Inteligência Artificial

Aula 5

Profª Bianca Zadrozny

http://www.ic.uff.br/~bianca/ia-pos

Busca Competitiva

Capítulo 6 – Russell & Norvig

Seção 6.1 a 6.5

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.

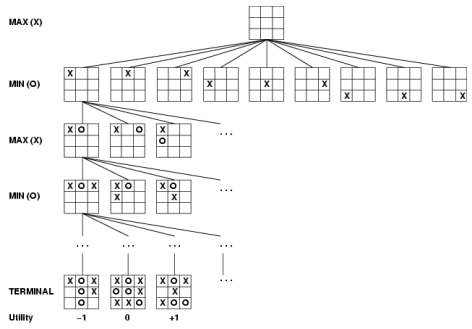
Jogos vs. busca

- 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

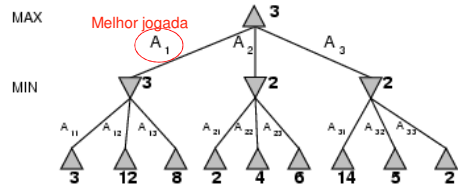
- A solução ótima para MAX depende dos movimentos de MIN, logo:
  - MAX deve encontrar uma *estratégia de contingência* que especifique o movimento de MAX no estado inicial, e depois o movimento de MAX nos estados resultantes de cada movimento de MIN e assim por diante.

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
```

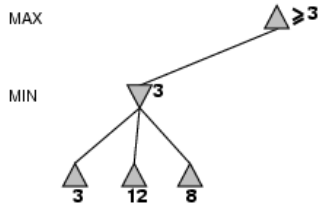

Propriedades do algoritmo minimax

- 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

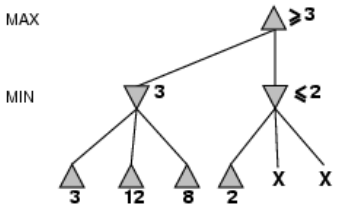
Poda α-β

- 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.

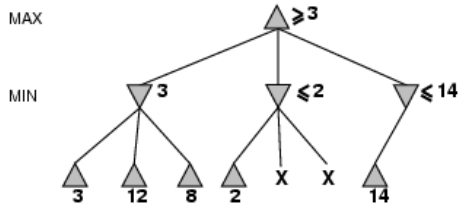
Poda α-β



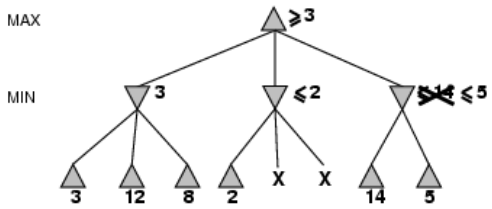
Poda α-β



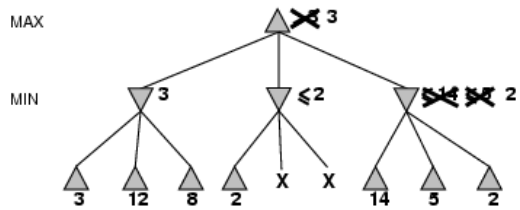
Poda α-β



Poda α-β



Poda α-β

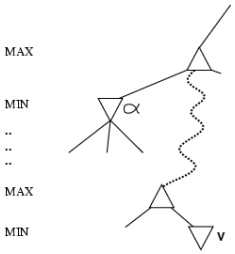


Poda α-β

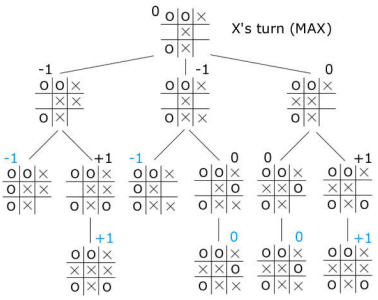
- A efetividade da poda α-β depende da ordem em que os sucessores são examinados.
- Com a melhor ordem possível a complexidade de tempo = $O(b^{m/2})$
  → dobra a profundidade da busca que conseguimos fazer

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.



Exemplo



Decisões imperfeitas em tempo real

- Minimax gera o espaço de busca todo.
- Poda α-β ainda tem que chegar até os estados terminais.

São muito ineficientes para jogos com muitos passos até os estados terminais isto é, quase todos os jogos interessantes!

Decisões imperfeitas em tempo real

- Sugestão (Shannon, 1950):
  - substituir a função utilidade por uma ***função de avaliação heurística*** e substituir o teste de término por um teste de *corte*.
  - Função de avaliação retorna uma *estimativa* da utilidade esperada do jogo a partir de uma dada posição.
  - isto é, nós não terminais se transformam em nós terminais para minimax ou poda α-β.

Decisões imperfeitas em tempo real

- ***Função de avaliação heurística***
  - Deve ordenar nós-terminais da mesma forma que a função utilidade;
  - A computação deve ser rápida;
  - Em estados não-terminais a função de avaliação deve estar relacionada com as chances reais de vitória;
    - o algoritmo será necessariamente incerto com relação aos resultados finais pois a busca será cortada!

Decisões imperfeitas em tempo real

- Definição de função de avaliação heurística: ***características de estado***
  - em conjunto definem *categorias* ou classes de *equivalência de estados* (ex: número de peões tomados);
    - Estados de cada categoria têm os mesmos valores para cada característica.
  - calcula contribuições numéricas separadas de cada característica e as combina para gerar um resultado final.

Decisões imperfeitas em tempo real

- Exemplo de características para de estado para **xadrez**:
  - valor material de cada peça: peão=1, cavalo ou bispo=3, torre=5, rainha=9
  - boa estrutura de peões, segurança do rei = 1/2 peão
- Função de avaliação: função linear ponderada das características

$Aval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$

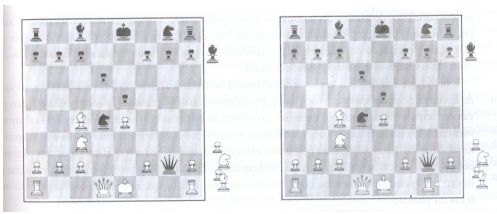
Decisões imperfeitas em tempo real

- Ao somar os valores de características assumimos que as contribuições de cada característica são independentes das outras.
  - *Ex: ignora o fato de um bispo ser mais valioso no fim do jogo quando tem mais espaço de manobra*
- É possível usar combinações não-lineares.
  - *Par de bispos pode valer mais que o dobro do valor de dois bispos.*

Decisões imperfeitas em tempo real

- Características e pesos não fazem parte das regras do jogo.
  - Foram aprendidos ao longo dos anos.
  - Pesos podem ser estimados usando técnicas de aprendizado automático.

Decisões imperfeitas podem levar a erros



- Suponha que a busca parou na profundidade em que as pretas tem vantagem de um cavalo e dois peões.
- No próximo movimento as brancas capturam a rainha e ganham o jogo.

Decisões imperfeitas podem levar a erros

- Solução: um corte mais sofisticado.
  - Busca quiescente: aplicar a função de avaliação somente em posições em que é improvável haver grandes mudanças de valores em estados futuros (posições quiescentes)
    - Exemplo no xadrex: posições em que podem ser feitas capturas não são quiescentes.

Decisões imperfeitas: efeito de horizonte

- Surge quando há um movimento do oponente que causa sérios danos e, em última instância, é inevitável.
  - Busca com profundidade fixa protela esse movimento "para além do horizonte da busca"
  - Pode ser minimizado com o uso de extensões singulares.
- Este problema torna-se menos comum conforme se aumenta a profundidade da busca.

Jogos Determinísticos na Prática

- Damas:
  - Chinook ganhou do campeão mundial Marion Tinsley in 1994.
    - Usa um banco de dados que define jogadas perfeitas para todas as configurações envolvendo 8 ou menos peças no tabuleiro, num total de 444 bilhões de posições.
- Xadrez:
  - Deep Blue ganhou do campeão mundial Garry Kasparov em 1997.
    - Busca 200 milhões de configurações por segundo, usa uma função de avaliação sofisticada, e métodos secretos para estender algumas linhas de busca até profundidade 40.
- Othello:
  - Campeões se recusam a jogar com computadores, porque eles são bons demais.
- Go:
  - Campeões se recusam a jogar com computadores, porque eles são ruins demais. No jogo Go, $b > 300$, então a maioria dos programas usa banco de dados de padrões para sugerir jogadas.

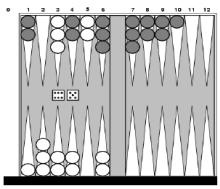
Jogos não-determinísticos

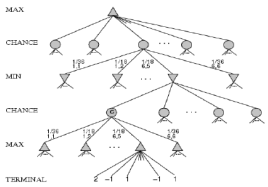
- Elemento aleatório proveniente de jogo de dados, sorteio de cartas, etc.
- Não-determinismo é inerente em ambientes reais.
  - O estudo de algoritmos para jogos com elemento aleatório é um passo em direção a algoritmos que podem ser aplicados no mundo real.

Jogos não-determinísticos

- Uma árvore de um jogo não-determinístico deve incluir **nós de acaso** além de nós minimax.
- Ramificações que saem dos nós de acaso denotam "resultados de dados possíveis" (anotadas com a probabilidade de cada mudança de estado).

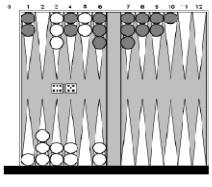
Exemplo: Gamão

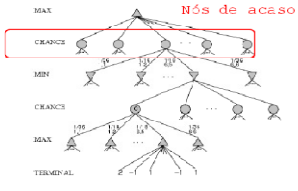




- Jogadas possíveis: (5→10,5→11), (5→11,19→24), (5→10,10→16) e (5→11,11→16).

Exemplo: Gamão





- [1,1],[6,6] tem probabilidade 1/36, todas as outras combinações têm probabilidade 1/18.
- Não é possível calcular o valor minimax exato, só o valor minimax **esperado**.

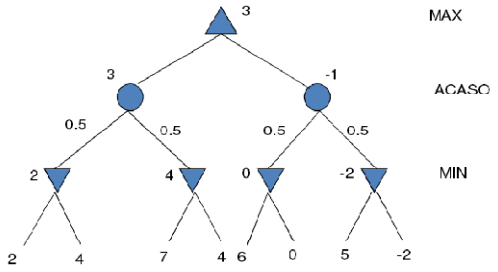
Valor de Expectiminimax

EXPECTIMINIMAX(n)=

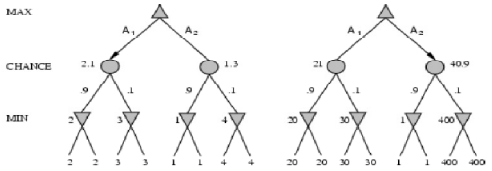
UTILIDADE(n)	Se n é terminal
$\max_{s \in sucessores(n)}$ EXPECTIMINIMAX(s)	Se n é um nó de MAX
$\min_{s \in sucessores(n)}$ EXPECTIMINIMAX(s)	Se n é um nó de MIN
$\sum_{s \in sucessores(n)} P(s)$. EXPECTEDMINIMAX(s)	Se n é um nó de acaso

- Essas equações podem ser aplicadas recursivamente até a raiz da árvore.

Exemplo: Expectiminimax



Jogos não-determinísticos com corte



- Neste caso, a escala dos valores é importante.
  - Esquerda, a jogada A1 é escolhida.
  - Direita, a jogada A2 é escolhida.
- A função de avaliação heurística tem que ser uma transformação positiva e linear da chance de ganhar o jogo a partir daquele ponto.

Complexidade de Expectiminimax

- Como considera todas as sequências de lançamentos de dados possíveis levará tempo O($b^m n^m$), onde n é o número de lançamentos distintos.
- É possível utilizar um tipo de poda alfa-beta se impusermos limites para os valores possíveis da função de utilidade.
  - A média pode ser limitada depois que alguns valores são conhecidos.