Escola Politécnica da Universidade de São Paulo

PCS2428 – Inteligência Artificial

Problemas de Decisão Seqüencial

Professora: Ana Helena da Costa Reali

André Felipe dos Santos Nº USP 3729471
Hernan da Cunha Martinez Nº USP 5178342
Leandro Moreira da Costa Nº USP 5178575

São Paulo, 05 de novembro de 2007

1 Índice

2 Objetivo

Já fora visto como realizar Problemas de Decisões Simples, onde como um agente deve tomar decisões de modo que, em média, ele consiga o que quer.

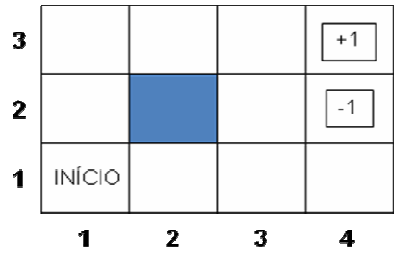
Nos Problemas de Decisões Simples a utilidade de cada ação no ambiente era conhecido.

Serão visto agora, nos Problemas de Decisões Complexos métodos para decidir o que fazer hoje, dado que nós teremos uma chance decidir de novo amanhã.

Nos Problemas de Decisões Sequenciais a utilidade dos agentes depende de uma sequência de decisões que envolvem incertezas e percepção.

3 Problemas de Decisão Seqüencial

Suponha o exemplo abaixo



A tarefa termina quando um dos estados +1 ou -1 é atingido. Sendo que +1 é o estado final ótimo e -1 é o estado final a ser evitado. Para cada estado 4 opções de movimentação são possíveis.

Na versão determinística do problema, cada ação move um quadrado para o lado desejado. Porém no caso mais genérico estocástico a intenção de mudar, por exemplo, do quadrado (1,1) para o quadrado (1,2) executa a mudança corretamente com 0.8 de chance, executa a mudança para (2,1) com 0.1 de chance e vai para a esquerda de encontro à parede com 0.1 de chance permanecendo-se no mesmo quadrado.

A solução do problema deve especificar o que o agente deve fazer em qualquer estado que ele esteja, porém uma seqüência fixa de ações não resolve, pois ações não confiáveis não geram estados deterministicamente. Sendo assim adota-se a metodologia de construir uma **Diretriz** que é a *recomendação* das ações que o agente executa para cada estado em que ele possa estar mapeando-se depois as ações a partir das diretrizes.

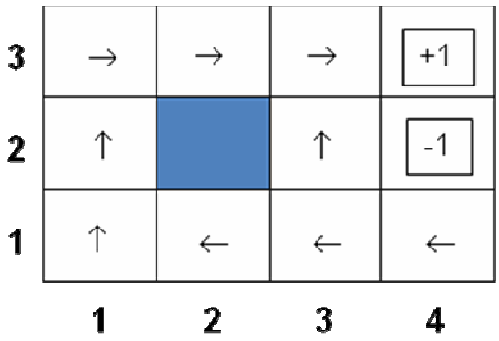
O problema de cálculo da Diretriz ótima em um ambiente acessível e estocástico num modelo de transição conhecido é chamado Problema de Decisão Markoviano onde as probabilidades de transição de um estado para outro depende apenas do estado atual, não de estados anteriores.

Uma vez calculado a Diretriz para o agente é trivial escolher a ação a ser executada para um estado dado que a Diretriz que produz a mais alta utilidade esperada é a escolhida:

function SIMPLE-POLICY-AGENT(*percept*) **returns** an *action*
static: *M*, a transition model
U, a utility function on environment histories
P, a policy, initially unknown

if *P* is unknown **then** ***P*** *<-- the optimal policy given *U, M*
return *P*[*percept*]

Assim, para o exemplo acima utilizando-se uma utilidade sobre seqüências de estados: U = 1/25*comprimento + UtilidadeDoEstado Alcançado , temos:



Cujas ações foram escolhidas a partir das utilidades da cada estado:

3	0.812	0.868	0.912	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388
	1	2	3	4

A seguir é explicado com calcula as utilidades de cada estado e depois a Diretriz que mapeia utilidades para ações.

4 Iteração de Valores

Objetivo da iteração: obtenção das utilidades de cada estado U(estado), para assim, selecionar a ação ótima.

A função utilidade apresenta uma dificuldade para ser calculada, que é a de não saber aonde uma ação levará. Uma idéia é usar o conceito de árvore, com a raiz sendo o estado atual, e caminhos até folhas sendo históricos de estados, com a notação H(*estado, diretriz*), denotando a árvore de histórico começando com *estado* e tomando ações de acordo com *diretriz*.

Temos a seguinte fórmula que retorna a utilidade de um estado:

$$U(i) \equiv EU\left(H(i, diretriz*)\middle|M\right) \equiv \sum P\left(H(i, política*)\middle|M\right) U_h\left(H(i, diretriz*)\right)$$

Ou seja, a utilidade é função da utilidade esperada, segundo uma diretriz ótima, que depende de um modelo de transição definido. Esta função é útil para tomar decisões racionais, de acordo com o princípio da máxima utilidade esperada. Uma necessidade para o caso de decisões seqüenciais é que a função de utilidade em históricos (U_h) seja separável, que é a seguinte propriedade:

$$U_h([s_0, s_1, \ldots, s_n]) = f(s_0, U_h([s_1, \ldots, s_n]))$$

Um exemplo de função utilidade separável é da forma aditiva:

$$U_h([s_0, s_1, \ldots, s_n]) = R(s_0) + U_h([s_1, \ldots, s_n])$$

Onde *R* é a função recompensa. Para nosso exemplo, com a função utilidade separável e aditiva, com a função recompensa *R* assumindo valores -1/25 para estados não terminais, +1 e -1 para os terminais. Funções em históricos são quase sempre aditivas. Outro exemplo de aditividade são as funções de custo de caminho nos algoritmos de busca com heurística.

Com uma função de utilidade com essas propriedades, e o princípio de máxima utilidade esperada, podemos ter que a ação ótima é a que dará a máxima utilidade nos estados de saída: $diretriz^*(i) = \arg\max_a \sum_j M_{ij}^a U(j)$

Sendo o *M* acima a probabilidade de se alcançar j, partindo de i, com ação a, e arg Max *f(a)* o valor de a cujo f seja máximo. E assim temos a utilidade dos estados, em função da utilidade dos sucessores:

$$U(i) = R(i) + \max_a \sum_j M_{ij}^a U(j)$$

Um problema que a fórmula acima oferece é que loops podem fazer os históricos não ter limites. A solução é utilizar um algoritmo iterativo que aproxima as utilidade dos estados. A equação acima é aplicada repetidamente, atualizando as utilidades de cada estado em função das utilidades dos estados vizinhos, ocorrendo convergência para muitas iterações:

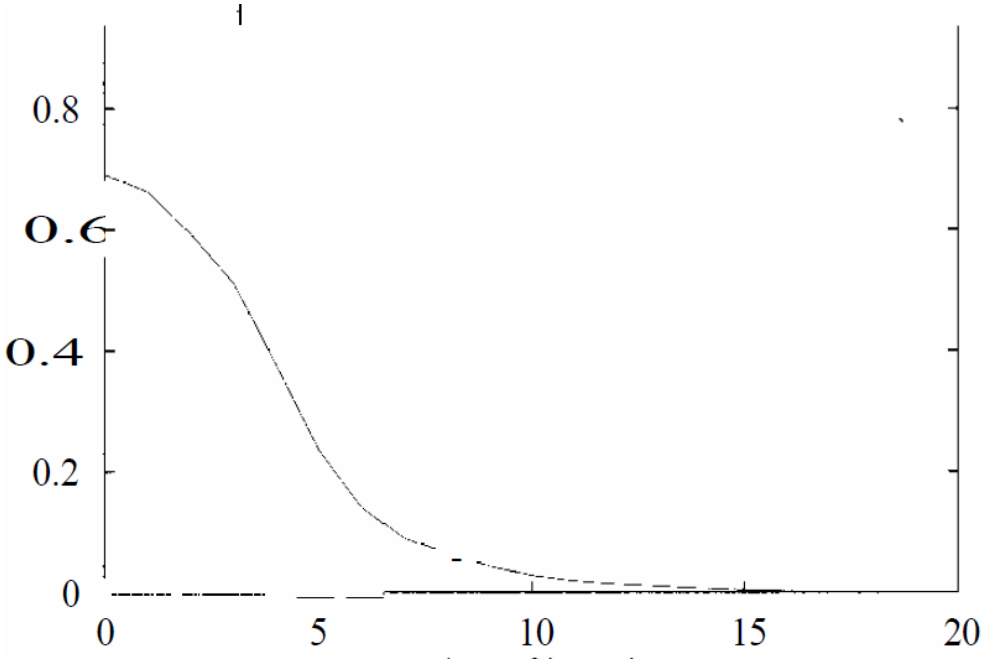
$$U_{t+1}(i) \leftarrow R(i) + \max_a \sum_j M_{ij}^a U_t(j)$$

Onde t é o número de iterações. O algoritmo em questão, Iteração de valores, é mostrado abaixo:

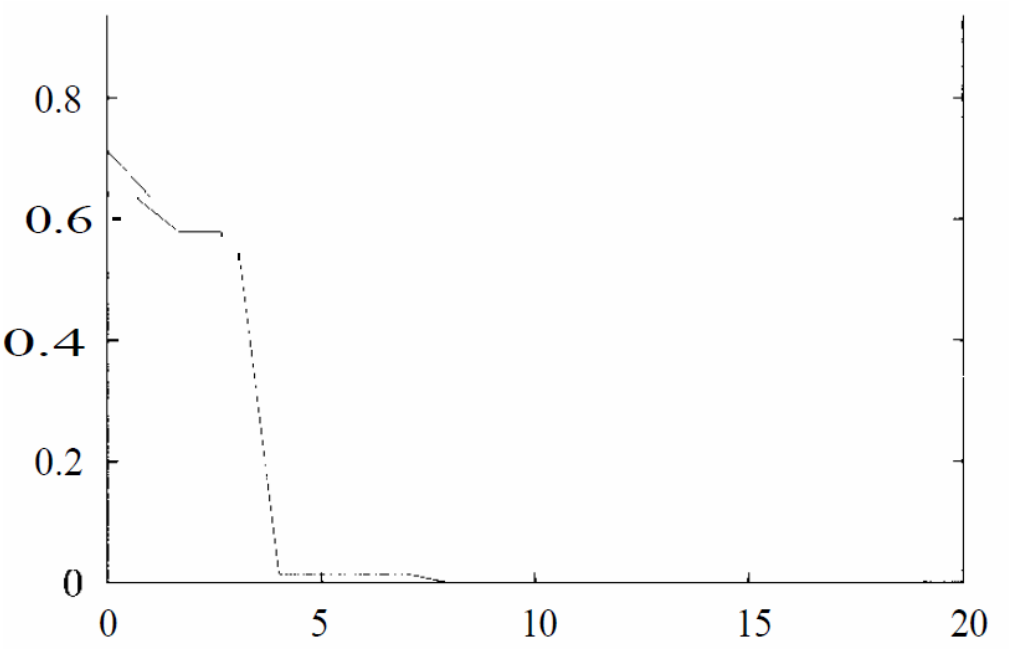
function VALUE-ITERATION(M, R) **returns** a utility function
  inputs: M, a transition model
    R, a reward function on states
  local variables: U, utility function, initially identical to R
    U', utility function, initially identical to R

  repeat
    $U \leftarrow U'$
    for each state i **do**
      $U'[i] \leftarrow R[i] + \max_a \sum_j M_{ij}^a U[j]$
    end
  until CLOSE-ENOUGH(U, U')
  return U

Uma questão importante é a quantidade de iterações. Para se medir o progresso, há duas maneiras. Uma é usar erro quadrático, comparando as os valores da função utilidade com os corretos. A outra é comparar a diretriz determinada pelas ações com a diretriz ótima, gerando uma métrica chamada de qualidade de diretriz. Abaixo, nas ilustrações, podemos notar que a diretriz torna-se ótima antes dos valores convergirem para os corretos.



Erro quadrático X Número de iterações



Qualidade de diretriz X número de iterações

5 Iteração de Diretrizes

A iteração de diretrizes é um algoritmo para melhorar a escolha de diretrizes ótimas. Esse algoritmo consiste em 2 passos, começando a partir de uma política π_0:

- Avaliação da Diretriz: A partir da política π_i , calculamos $U_i = U^{\pi i}$;
- Melhoria da Diretriz: Utilizando U_i do resultado anterior aplicamos uma nova Diretriz π_{i+1};

Esse algoritmo termina quando a melhoria da diretriz não mudar entre um passo e outro. Nesse ponto sabemos que a função utilidade U_i é uma solução da função de Bellman e que π_i é uma diretriz ótima.

5.1 Avaliação de Diretrizes

A avaliação de diretrizes desse algoritmo é uma versão simplificada da equação de Bellman padrão. Isso ocorre porque ao invés do que ocorre na equação padrão em cada iteração i a política π_i só específica uma ação π_i (s) em um determinado estado s. Isso significa tirar da equação a parcela Max e transformar a mesma em uma equação linear como mostrado abaixo:

$$U_i(s) = R(s) + \gamma \sum_s M_{ij}^a U_i(s')$$

Essa nova equação tem $O(n^3)$, sendo muito eficiente para encontrar valores em problemas com poucos estados. Mas a complexidade fica proibitiva para problemas maiores. Por isso foi criada a Diretriz de Iteração modificada.

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} M_{ij}^{a} U_i(s')$$

Essa equação é aplicada k vezes para produzir a próxima estimativa de U_{i+1}, sendo muito mais eficiente que a função de iteração de valores original.

5.2 Melhoria da Diretriz

Bem mais simples de ser aplicada que a avaliação de diretrizes sendo só utilizar o algoritmo abaixo:

function POLICY-ITERATION(M, R) **returns** a policy
  inputs: M, a transition model
    R, a reward function on states
  local variables: U, a utility function, initially identical to R
    P, a policy, initially optimal with respect to U

  repeat
    $U \leftarrow$ VALUE-DETERMINATION(P, U, M, R)
    unchanged? $\leftarrow$ true
    for each state i **do**
      if $\max_a \sum_j M_{ij}^a U[j] > \sum_j M_{ij}^{P[i]} U[j]$ **then**
        $P[i] \leftarrow \arg\max_a \sum_j M_{ij}^a U[j]$
        unchanged? $\leftarrow$ false
    end
  until *unchanged?*
  return P

Figura 1- Algoritimo de Iteração de Diretrizes