
- Semelhante ao DFS.
  - Não é ótimo e é incompleto.
  - Complexidade Temporal e Espaço: $O(b^m)$, em que m é a máxima profundidade do espaço de busca.
  - Heurísticas melhores podem amenizar a complexidade.

BH: Busca Gulosa



(a) Cidades na Romênia

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

(b) Função $h(n)$ de menor distância em linha reta até Bucareste.

Figura 5: Espaço de busca romeno.

Ir de Arad à Bucareste, por meio de busca gulosa, utilizando a função heurística de menor distância em linha reta (Figura 5b).

Busca Gulosa: *Arad* $\rightarrow$ *Sibiu* $\rightarrow$ *Fagaras* $\rightarrow$ *Bucareste*

Menor distância: *Arad* $\rightarrow$ *Sibiu* $\rightarrow$ *Râmnicu Vâlcea* $\rightarrow$ *Pitești* $\rightarrow$ *Bucareste*

Nova busca, de Iasi para Fagaras?!

BH: Busca Gulosa

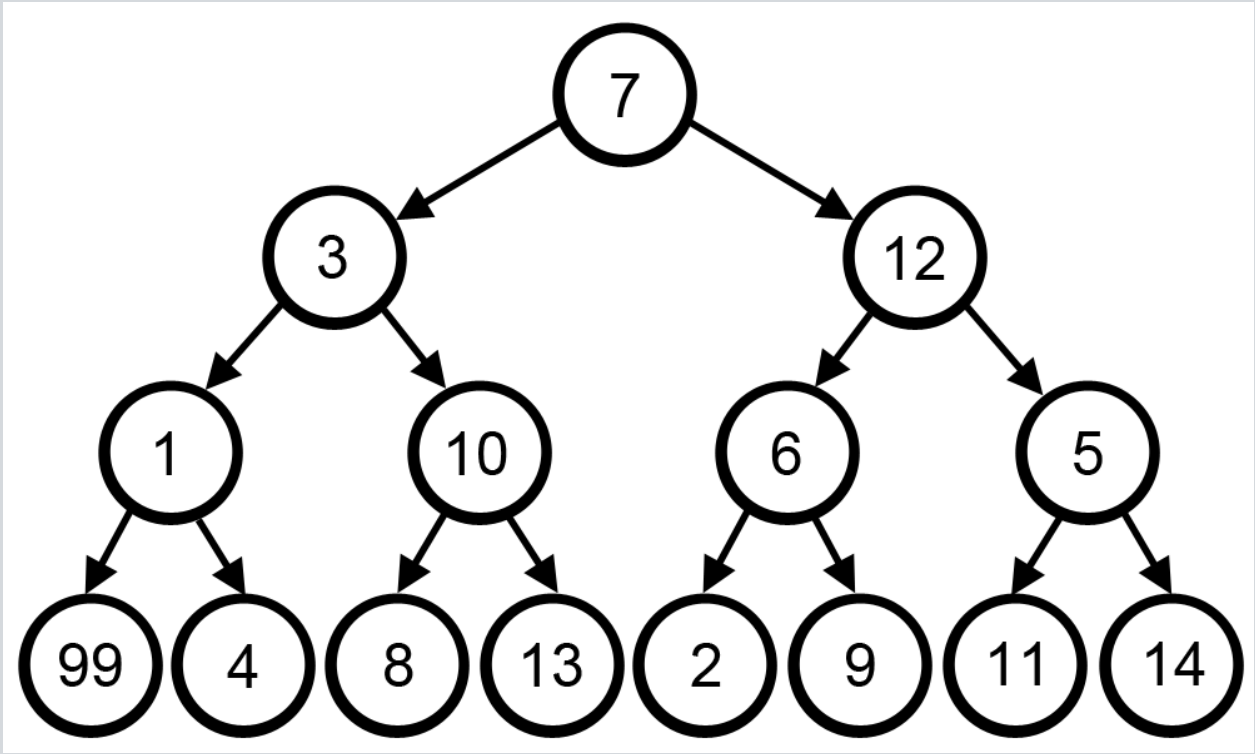


Figura 6: "*A example of greedy algorithm, searching the largest path in a tree*" by Swfung8 is licensed under CC BY-SA 3.0.

BH: Algoritmo A*

Função heurística $f(n)$

$$f(n) = g(n) + h(n) \tag{1}$$

- $g(n)$ é o custa até a chagada do nodo de início até no nodo n.
- $h(n)$ é o custo estimado do nodo n até o estado final mais próximo.

Logo, têm-se que $f(n)$ representa o **menor custo estimado** de uma solução que passe pelo nodo n.

BH: Algoritmo A* - Implementação

Algorithm 2 Implementação de Busca A*

1: **function** A*(problem) **return** a solution, or failure
2: $open \leftarrow \{(e_0, \perp, g(e_0) = 0, h(e_0))\}$ ▷ put the starting node on the open list
3: $closed \leftarrow \emptyset$
4: **if** $open = \emptyset$ **then return** *Failure*
5: $n_i \leftarrow min_f(open)$ ▷ find the node with the least f on the open list
6: $open \leftarrow open - \{n_i\}$ ▷ pop n_i off the open list
7: $closed \leftarrow \cup\{n_i\}$ ▷ push n_i on the closed list
8: $n_i = (e_i, p_i, g_i, h_i)$
9: **if** IS_FINAL(e_i) **then return** *Success*
10: **for all** e_j **in** SUCESSORS(e_i) **do** ▷ for each successor e_j from node n_i
11: $n_j = (e_j, n_i, g_i + cost(e_i, e_j), h(e_j))$
12: $n_o = (e_o, n_o, g_o, h_o) \in open$ and $e_j \equiv e_o$
13: $n_c = (e_c, n_c, g_c, h_c) \in closed$ and $e_j \equiv e_c$
14: **if** $\nexists n_o \wedge \nexists n_c$ **then** $open \leftarrow open \cup \{n_j\}$
15: **if** $\exists n_o \wedge f_j < f_o$ **then** $open \leftarrow open \cup \{n_j\} - \{n_o\}$
16: **if** $\exists n_c \wedge f_j < f_c$ **then** $open \leftarrow open \cup \{n_j\}, closed \leftarrow closed - \{n_j\}$
17: Return to line 4

BH: Algoritmo A*

Função heurística $f(n)$

- A* é ótimo se $h(n)$ for uma **heurística admissível**.
  - $h(n)$ nunca superestima o custo para atingir a solução.
  - $f(n)$ **nunca superestima o verdadeiro custo** da solução que passe pelo nodo n.

Logo, têm-se que $f(n)$ representa o **menor custo estimado** de uma solução que passe pelo nodo n.

BH: Algoritmo A*

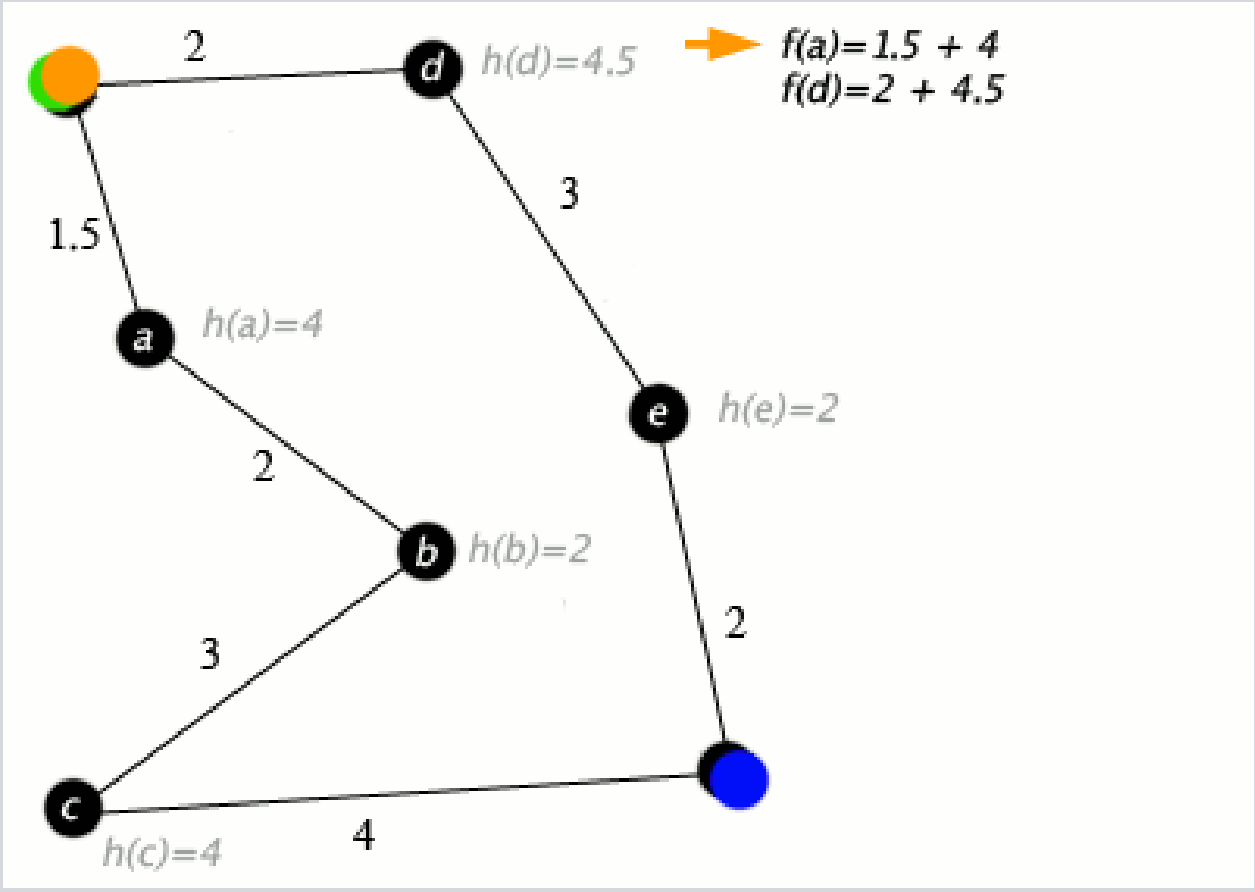


Figura 7: "*An example of A star (A*) algorithm in action (nodes are cities connected with roads, h(x) is the straight-line distance to target point) green - start, blue - target, orange - visited*" by CountingPine is licensed under CC0.

BH: Algoritmo Iterative-deepening A* (IDA*)

- Variação do algoritmo A* com DFS.
  - Utiliza estratégia de busca em profundidade iterativa.
- Redução dos requisitos de memória para A*.
  - Pesquisa heurística limitada por memória.
- Em cada iteração, realiza um DFS, desconsiderando um caminho quando seu custo tatal f excede um valor de corte. Tal valor começa com a estimativa do estado inicial, sendo sistematicamente atualizado.
- A cada iteração, o valor de corte é o menor custo da função heurística f de qualquer nodo que excedeu o corte na iteração anterior.

BH: Algoritmo IDA* - Implementação

Algorithm 3 Implementação de Busca IDA*

```
 1: function IDA*(problem) return a solution, or failure
 2:     threshold ← h(n_0); path ← {n_0}
 3:     loop
 4:         temp ← SEACH(path, 0, threshold)
 5:         if temp = found then return (path, threshold)
 6:         if temp = ∞ then return not_found
 7:         threshold ← temp
 8: function SEARCH(path,g,threshold)
 9:     node ← path.last; f ← g + h(node)
10:     if f > threshold then return f
11:     if IS_FINAL(node) then return found
12:     min ← ∞
13:     for all succ in SUCESSORS(node) do
14:         if succ not in path then
15:             path ← path ∪ {succ}
16:             temp ← SEARCH(path, g + cost(node, succ), threshold)
17:             if temp = found then return found
18:             if temp < min then min ← temp
19:             path ← path − {succ}
    return min
```

BH: Algoritmo Iterative-deepening A* (IDA*)

- Como o algoritmo A*, IDA* possui otimalidade se $h(n)$ for uma **heurística admissível**.
- É prático para problemas com custo unitários, evitando sobrecarga associada com manter uma fila ordenada de nodos, uma vez que possui apenas em sua memória os nodos presentes no caminho atual.
- Possui complexidade espacial de $O(d)$, em que d é profundidade do nodo objetivo mais raso.
- Exemplo visual: `https://algorithmsinsight.files.wordpress.com/2016/03/ida-star.gif`
- Uma aplicação famosa de tal algoritmo é na resolução de Cubo de Rubik.
- 3x2 blocos deslizantes com IDA*: `https://www.movingai.com/SAS/IDA/`

BH: Subida de Encosta

- Função heurística para exploração do espaço de estados.
  - *State space landscape* com "localização" (definida pelo estado) e "elevação" (definida pelo valor da função de custo heurística ou função objetivo).
- Implementação de um *loop* que continuamente visa a maximização do valor (em direção ao "pico").
  - A cada passo, um novo estado é criado e avaliado.
- Não mantém uma árvore de busca. O nodo atual apenas armazena o estado e o valor de sua função objetivo.
- Também denominada de *greedy local search*.

BH: Subida de Encosta - Algoritmo

Algorithm 4 Implementação de Subida de Encosta

function HILL-CLIMBING(problem) **return** a state that is local maximum
  current ← MAKE-NODE(*Initial* − *State*[*problem*])
  loop
    neighbor ← a highest-value-successor of *current*
    if Value[*neighbor*] ≤ Value[*current*] **then return** State[*current*]
    current ← *neighbor*

BH: Subida de Encosta - Características

Negativas

- Ficar cativo a um máximo local.
- *Ridges* ou Cordilheiras: sequências de máximas locais dificultam a navegação de tal algoritmo.
- Platôs: área no espaço de estados em que a função de avaliação é "plana". Pode não retornar uma solução.
- Não armazena caminhos, logo não retrocede.

BH: Subida de Encosta - Variações

Variações

- *Stochastic hill climbing*: Escolhe aleatoriamente entre os movimentos ascendentes; a probabilidade de seleção pode variar com a inclinação do movimento.
- *First-choice hill climbing*: Implementa *stochastic hill climbing* gerando sucessores aleatoriamente até se deparar com um melhor do que o estado atual.
  - Bom caso um estado tenham muitos sucessores.
- *Random-restart hill climbing*: Execução de uma série de buscas de subida de enconsta de estados iniciais aleatoriamente gerados, parando quando uma solução é encontrada.
  - Garantia de sucesso por razão trivial.

BH: Busca Tabu

- É um procedimento adaptativo auxiliar, que guia um algoritmo de busca local na exploração contínua dentro de um espaço de busca.
- Mantém uma lista tabu dos k estados previamente visitados, aos quais não podem ser revisitados.
  - Evita retornar a um ótimo local visitado previamente.
  - A lista permanece na memória durante um determinado espaço de tempo ou certo número de iterações (prazo tabu).
- Aumente a eficiência ao pesquisar em gráfos e permite o algoritmo superar a otimalidade local.

BH: Busca Tabu

- A lista tabu clássica contém os movimentos reversos aos últimos $|T|$ movimentos realizados (em que $|T|$ é um parâmetro do método) e funciona como uma fila de tamanho fixo.
  - Quando um novo movimento é adicionado à lista, o mais antigo sai.
- Excluem-se da busca os vizinhos da solução corrente obtidos por movimentos que constam na lista tabu.
- Reduz o risco de ciclagem, mas, ainda, pode proibir movimentos para soluções que ainda não foram visitadas.
- Função de aspiração para retirar o *status* tabu de um movimento, sob certas circunstâncias.

BH: Busca Tabu

- Nível de aspiração A(v):.
  - Uma solução s' em V(subconjunto V da vizinhança N(s) da solução corrente s) pode ser gerada se $f(s') < A(f(s))$, mesmo que o movimento m esteja na lista tabu.
  - Para cada valor v da função objetivo, retorna outro valor A(v), que representa o valor que o algoritmo aspira ao chegar de v.
  - Aspiração clássica: Aspiração por objetivo, em que se aceita um movimento tabu apenas se levar a uma solução melhor do que a solução melhor do que a atual (função objetivo global).
    - $A(f(s)) = f(s*)$, em que é a melhor solução atual.

BH: Busca Tabu

Critérios de parada

- Quando é atingido um certo **número máximo de iterações**, sem melhora no valor da melhor solução.
- Quando o valor da melhor solução chega a um **limite inferior conhecido** (ou próximo dele).

BH: Algoritmo Tabu - Implementação

Algorithm 5 Implementação de Busca Tabu

1: **function** BUSCA-TABU(problem)
2: **be** s_0 *the initial solution*
3: $s* \leftarrow s_0$
4: $Iter \leftarrow 0$
5: $BestIter \leftarrow 0$
6: **be** $BTmax$ *the maximum number of iterations without improvement in* $s*$
7: $T \leftarrow \emptyset$
8: INIT-ASPIRACAO( )
9: **while** $Iter - BestIter \leq Btmax$ **do**
10: $Iter \leftarrow Iter + 1$
11: $s' \leftarrow s$
12: UPDATE(T)
13: $s \leftarrow s' \oplus m$ the best element of $V \subseteq N(s)$ such that movement m is not tabu ($m \notin T$) or s' meets the ASPIRACAO($f(s) < f(s*)$)
14: **if** $f(s) < f(s*)$ **then** $s* \leftarrow s$; $BestIter \leftarrow Iter$
15: UPDATE-ASPIRACAO()
return $s*$

BH: Busca Tabu

Typos de Memória (Estratégias)

- Memória de curto prazo: lista de soluções recentemente consideradas, se acessíveis por meio de movimentos presentes na lista tabu, são desconsideradas até um certo prazo tabu.
- Memória de médio prazo (intensificação): Objetiva concentrar a pesquisa em determinadas regiões, do espaço de busca, consideradas promissoras.
  - Visitação à uma solução já vista para explorar sua vizinhança de forma mais completa.
  - Incorporação de atributos das melhores soluções já encontradas.
- Memória de longo prazo (diversificação): Direcionar a pesquisa para regiões ainda não exploradas do espaço de soluções.
  - Geração de soluções com atributos mais diversificados dos encontradas nas melhores soluções visitadas.
  - Aplicada pontualmente. Situações em que o algoritmo chegou em seu limite de análise em tal região.

BH: Busca Tabu

- Por vezes, a busca tabu é utilizada em combinação com outras metaheurísticas para a criação de metodologias de otimização híbridas.
- Uma combinação famosa é a de busca tabu com a metaheurística, baseada em conceito de população, *scatter search*. Esta que demonstra produzir resultados de alta qualidade para problemas difíceis de otimização combinatória.

BH: Busca Tabu - Aplicações Famosas

- Design
  - Rede tolerantes à falhas
  - Design de redes de transporte
  - Arquitetura de planajamento de espaço
- Produção, inventário e investimento
  - *Flexible Manufacturing*
  - *Multi-item Inventory Planning*
  - *Fixed Mix Investment*
- Roteamento
  - Roteamento de janelas de tempo
  - Caixeiro viajante
  - *Mixed Fleet Routing*
- Otimização de grafo
  - Problema do clique máximo
  - Particionamento de grafos
  - Coloração de grafos
- Scheduling
  - Escalonamento em máquinas
  - *Flow Shop Scheduling*
  - *Job Shop Scheduling*

Bibliografia

- Russell, S., Norvig, P. (2003). Artificial Intelligence: A Modern Approach, SECOND EDITION. Pearson Education, Inc, (cap.3-4).
- G. Bittencourt, Inteligência Artificial: Ferramentas e Teorias, 3a Edição, Editora da UFSC, Florianópolis, SC, 2006 (cap. 4).
- E. Rich and K. Knight, Artificial Intelligence, McGraw-Hill, 1991 (cap.2-3)
- Korf, Richard E. (1985). "Depth-first Iterative-Deepening: An Optimal Admissible Tree Search". Artificial Intelligence.
- Notas de aula:
  - BUSCA TABU - `https://www.ime.unicamp.br/~sandra/MS915/handouts/BuscaTabu.pdf`

Obrigado!

Contato:
m.luize.pinheiro@posgrad.ufsc.br, jerusa.marchi@ufsc.br

UNIVERSIDADE FEDERAL
DE SANTA CATARINA