Busca Heurística ou Informada

Juliana Kaizer Vizzotto

Universidade Federal de Santa Maria

Disciplina de Inteligência Artificial

Roteiro

- Como as funções heurísticas afetam a performance
- Inventando funções heurísticas

Heurísticas e Performance

- Podemos usar o **fator efetivo de branching**, b^*, para caracterizar.
- Se o número total de nodos expandido por A^* para um problema particular é N, e a profundidade é d, então b^* é o fator de branching que uma árvore uniforme de profundidade d deveria ter para conter N nodos.
- Assim:

$$N = 1 + b^* + (b^*)^2 + ... + (b^*)^d$$

- Por exemplo, se A^* encontra uma solução na profundidade 5 usando 52 nodos, então o fator de branching efetivo é 1.91.
- Geralmente, o fator efetivo de branching exibido por uma dada heurística é razoavelmente constante sobre um número grande de instâncias do problema.

Heurísticas e Performance

- Assim, medidas experimentais de b^* em um conjunto pequeno de problemas fornece um bom guia sobre a heurística.
- **Uma heurística bem projetada deve ter um valor de b^* perto de 1, possibilitando que problemas consideravelmente grandes possam ser resolvidos**.
- Podemos testar as funções heurísticas h_1 e h_2 do problema 8-*puzzle* visto na aula passada!
- A tabela a seguir mostra 100 instâncias do problema geradas randomicamente, cada uma com soluções de tamanhos 2,4,...,20, usando A^* e *iterative deepening search* (lembram?).
- A figura mostra uma média do número de nodos expandido em cada estratégia e o fator de branching efetivo.
- Os resultados mostram que h_2 é melhor que h_1, e que busca cega é consideravelmente pior.

Heurísticas e Performance

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Figure 4.8 Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A* algorithms with $h1$, h_2. Data are averaged over 100 instances of the 8-puzzle, for various solution lengths.

Heurísticas e Performance

- h_2 é sempre melhor que h_1, pois podemos notar nas definições das heurísticas que para qualquer nodo n, $h_2(n) \geq h_1(n)$.
- Dizemos que h_2 **domina** h_1!
- Dominação se traduz diretamente em eficiência: A^* com h_2 expandirá menos nodos, em média, do que usando A^* com h_1.

Inventando Funções Heurísticas

- Vimos que ambos h_1 e h_2 são funções heurísticas boas para o jogo 8-*puzzle*.
- Já sabemos que h_2 é melhor!
- Mas ainda não sabemos como inventar uma função heurística.
- Como inventamos h_2?
- Você acha que um programa de computador pode mecanicamente inventar heurísticas?
- h_1 e h_2 são *estimativas* do tamanho do caminho restante para o 8-*puzzle*.
- Mas para versões simplificadas do jogo, tal como permitir que um quadrinho possa se mover para qualquer lugar (ao invés de somente permitir movimentos adjacentes), h_1 daria o custo **preciso** do menor solução.

Inventando Funções Heurísticas

- Similarmente, se um quadrinho pudesse se mover em qualquer direção, até mesmo em um espaço já ocupado, então h_2 daria o custo **preciso** do menor solução.
- Um problema com menos restrições nos operadores é chamado *problema relaxado*.
- *Em geral o custo de uma solução exata para o problema relaxado é uma boa heurística para o problema original.*
- Se a definição de um problema é escrita em uma *linguagem formal* é possível construir problemas relaxados automaticamente.

Inventando Funções Heurísticas

- Por exemplo, se os operadores do 8-*puzzle* são descritos como:
  - Um quadrinho pode mover de uma posição A para B, se A é adjacente à B e B é vazio.
- Pode-se gerar a versão relaxada do problema removendo uma ou mais condições:
  - Um quadrinho pode mover de uma posição A para B, se A é adjacente à B.
  - Um quadrinho pode mover de uma posição A para B, se B está vazio.
  - Um quadrinho pode mover de uma posição A para B.

Inventando Funções Heurísticas

- Como tarefa para o final de semana você deve ler o artigo sobre o programa ABSOLVER, que gera heurísticas automaticamente.

- **Exercício**: Uma versão em .pdf do artigo está disponível no site da disciplina.

Inventando Funções Heurísticas

- Um problema com a geração de novas heurísticas é que é difícil decidir claramente qual é a melhor heurística.
- Se temos uma coleção de heurísticas $h_1, h_2 ... h_m$ para um problema e nenhuma delas domina a outra, qual devemos escolher?
- Uma ideia é utilizar a melhor sempre que possível:

$$h(n) = max(h_1(n), ..., h_m(n))$$

- Essa composição heurística usa a melhor para o nodo em questão.

Usando Informações Estatísticas

- Outra maneira de inventar uma boa heurística é usar informação estatística.
- Isso pode ser feito executando uma procura sobre um número de problemas de treinamento, tal como escolher randomicamente 100 configurações para o 8-*puzzle* e obter estatísticas.
- Por exemplo, podemos descobrir que quando $h_2(n) = 14$, em 90% das vezes a distância real do objetivo é 18.
- Então, quando estamos rodando o problema real, podemos usar 18 sempre que $h_2(n)$ for igual a 14.
- É claro, que usando informação probabilística como essa, podemos perder a garantia da heurística ser *admissível*, mas vamos expandir menor nodos em média.

Usando Características dos Estados

- Frequentemente, podemos usar **características** dos estados que contribuem para a sua função de avaliação heurística.
- Por exemplo, no jogo de xadrez, o objetivo é colocar o oponente em cheque-mate, e características relevantes incluem o número de peças de cada tipo pertencendo a cada lado, o número de peças que são atacadas pelo lado oponente...
- Geralmente, a função de avaliação é uma combinação linear dos valores dessas características.
- Mesmo que não saibamos o quão importante cada característica é, ou mesmo se uma característica é boa ou ruim, é ainda possível usar algum algoritmo de aprendizado para obter coeficientes interessantes para cada característica.
- No xadrez, por exemplo, um programa pode aprender que a sua própria rainha deve ter um coeficiente positivo grande, e que o peão do oponente deve ter um coeficiente negativo

Mais sobre Heurísticas

- Outro fator que não consideramos é o custo de executar a função heurística em um nodo.
- Estamos assumindo que o custo de computar a função heurística é mais ou menos o mesmo que expandir o nodo.
- Assim, minimizar o número de nodos expandidos é interessante.
- Mas se a função heurística é tão complexa, que computar seu valor para um nodo leva o mesmo tempo de expandir centenas de nodos, então temos que reconsiderar tal heurística.