Capítulo 4: Métodos de Busca Heurística

Busca pela Melhor Escolha (***best-first***)

- **Função de avaliação:** avalia um nodo segundo certo "critério" (resultado: um *número*).
- Em cada passo, expande o nodo não expandido com "melhor" avaliação.

Figura 4.1

Duas abordagens:

- Busca gulosa
- A^*

Busca gulosa:

- Minimizar **custo estimado** para atingir a meta.
- Expandir o nodo cujo estado é previsto como o **mais perto de uma meta**.

Função heurística:

$h(n)$ = custo estimado do caminho mais barato do estado n a uma meta.

$h(n) = 0$ se n é meta.

function GREEDY-SEARCH(*problem*) **returns** a solution or failure   **return** BEST-FIRST-SEARCH(*problem*,*h*)

Um Exemplo de Busca Gulosa:

$h_{DLR}(n) =$ distância em linha reta de n à meta.

Figura 4.2

Estágios em uma pesquisa gulosa por Bucharest usando h_{DLR}:

Figura 4.3

Solução encontrada:

- Arad→Sibiu→Fagaras→Bucharest

Solução ótima:

- Arad→Sibiu→Rimnicu Vilcea→Pitesti →Bucharest

Propriedades da Busca Gulosa:

- Completo: não (pode entrar em *loop*).
  Ex: Iasi$\rightarrow$Neamt$\rightarrow$Iasi$\rightarrow \cdots$
- Tempo: $O(b^m)$.
  Boa função h é essencial para melhorar o desempenho.
- Espaço: $O(b^m)$.
  Boa função h é essencial para melhorar o desempenho.
- Ótimo: não.

Busca A^*

- Combina busca por custo uniforme e busca gulosa.
- Função de avaliação: $f(n) = g(n) + h(n)$.

  $g(n)$: custo **real** do caminho do estado inicial até n.

  $h(n)$: custo **estimado** do caminho mais barato de n até um estado meta.

  $f(n)$: custo **estimado** do caminho mais barato do estado inicial até um estado meta.

A^* usa Heurística Admissível:

h tal que $h(n) \leq h^*(n)$ para todo n.

($h^*(n)$ dá o custo **real** de n até uma meta.)

Ex: h_{DLR}.

```
function A*-SEARCH(problem) returns a solution or failure
    return BEST-FIRST-SEARCH(problem,g + h)
```

Estágios em uma pesquisa A^* por Bucharest usando h_{DLR}:

Figura 4.4

$\Rightarrow$ Busca A^* é ótima!

Equação pathmax:

Seja n' um sucessor de n. Então:

$$\boxed{f(n') = max(g(n') + h(n'), f(n))}$$

Assim, f é **monotônica**.

Justificando por meio de um exemplo:

$$\begin{array}{ll} n & g(n) = 5,\ h(n) = 4 \Rightarrow f(n) = 9 \\ {\scriptstyle 1}\downarrow & \\ n' & g(n') = 6,\ h(n') = 2 \Rightarrow f(n') = 8 \end{array}$$

- Como h é **admissível** e $f(n) = 9$, o custo de um caminho solução passando por n é ≥ 9.
- Logo, o custo de um caminho solução passando por n' também é ≥ 9!

Otimalidade de A^* – Intuição:

Figura 4.5

Otimalidade de A^* – Prova:

Figura 4.6

G_2: **meta sub-ótima** ainda não expandida.

n: nodo não expandido no caminho até **meta ótima** G.

$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{pois } h(G_2) = 0 \\
&> g(G) && \text{pois } G_2 \text{ é sub-ótima} \\
&\geq f(n) && \text{pois } h \text{ é admissível}
\end{aligned}
$$

Como $f(G_2) > f(n)$, A^* não seleciona G_2 para expansão!

Propriedades de A^*:

- Completo: sim.
  Grafo com fator de ramificação finito e nodo com custo $\geq \delta$.
- Tempo: exponencial.
  (A menos que $|h(n) - h^*(n)| \leq O(\log h^*(n))$.)
- Espaço: exponencial.
  Mantém todos os nodos na memória.
- Ótimo: sim.

Funções Heurísticas

Ex: 8-puzzle/heurísticas admissíveis

Figura 4.7

- $h_1(n)$: número de ladrilhos fora de lugar.
- $h_2(n)$: *distância Manhattan* total.

Para o exemplo acima:

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

Dominância

Sejam h_1 e h_2 admissíveis.
h_2 **domina** h_1 se $h_2(n) \geq h_1(n)$ para todo n.
$\Rightarrow h_2$ é "melhor".

Quando h tende para h^*, o fator de ramificação médio tende para 1.

Custos típicos para 8-puzzle:

	Custo da busca			*F. de R. médio*		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
4	112	13	12	2.87	1.48	1.45
10	47.127	93	39	2.79	1.38	1.22
14	3.473.941	539	113	2.83	1.44	1.23
20	boom	7.226	676	$--$	1.47	1.27
24	boooom	39.135	1.641	$--$	1.48	1.26

Criando funções heurísticas

Heurísticas admissíveis podem ser obtidas de *soluções exatas* de **versões relaxadas** do problema.

Exemplos para *8-puzzle*:

- Um ladrilho pode ser movido para *qualquer posição*.

  ⇒ **Heurística:** $h_1(n)$.

- Um ladrilho pode ser movido para *qualquer posição adjacente*.

  ⇒ **Heurística:** $h_2(n)$.

- Sejam $h_1, h_2, \ldots, h_m$ heurísticas admissíveis.

  $$\Rightarrow h(n) = max(h_1(n), h_2(n), \ldots, h_m(n))$$

  domina todas.

- Usando-se problemas de *treinamento*, pode-se obter informações **estatísticas** para ajustar h.
- Função (não) linear de **características** do estado.
- **Custo** para calcular a função heurística é muito importante.

Algoritmos de Aperfeiçoamento Iterativo

- O algoritmo mantém apenas o **último estado** gerado.
- Para gerar o próximo estado, modifica (*aperfeiçoa*) o estado atual.

Aplicável em situações onde:

- O *caminho* do estado inicial até o atual é irrelevante: o estado atual contém toda informação necessária para orientar a busca.

$\Rightarrow$ Espaço constante!

Exemplos:

- Problema do Caixeiro Viajante.
- n Rainhas.

A idéia: encontrar um pico na superfície de estados.

Figura 4.13

⇒ Escalando o Everest com forte neblina, com amnésia...

Dois tipos de algoritmos de aperfeiçoamento iterativo:

- *Hill-climbing*
- *Simulated annealing*

Hill-climbing

Figura 4.14

Problema: máximos locais.

Simulated annealing

Idéia:

- Escapar de máximos locais permitindo alguns movimentos “ruins”, *mas* diminuindo progressivamente sua amplitude e freqüência.

Figura 4.15

Se a “temperatura” T *decresce devagar*, atinge o melhor estado.

Aplicaçẽs práticas em: leioute de VLSI, escalonamento, etc.