Busca em Espaço de Estados

Jerusa Marchi

jerusa.marchi@ufsc.br

Inteligência Artificial

Departamento de Informática e Estatística

INE - UFSC

Tipos de Problemas

- Problemas com solução algoritmica em tempo polinomial
  - encontrar a raiz quadrada de um número
  - encontrar as raizes de um polinômio de grau n
- Problemas com solução algoritmica em tempo exponencial (ou seja, cuja complexidade torna as soluções impraticáveis)
  - Circuito Hamiltoniano (problema do caixeiro viajante)
  - Verificar se uma fórmula lógica é satisfazível (SAT)

Tipos de Problemas

- Problemas com solução algoritmica em tempo exponencial
  - Problemas abordados pela IA
    - prova automática de teoremas
    - quebra-cabeças
    - jogos

Tipos de Problemas

- Estes problemas, além de não disporem de soluções algorítmicas viáveis, apresentam uma série de características que os torna bons candidatos para a pesquisa em IA:
  - São solucionáveis por seres humanos, e neste caso, sua solução está associada à inteligência
  - Formam classes de complexidade variável, existindo desde instâncias triviais (como o jogo da velha) até instâncias extremamente complexas (como xadrez)
  - São problemas de *conhecimento total*, isto é, tudo que é necessário saber para solucioná-los é conhecido, o que facilita sua formalização
  - Suas soluções têm a forma de uma sequência de situações legais e as maneiras de passar de uma situação para outra são em número finito e conhecidas

Busca em espaço de estados

Diante da falta de solução algorítmica viável, o único método de solução possível é a *busca*

- Elementos de um problema de busca:
  - Conjunto de descrições (*espaço de busca*), onde cada estado descreve uma possível situação do problema
  - Um *estado inicial* que descreve a situação inicial do problema
  - Um ou mais *estados finais*, que são as situações que se deseja alcançar
  - Um conjunto de *operadores* que determinam todos os estados que podem ser alcançados a partir do estado dado

Busca em espaço de estados

- Exemplos:
  - Problema dos Missionários e Canibais
  - Problema dos Jarros d'Água
  - Problema do Jogo dos oito

Busca em espaço de estados

- Uma vez que todas as possíveis transições são conhecidas, poderia se gerar um conjunto de regras de produção
  - Problema dos Jarros d'Água
    - Você tem dois jarros, um com capacidade de 4 litros e outro com capacidade de 3 litros. Nenhum deles tem qualquer marcação de medidas. Há uma fonte que pode ser usada para encher os jarros com água. Como você consegue colocar exatamente 2 litros de água na jarra de 3 litros?
    - Estado inicial?
    - Estado final?
    - Regras de Produção?

Busca em espaço de estados

- Problemas:
  - Para um problema simples, o número de regras é pequeno...já para um jogo de xadrez, o número de regras gira em torno de 10^{120} (possíveis posições do tabuleiro)
  - Escolha da estratégia de controle
    - Se a escolha da próxima regra for ineficiente, a solução do problema pode não ser encontrada

Melhor Solução
Explorar o espaço de busca de maneira sistemática

Árvores de Busca

- Associar o nó raiz ao estado inicial
- Os nós sucessores de qualquer nó (estado atual) são associados aos estados obtidos através da aplicação dos operadores sobre a descrição do estado associado ao nó atual

Árvores de Busca

- A solução do problema pode ser:
  - um caminho - prova automática de teoremas ou quebra-cabeça
    - o mecanismo de busca é livre para escolher qualquer caminho dentro da árvore de busca
  - um estado - jogos
    - onde deve-se considerar as possíveis jogadas do oponente
    - a cada passo, soluções parciais (melhor jogada) são indicadas e uma nova árvore de busca deve ser construída a partir do movimento do oponente

Estratégias de Busca

- Busca Cega - não considera informações acerca do problema a ser resolvido
  - Busca em Largura
  - Busca em Profundidade
  - Busca Bidirecional
- Busca Heurística - associa funções heurísticas ao processo de busca, privilegiando nós com melhor avaliação
  - Busca Gulosa
  - Algoritmo A*
  - Subida de Encosta
  - Têmpera Simulada

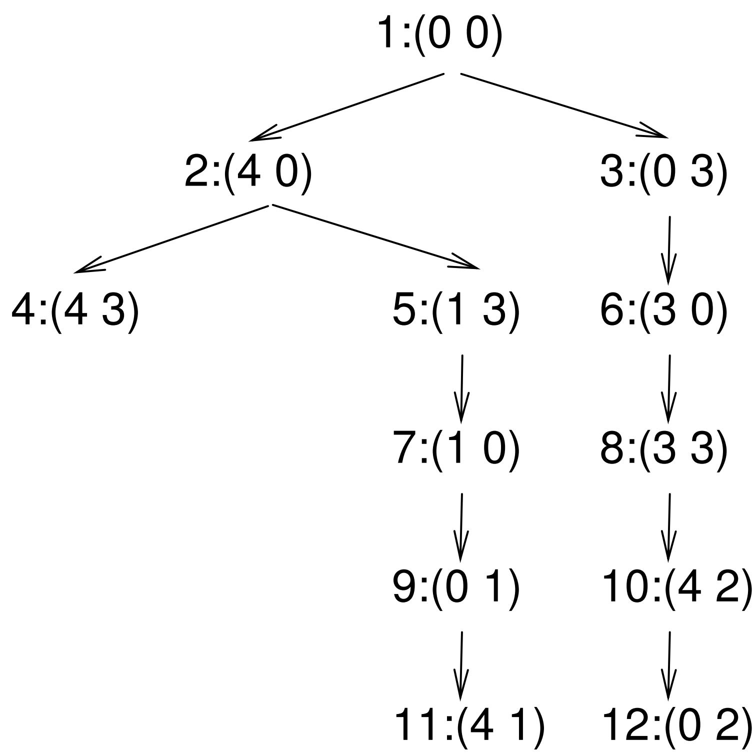
Estratégias de Busca

- Problema dos Jarros d'água
  - Representação:
    - Estados: Pares ordenados

      (<conteúdo em 4 litros> <conteúdo em 3 litros>)
    - Estado inicial (0 0)
    - Estado final (0 2)
    - Operadores: enche-4, enche-3, esvazia-4, esvazia-3, joga-43, joga-34

Busca em Largura

- Cada nível da árvore é inteiramente construído antes de qualquer outro nodo do próximo nível
- Listas de nós expandidos e a expandir devem ser mantidas para evitar “loops”



Busca em Largura

- Vantagens:
  - Nunca explora um beco sem saída ou percorre um caminho infrutífero por longo tempo;
  - Se houver uma solução, a busca em largura a encontrará;
  - Caso existam várias soluções, sempre a solução mínima será a encontrada.
- Desvantagens:
  - Se o espaço de estados for amplo, pode ocorrer o fenômeno da explosão combinatória;
  - A busca em largura requer grande quantidade de memória, pois todos os nós devem ser armazenados até que a solução seja encontrada;
  - É necessário explorar todos os nós do nível n antes de explorar os de nível $n + 1$.

Busca em Profundidade

- O nó expandido é sempre o último nó adicionado a árvore
- Listas de nós expandidos e a expandir devem ser mantidas para evitar “loops”

$$
\begin{tikzcd}
\text{1:(0 0)} \arrow[d] \\
\text{2:(0 3)} \arrow[d] \\
\text{3:(3 0)} \arrow[d] \\
\text{4:(3 3)} \arrow[d] \\
\text{5:(4 2)} \arrow[d] \\
\text{6:(0 2)}
\end{tikzcd}
$$

Busca em Profundidade

- Vantagens:
  - Baixo consumo de memória, uma vez que só os nós do caminho corrente precisam ser armazenados;
  - É possível encontrar a solução sem examinar grande parte do espaço de estados;
- Desvantagens:
  - Pode-se despender muito tempo em um caminho infrutífero, ou em caminhos cíclicos (caso o controle de ciclos não seja implementado);
  - Nem sempre a menor solução será a solução encontrada.

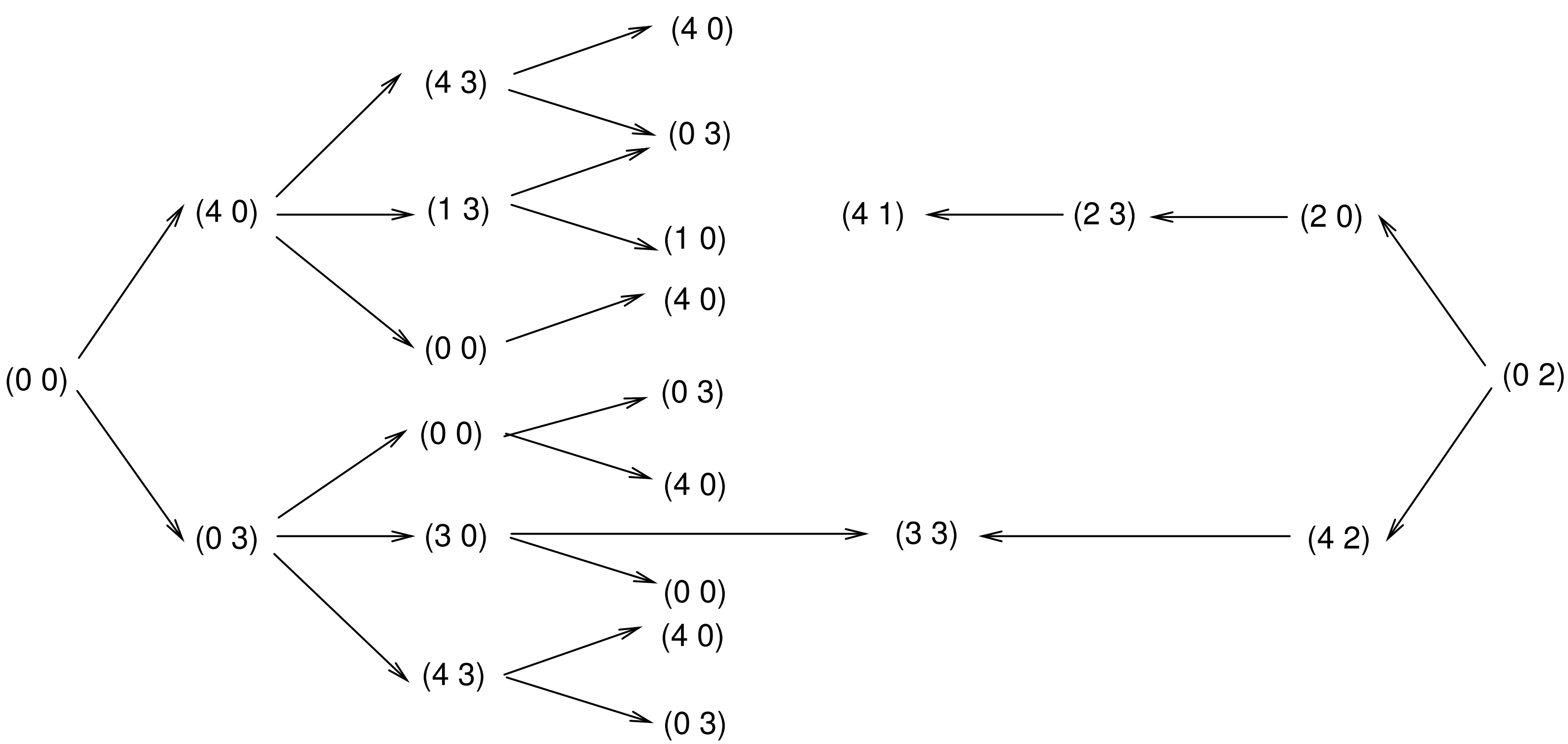
Algoritmo: Busca Cega

Busca Cega()

0. $open \leftarrow \{(e_0, \perp, 0)\}$;
   $closed \leftarrow \emptyset$;
1. se $open = \emptyset$ então retorne $Falha$
2. $n_i \leftarrow Seleciona(open)$
3. $open \leftarrow open - \{n_i\}$
4. $closed \leftarrow \cup\{n_i\}$
5. seja $n_i = (e_i, p_i, g_i)$
   se $Final(e_i)$ então retorne $Sucesso$
6. $\forall e_j \in Sucessores(e_i)$
   se $\nexists n_k = (e_k, p_k, g_k) \in open, e_j = e_k \wedge$
   $\nexists n_k = (e_k, p_k, g_k) \in closed, e_j = e_k$
   então $open \leftarrow open \cup \{e_j, n_i, g_i + 1\}$
7. $Volte\ para\ 1$

Busca Bidirecional

- A busca é realizada simultaneamente a partir do estado inicial e do estado objetivo
- A busca termina quando um estado comum é encontrado



Algoritmo: Busca Bidirecional

Busca Bidirecional()

0. $openUp \leftarrow \{(e_0)\}$;
   $openDown \leftarrow \{(e_f)\}$;
1. enquanto $openUp \neq \emptyset$ e $openDown \neq \emptyset$ do
2. $e_i \leftarrow Seleciona(openUp)$
3. se $e_i \in openDown$ então retorne $Sucesso$
4. senão $\forall e_j \in Sucessores(e_i)$
   $openUp \leftarrow openUp \cup \{e_j\}$
5. $e_k \leftarrow Seleciona(openDown)$
6. se $e_k \in openUp$ então retorne $Sucesso$
7. senão $\forall e_j \in Antecessores(e_k)$
   $openDown \leftarrow openDown \cup \{e_j\}$

Busca Bidirecional

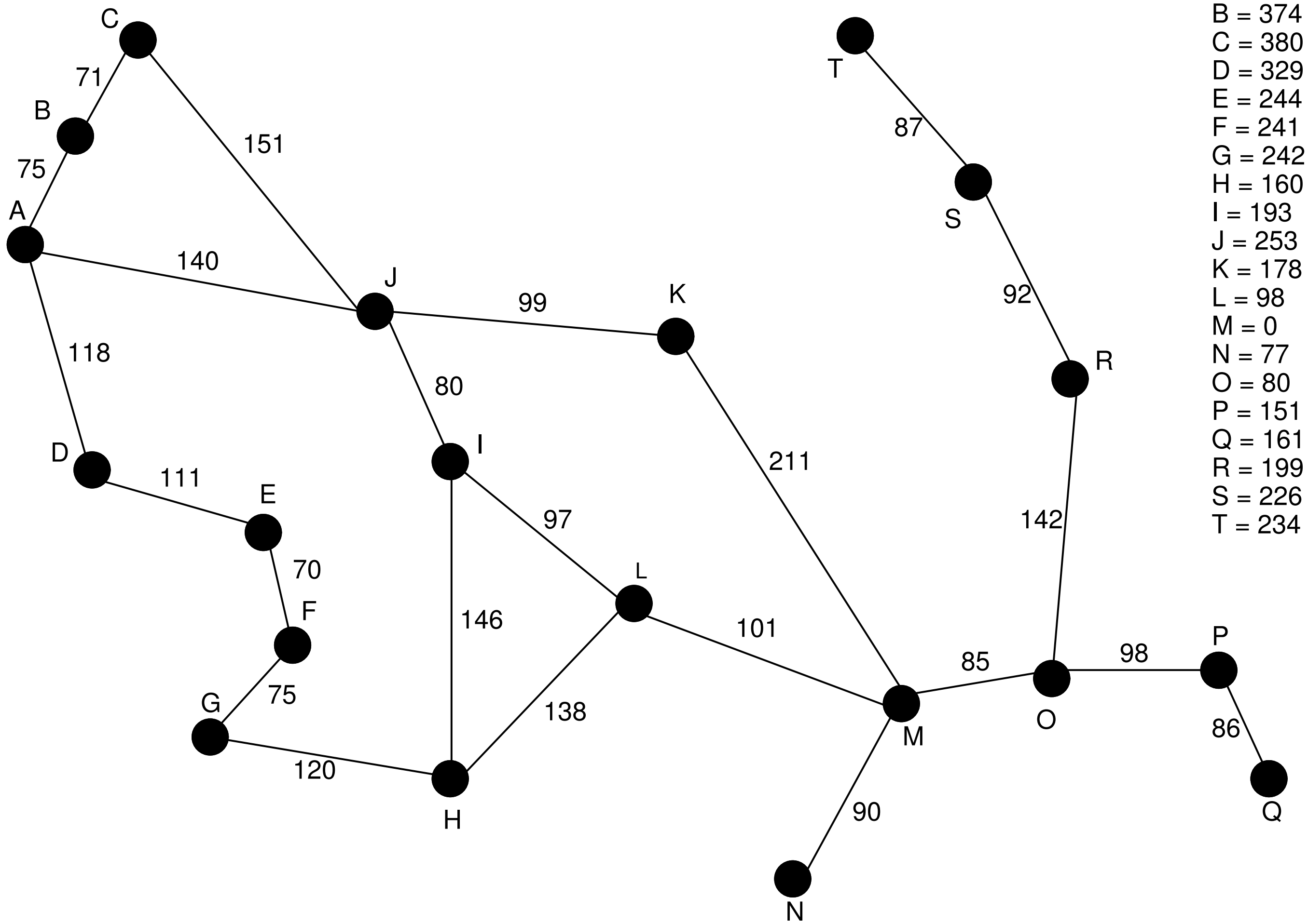
- Vantagens:
  - Menor consumo de memória, se comparado a busca em largura
  - Mais veloz do que as buscas em largura e profundidade
- Desvantagens:
  - É necessário gerar antecessores

Busca Heurística

- Compromisso entre a busca em largura e a busca em profundidade
- Objetivo: minimizar a árvore de busca
- não garante a solução ótima, mas garante uma solução em tempo razoável
- Usa uma função estimativa *h(n)* (Heurística) que avalia o custo aproximado da solução que passa pelo nó *n*
  - Para estimar $h(n)$ é necessário utilizar informações específicas sobre o problema
  - Na maioria dos problemas este custo não pode ser determinado com precisão

Busca Heurística

- Considere agora o problema de ir da cidade A até a cidade M



A = 366
B = 374
C = 380
D = 329
E = 244
F = 241
G = 242
H = 160
I = 193
J = 253
K = 178
L = 98
M = 0
N = 77
O = 80
P = 151
Q = 161
R = 199
S = 226
T = 234

Busca Gulosa

- Escolhe sempre o nodo com menor estimativa de custo (menor h)
- Para o problema da viagem de A a M, a função heurística pode ser a distância em linha reta até a cidade M

Busca Gulosa

- A busca gulosa pode não encontrar a solução ou ainda a solução encontrada não ser a de menor custo (mesmos problemas da busca em profundidade)
- Assim como a busca em largura, requer que as informações sobre os nodos expandidos seja armazenada (espaço)

Algoritmo A*

- A idéia é unir a eficiência da busca gulosa com a busca em largura, garantindo que a melhor solução seja encontrada
- associa duas funções:

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

  - O custo do caminho do estado inicial até o nó *n* é (*g(n)*)
  - O custo estimado do caminho de *n* até o estado final mais próximo (*h(n)*)

Algoritmo A*

*A**()

0. $open \leftarrow \{(e_0, \perp, 0, h(e_0))\}$; $closed \leftarrow \emptyset$;
1. se $open = \emptyset$ então retorne $Falha$
2. $n_i \leftarrow min_f(open)$
3. $open \leftarrow open - \{n_i\}$
4. $closed \leftarrow \cup\{n_i\}$
5. seja $n_i = (e_i, p_i, g_i, h_i)$

   se $Final(e_i)$ então retorne $Sucesso$
6. $\forall e_j \in Sucessores(e_i)$

   seja $n_o = (e_o, p_o, g_o, h_o) \in open, e_j = e_o,$

   $n_c = (e_c, p_c, g_c, h_c) \in closed, e_j = e_c,$

   $n_j = (e_j, n_i, g_i + 1, h(e_i))$

   se $\nexists n_o \wedge \nexists n_c$ então $open \leftarrow open \cup \{n_j\}$

   se $\exists n_o \wedge f_j < f_o$ então $open \leftarrow open \cup \{n_j\} - \{n_o\}$

   se $\exists n_c \wedge f_j < f_c$ então $open \leftarrow open \cup \{n_j\}$, $closed \leftarrow closed - \{n_c\}$
7. $Volte\ para\ 1$

Algoritmo A*

- O algoritmo A* assegura que a melhor solução seja encontrada desde que a função $h(n)$ seja admissível
- Uma função heurística é admissível se não superestima o custo para alcançar o objetivo
  - Por exemplo, a heurística utilizada para alcançar a cidade M é admissível, pois a distância real será maior ou igual à estimada

Subida de Encosta

- O algoritmo gera a cada passo um novo estado, que é avaliado
- Caso a avaliação seja melhor que a do estado corrente, este estado é escolhido como novo estado corrrente
  - Uso da função heurística para deslocar-se no espaço de busca

Algoritmo: Subida de Encosta

Hill Climbing()

0. $atual \leftarrow \{(e_0, h(e_0))\}$
1. $prox \leftarrow$ sucessor($atual$)
2. Se $h(prox) \leq h(atual)$ e $atual <> \{(e_0, h(e_0))\}$ então
   retorne $atual$ /*sem sucessor melhor*/
   senão $atual \leftarrow prox$
3. Volte para 1

Subida de Encosta

- Problemas
  - Máximos locais
  - Planaltos - todos os passos possíveis, a partir do nó corrente, parecem igualmente bons (ou ruins)
    - Nesse caso, a subida da encosta não é melhor do que a busca em profundidade
    - Pode não retornar uma solução
  - Não mantém a árvore, logo não pode retornar o caminho

Subida de Encosta (Variação)

- Subida de Encosta pela Trilha mais Íngreme
  - Considera todos os possíveis movimentos a partir do estado corrente e seleciona o melhor

Algoritmo: SE pela Trilha mais Íngreme

Steepest-ascent Hill Climbing $()$

0.$atual \leftarrow \{(e_0, h(e_0))\}$;

1. Se $atual$ = $meta$ então retorne $atual$

2. Senão repita até encontrar um estado $meta$ ou
   até que não haja alteração no estado $atual$

3.Seja $Sucessor$ um estado qualquer tal que qq. estado sucessor
   ao estado corrente seja melhor do que ele

4. Para cada operador aplicável ao estado corrente faça:

5. Aplique o operador e gere um novo estado $novo \leftarrow \{(e_j, h(e_j)\}$

6. Se $novo = meta$ então retorne-o e encerre.

7. Senão $h(novo) \geq h(Sucessor)$ então $Sucessor \leftarrow \{(e_j, h(e_j)\}$.

8 Se $h(sucessor) \geq h(atual)$ então
   $atual \leftarrow sucessor$

9. Volte para 2

Subida de Encosta (Variação)

- Considerações:
  - Difícil de aplicar quando há muitos operadores
  - Maior tempo para escolher um movimento
  - Menor número de movimento em comparação com a subida de encosta simples
  - Pode não encontrar a solução
    - Planaltos e máximos locais

Têmpera Simulada

- Para escapar dos máximos locais, pode-se permitir que a busca faça alguns movimentos para baixo
  - Ao invés de sempre pegar o melhor caminho, o algoritmo escolhe um caminho aleatório
  - Se o caminho melhorar a situação ele é escolhido. Caso contrário, existe probabilidade do caminho ser escolhido
  - A probabilidade de escolher um caminho ruim tende a decrescer com o tempo

Têmpera Simulada

- A função heurística passa a ser chamada de função objetivo
- A idéia é minimizar ao invéz de maximizar (descida de vale)
- Baseada na idéia da têmpera de metais

Algoritmo: Têmpera Simulada

Simulated Annealing()

0. $atual \leftarrow \{(e_0, h_0))\}$;
1. $T \leftarrow schedule[t]$ /*mapeamento do tempo decorrido para a “temperatura”*/
2. se $T = 0$ então retorne $atual$
3. senão $prox \leftarrow$ um sucessor (escolha aleatória de um operador)
4. $\Delta E \leftarrow h(prox) - h(atual)$
5. se $\Delta E < 0$ então $atual \leftarrow prox$
6. senão $atual \leftarrow prox$ com uma probabilidade $e^{-\Delta E/T}$
7. Volte para 1

Têmpera Simulada

- Considerações
  - A escolha de um *cronograma de têmpera* é empírica
  - O valor de decréscimo da temperatura é uma variável importante
    - Se a temperatura esfriar muito rápido, a solução pode ser um mínimo local
    - Se a temperatura demorar muito a esfriar, o tempo de execução pode ser elevado

Busca em espaço de estados

Bibliografia:

- G. Bittencourt, *Inteligência Artificial: Ferramentas e Teorias*, 3^a Edição, Editora da UFSC, Florianópolis, SC, 2006 (cap. 4)
- E. Rich and K. Knight, *Artificial Intelligence*, McGraw-Hill, 1991 (cap. 2 e 3)
- Notas de aula:
  - Prof. Jomi Hubner - http://www.inf.furb.br/∼jomi/