Método Minimax

Aula 5

Método Minimax

- O método Minimax é o método mais conhecido para lidar com jogos. Admite-se que existe um gerador de estados e uma função que avalia a vantagem ou desvantagem de um dado estado.
- Vamos considerar que esta função de avaliação é um estimador heurístico da estimativa das hipóteses de vitória do ponto de vista de um dos jogadores:
  - Quanto maior for o valor, maior serão as hipóteses do jogador vencer
  - Quanto menor for o valor, maior serão as hipóteses do oponente vencer
- O objectivo é tentar maximizar o valor dado pela função de avaliação.

Método Minimax

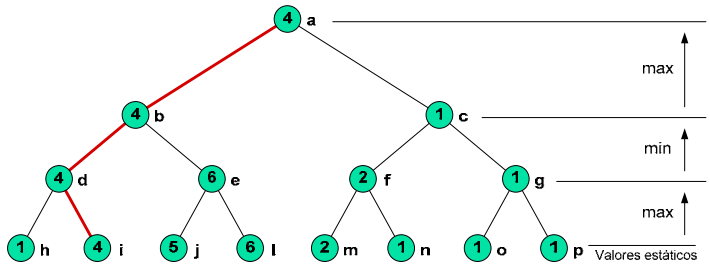
- O objectivo da implementação é seleccionar a jogada que garanta a melhor situação ao fim de n jogadas
- A melhor situação corresponde ao estado cujo valor da função de avaliação seja o maior (problema de maximização)
- O objectivo é alcançado propagando o valor correspondendo ao melhor estado até ao nó raiz; este valor corresponde ao ganho mínimo que se obtém se optarmos pela jogada correcta

Método Minimax

- A propagação de valores através dos nós da árvore é realizada da seguinte forma:
  - Nos níveis que correspodem às acções do jogador, selecciona-se para propagação o maior valor (nível de maximização)
  - Nos níveis que correspondem às acções do oponente, selecciona-se para propagação o menor valor (nível de minimização)
- Os valores associados aos nós folha são obtidos pela aplicação da função heurística de avaliação do mérito de cada um dos estados, de acordo com o ponto de vista do jogador

Método Minimax

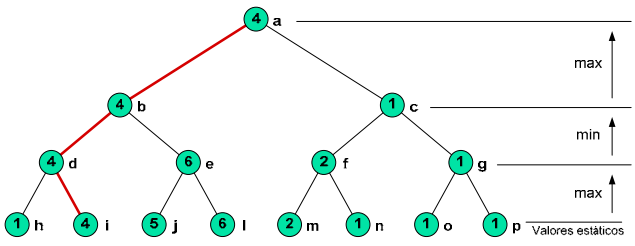
Consideremos o problema genérico representado na figura. Os nós representam estados e os ramos representam as jogadas possíveis a partir de cada estado. Os valores associados aos nós folha são obtidos por uma função de avaliação.



Método Minimax

Relações do problema genérico

```
max_to_move(d).
max_to_move(e).
max_to_move(f).
max_to_move(g).
min_to_move(b).
min_to_move(c).
min_to_move(h).
min_to_move(i).
min_to_move(j).
min_to_move(l).
min_to_move(m).
min_to_move(n).
min_to_move(o).
min_to_move(p).
```

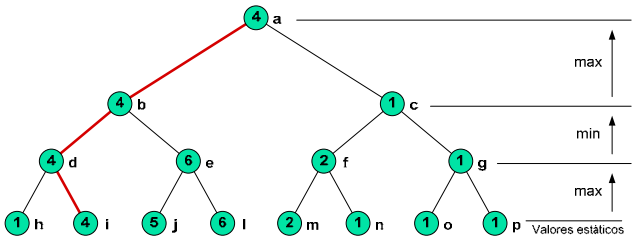


max_to_move/1 e ***min_to_move/1*** definem para cada nó qual o valor dos seus descendentes é propagado até ao próprio nó; o maior no caso de *max_to_move/1* ou o menor no caso de *min_to_move/1*

Método Minimax

Relações do problema genérico (cont.)

```
moves(a,[b,c]).
moves(b,[d,e]).
moves(c,[f,g]).
moves(d,[h,i]).
moves(e,[j,l]).
moves(f,[m,n]).
moves(g,[o,p]).
```

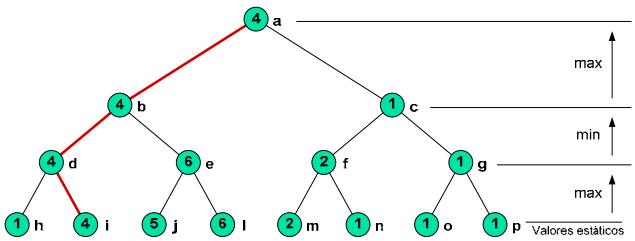


moves/2 define para cada nó a lista de nós descendentes (correspondem às jogadas válidas a partir de cada nó); num problema concreto a lista de nós descendentes é gerada em função das regras do jogo

Método Minimax

Relações do problema genérico (cont.)

```
static_eval(h,1).
static_eval(i,4).
static_eval(j,5).
static_eval(l,6).
static_eval(m,2).
static_eval(n,1).
static_eval(o,1).
static_eval(p,1).
```



static_eval/2 define o valor do mérito, do ponto de vista do jogador representado pelo algoritmo, de cada estado terminal (nós folha da árvore); num problema concreto, este valor é estimado através de uma função de avaliação

Método Minimax

Algoritmo

O objectvo do algoritmo é propagar o valor *minimax* para uma dada posição a partir dos nós folha.

O predicado principal é

minimax(Pos, BestPos, Val, NodesList)

onde **Val** é o valor minimax da posição **Pos**, **BestPos** é a melhor posição que sucede a **Pos** (ou seja, a jogada a realizar para alcançar o valor **Val**) e **NodesList** contém a sequência de nós da árvore usados para propagar o valor de uma posição final até **Pos**.

Método Minimax

```
minimax(Pos, BestPos, Val, NodesList):-
        moves(Pos, PosList), !,
        best(PosList, BestPos, Val, NodesList).
```

moves/2 falha se **Pos** é uma folha

best/4 selecciona a "melhor" posição **BestPos** a partir de uma lista de posições candidatas **PosList** (descendentes de **Pos**). **BestVal** é o valor de **BestPos** e logo também de **Pos**. A melhor posição pode ser um máximo ou um mínimo, dependendo do nível de **Pos**.

```
minimax(Pos, BestPos, Val, []):-
        static_eval(Pos, Val).
```

Pos é uma folha

Método Minimax

Algoritmo (cont.)

Quando a lista de candidatos contém apenas um elemento, ***minimax/4*** é usado para propagar o melhor valor a partir dos nós folha que descendem de **Pos**

```
best([Pos], Pos, Val, NodesList):-
   minimax(Pos, _, Val, NodesList), !.
best([Pos1|PosList], BestPos, BestVal, [BestPos|NodesList]):-
   minimax(Pos1, _, Val1, NodesList1),
```

Val1 é o melhor valor para **Pos1**

```
   best( PosList, Pos2, Val2, NodesList2),
```

Val2 é o valor de **Pos2**, sendo esta a melhor posição entre as posições contidas em **PostList**

```
   betterof(Pos1, Val1, Pos2, Val2, BestPos, BestVal,
       NodesList1, NodesList2, NodesList).
```

BestPos é a melhor posição entre **Pos1** e **Pos2**

Método Minimax

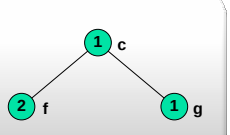
Algoritmo (cont.)

Regras de propagação dos valores dos nós:

v(P) – valor da função de avaliação aplicada ao estado P (nó folha)
V(P) – valor propagado para o estado P a partir dos estados descendentes de P

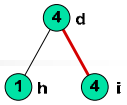
$V(P) = v(P)$	se P é um estado que ocupa um nó folha
$V(P) = \max_i V(P_i)$	se P é um estado relativo a um nível de maximização
$V(P) = \min_i V(P_i)$	se P é um estado relativo a um nível de minimização

```
max_to_move(f).
max_to_move(g).

?- betterof(f, 2, g, 1, BestPos, Val, _, _, _).
BestPos = g
Val = 1
```




```
betterof(Pos0, Val0, Pos1, Val1, Pos0, Val0, NodesList0, NodesList1, NodesList0):-
   min_to_move(Pos0), Val0 > Val1, !.
betterof(Pos0, Val0, Pos1, Val1, Pos0, Val0, NodesList0, NodesList1, NodesList0):-
   max_to_move(Pos0), Val0 < Val1, !.
betterof(Pos0, Val0, Pos1, Val1, Pos1, Val1, NodesList0, NodesList1, NodesList1).
```

Método Minimax



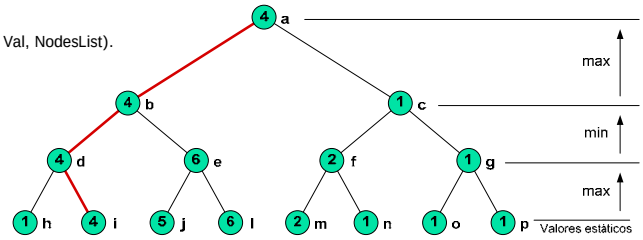

```
?- minimax(d, BestPos, Val, NodesList).
(1)minimax(d, BestPos, Val, NodesList):-
     moves(d, PosList),                                  PosList=[h,i]
     best([h,i], BestPos, Val, NodesList).
       (1)Falha
       (2)best([h|[i]], BestPos, Val, [BestPos|NodesList]):-
              minimax(h, _, Val1, NodesList1),           Val1=1
                                                         NodesList1=[]
            (1)Falha
            (2)minimax(h, _, Val, []):-
                 static_eval(h, Val).        Val=1       Pos2=i
                                                         Val2=4
              best([i], Pos2, Val2, NodesList2),         NodesList2=[]
            (1)best([i], i, Val, NodesList):-
                    minimax(i, _, Val, NodesList), !.    Val=4
                                                         NodesList=[]
                      (1)Falha
                      (2)minimax(i, _, Val, []):-
                           static_eval(i, Val).          Val=4

              betterof(h,1,i,4,BestPos,Val,[],[],NodesList).
                                                         BestPos=i
                                                         Val=4
                                                         NodesList=[]
     BestPos=i
     Val=4
     Nodeslist=[i]
```


Método Minimax

Resultado

?- minimax(a, BestPos, Val, NodesList).

BestPos = b
Val = 4
NodesList = [b, d, i]



Implementação do método Minimax com cortes Alpha-Beta:
Prolog Programming for Artificial Intelligence
Ivan Bratko
Addison-Wesley