Jogos com Oponentes

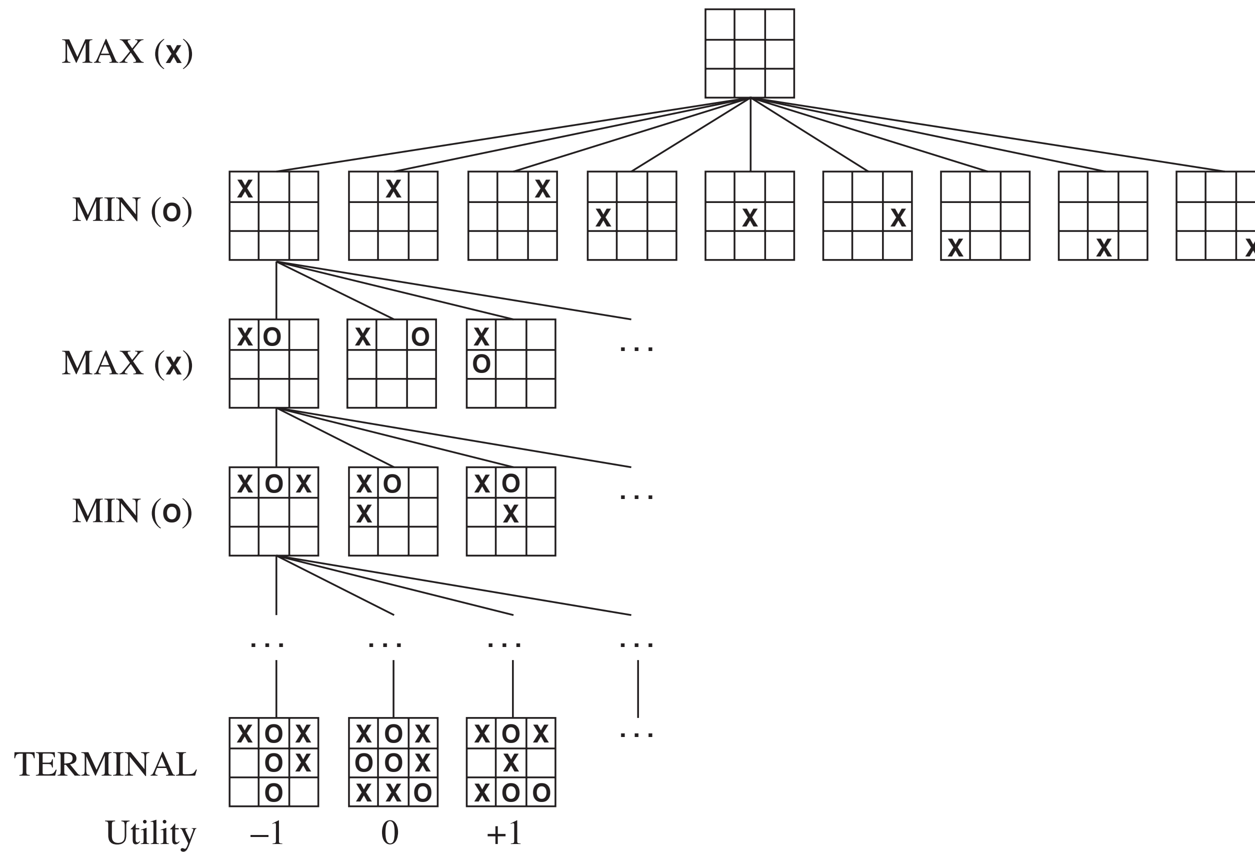
- Problemas de busca: não assumem a presença de um oponente
- Jogos: oponente $\rightarrow$ INCERTEZA!
- Incerteza porque não se conhece as jogadas exatas do oponente e não por causa de falta de informação.
- Jogos são diferentes de busca por duas razões principais:
  - espaço de busca muito grande
  - tempo para cada jogada
- Ex: xadrez, em média fator de ramificação = 35
- em média, 50 jogadas para cada jogador, onde de cada jogada devemos selecionar 1 de 35 $\rightarrow 35^{100}$ nós!!!

Jogos

- Um jogo pode ser definido como um tipo de problema de busca com os seguintes componentes:
  - estado inicial: configuração do tabuleiro e indicação de quem é a vez
  - função para gerar os sucessores: retorna uma lista de pares com jogadas possíveis e estado resultante (a,s no algoritmo)
  - teste de terminação
  - Função *utilidade* (utility function) que devolve o valor numérico do jogo. Ex: ganhou, empatou, perdeu.
- Jogo com dois jogadores: MIN e MAX
- MAX inicia o jogo

Exemplo: jogo do galo

(velha - BR ou tic-tac-toe - US ou noughts and crosses - UK)



Exemplo: jogo do galo (fonte: Nilsson)

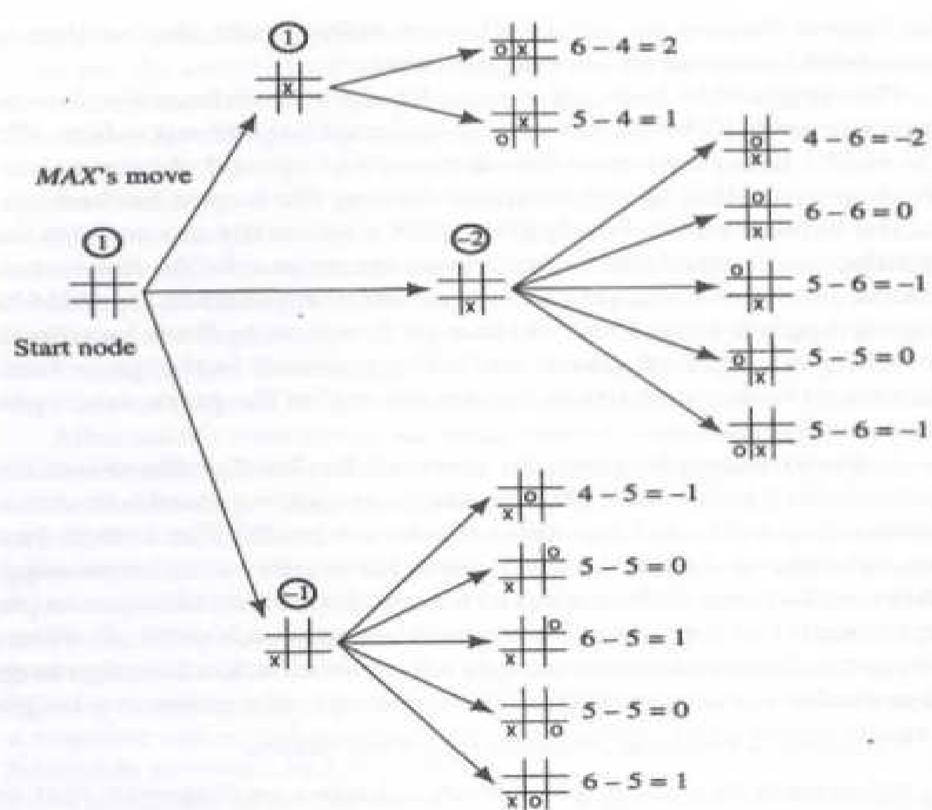


Figure 12.3
The First Stage of Search in Tic-Tac-Toe

Observações:

- fator de ramificação dos nós mais próximos da raiz pode ser reduzido removendo estados simétricos
- fator de ramificação dos nós mais profundos da árvore de procura vai automaticamente sendo reduzido (sem necessidade de verificação de simetrias, porque o número de posições vagas no tabuleiro diminui)
- se expandir todos os nós até o último nível da árvore, a função utilidade pode ser: 0 para empate, -1 para perdeu e +1 para ganhou
- se não expandir os nós até o último nível, pode utilizar a função utilidade que devolve o número de fileiras vazias para o MAX menos o número de fileiras vazias para o MIN.

Jogos: Minimax

MINIMAX-VALUE(n) =

$$\begin{cases} UTILITY(n) & \text{if } n \text{ is a terminal state} \\ max_{s \in \text{successors(n)}} \text{MINIMAX-VALUE(s)} & \text{if } n \text{ is a MAX node} \\ min_{s \in \text{successors(n)}} \text{MINIMAX-VALUE(s)} & \text{if } n \text{ is a MIN node} \end{cases}$$

Jogos: Algoritmo Minimax

```
function MINIMAX_DECISION(state) returns an action
   inputs: state -> estado corrente no jogo
   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 <- -infinito
   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 <- infinito
   for a, s in SUCCESSORS(state) do
     v <- MIN(v, MAX-VALUE(s))
   return v
```

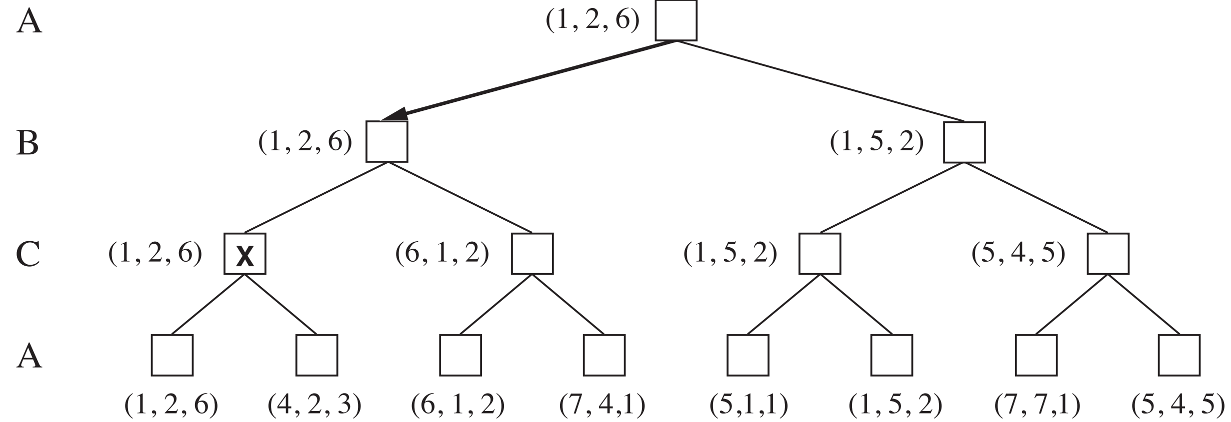
Jogos: Algoritmo Alfa-Beta

```
function ALPHA-BETA-SEARCH(state) returns an action
   inputs: state -> estado corrente no jogo
   v <- MAX-VALUE(state,-inf,+inf) % alfa inicia com -inf,
                                   % beta inicia com +inf
   return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state,alfa,beta) returns a utility value
   inputs: state -> estado corrente no jogo
           alfa  -> valor da melhor alternativa para MAX
           beta  -> valor da melhor alternativa para MIN
   if TERMINAL_TEST(state) then return UTILITY(state)
   v <- -infinito
   for a, s in SUCCESSORS(state) do
     v <- MAX(v, MIN-VALUE(s,alfa,beta))
     if (v >= beta ) then return v % momento da poda
     alfa <- MAX(alfa,v)
   return v

function MIN-VALUE(state,alfa,beta) returns a utility value
   inputs: state -> estado corrente no jogo
           alfa  -> valor da melhor alternativa para MAX
           beta  -> valor da melhor alternativa para MIN
   if TERMINAL_TEST(state) then return UTILITY(state)
   v <- +infinito
   for a, s in SUCCESSORS(state) do
     v <- MIN(v, MAX-VALUE(s,alfa,beta))
     if (v <= alfa ) then return v % momento da poda
     beta <- MIN(beta,v)
   return v
```

Exemplo: jogos com múltiplos jogadores (fonte: Russell)



Exemplo: jogos com múltiplos jogadores

Observações:

- representação de valores para cada jogador em forma de vetor (tuplas), onde cada elemento indica a função utilidade para um jogador, na ordem de jogada
- algoritmo aplicado similar ao minimax
- complicações:
  - mudanças dinâmicas de estratégia durante o jogo quando acontecem alianças entre jogadores para derrubar o jogador mais forte e mais tarde estas alianças desaparecem (jogo flutua entre cooperação e competição)
  - quando no jogo é introduzido algum fator aleatório (jogos com utilização de dados, por exemplo)