Inteligência Artificial

Aula 8

Profª Bianca Zadrozny
http://www.ic.uff.br/~bianca/ia

Busca Competitiva

Capítulo 6 – Russell & Norvig
Seção 6.1, 6.2 e 6.3

Até aqui...

- Problemas sem interação com outro agente.
- O agente possui total controle sobre suas ações e sobre o efeito de suas ações.
- Muitas vezes encontrar a solução ótima é factível.

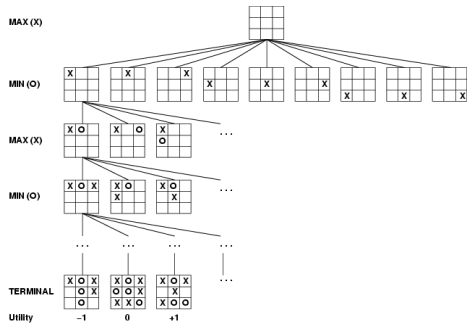
Jogos vs. busca

- O oponente é "imprevisível"
  - O agente tem que levar em consideração todos os movimentos possíveis de oponente.
- Limite de tempo
  - O agente tem que tomar uma decisão, mesmo que não seja ótima.

Decisões ótimas em jogos

- Consideraremos jogos com dois jogadores:
  - MAX e MIN
  - MAX faz o primeiro movimento e depois eles se revezam até o jogo terminar.
- Um jogo pode ser definido como um problema de busca com:
  - estado inicial
  - função sucessor (-> movimento, estado)
  - teste de término
  - função utilidade: dá um valor numérico para os estados terminais

Exemplo: Árvore de jogo (2 jogadores)



Do ponto de vista de MAX, valores altos de utilidade são bons.

Estratégias ótimas

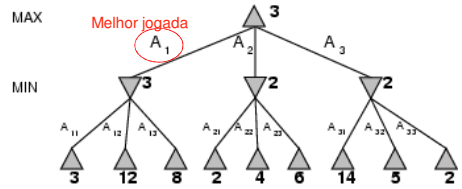
- A solução ótima para MAX depende dos movimentos de MIN, logo:
  - MAX deve encontrar uma *estratégia de contingência* que especifique o movimento de MAX no estado inicial, e depois o movimento de MAX nos estados resultantes de cada movimento de MIN e assim por diante.

Estratégias ótimas

- Dada uma árvore de jogo, a estratégia ótima pode ser determinada a partir do valor ***minimax*** de cada nó.
- O valor minimax (para MAX) é a utilidade de MAX para cada estado, ***assumindo que MIN escolhe os estados mais vantajosos*** para ele mesmo (i.e. os estado com menor valor utilidade para MAX).

Minimax

- Melhor estratégia para jogos determinísicos
- Idéia: escolher a jogada com o melhor retorno possível supondo que o oponente também vai fazer a melhor jogada possível
- Ex: Jogo simples, cada jogador faz um movimento



Valor minimax

VALOR-MINIMAX(n)=

$$\begin{cases} \text{UTILIDADE(n) se n é terminal} \\ \max_{x \in Succ(n)} \text{Valor Minimax(x) se n é um nó de MAX} \\ \min_{x \in Succ(n)} \text{Valor Minimax(x) se n é um nó de MIN} \end{cases}$$

Algoritmo minimax

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

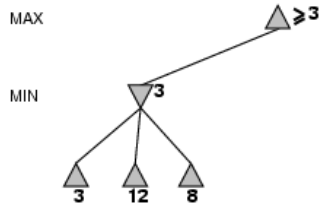

Propriedades do algoritmo minimax

- Equivale a uma busca completa em profundidade na árvore do jogo.
  - m: profundidade máxima da árvore
  - b: movimentos válidos em cada estado
- Completo? Sim (Se a árvore é finita)
- Ótimo? Sim (contra um oponente ótimo)
- Complexidade de tempo? $O(b^m)$
- Complexidade de espaço? O(bm)
- Para xadrez, b ≈ 35, m ≈100 para jogos "razoáveis"
  → solução exata não é possível

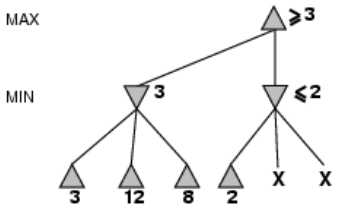
Poda α-β

- Algoritmo minimax: nº de estados do jogo é exponencial em relação ao nº de movimentos
- Poda α-β:
  - calcular a decisão correta sem examinar todos os nós da árvore,
  - retorna o mesmo que minimax, porém sem percorrer todos os estados.

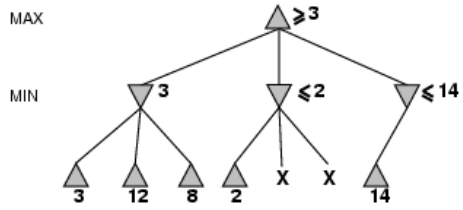
Poda α-β



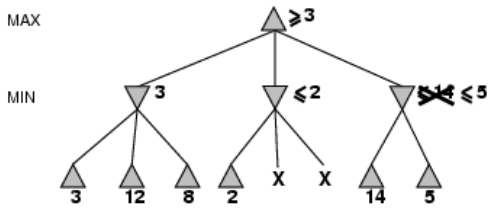
Poda α-β



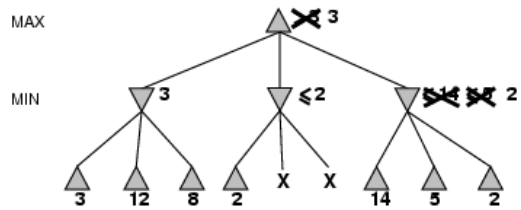
Poda α-β



Poda α-β



Poda α-β



Poda α-β

- A efetividade da poda α-β depende da ordem em que os sucessores são examinados.
- Com a melhor ordem possível a complexidade de tempo = $O(b^{m/2})$
  → dobra a profundidade da busca que conseguimos fazer

Por que "α-β" ?

- α é o valor da melhor escolha (valor mais alto) encontrado até então para qualquer ponto de escolha de MAX;
- Se *v* é pior do que α, MAX não percorrerá este caminho (irá podar este ramo de busca)
- β é definido de maneira análoga.