Game Playing: MiniMax

Juliana Kaizer Vizzotto

Universidade Federal de Santa Maria

Disciplina de Inteligência Artificial

Roteiro

- Introdução
- MiniMax: *perfect decision in two person games*

Game Playing

- Uma das áreas mais antigas de IA: 1950 Claude Shannon (invetor da teoria da informação) e Alan Turing escreveram os primeiros programas de xadrez.
- Pq o xadrez?
  - Requer inteligência!
  - Regras simples!
  - Jogo de *informação perfeita*: estado do *mundo* é completamente acessível ao programa.
  - Fácil representar o jogo como uma busca no espaço de posições possíveis.

Game Playing

- A presença de um oponente torna o problema de decisão um pouco mais complicado.
- O oponente introduz *incerteza*, pois não se sabe o que ele irá fazer!
- **Problemas de Contingência**.
  - Em geral, cada ramo da árvore lida com alguma *contingência* que pode acontecer.
  - Requer alguma "sensibilidade" durante a fase de execução.
  - A cada passo deve-se calcular toda a árvore de ações, ao invés de uma única sequência de ações.

Game Playing

- Em geral, **jogos** são problemas bem difíceis de resolver :)
- Xadrez, por exemplo, tem em média um fator de ramificação de 35, e jogadas frequentemente contemplam 50 movimentos para cada jogador: a árvore de busca tem mais ou menos 35^{100} nós. Embora existam somente 10^40 posições legais diferentes.
- A complexidade de jogos introduz um novo tipo de incerteza!
- A incerteza não é porque temos informação perdida, mas pq não temos tempo para calculuar a sequência exata de qualquer movimento.

Game Playing

- Ao invés disso, deve-se fazer uma **boa tentativa** baseando-se em experiências passadas.
- Vamos começar nosso estudo analisando como encontrar teoricamente o *melhor* movimento.
- Escolher um bom movimento quando tempo é limitado!
- **Pruning** possibilita que ignoremos porções da árvore de busca que não fazem diferença na escolha final.
- **Avaliação Heurística**: possibilita uma aproximação da utilidade real de um estado sem fazer uma busca completa.

Perfect Decision in Two-Person Games

- Jogos com dois jogadores: MAX e MIN
- MAX move primeiro, e então eles fazem as rodadas até que o jogo termine.
- No final do jogo, o vencedor ganha pontos (ou perdedor é penalizado).

Perfect Decision in Two-Person Games

- Um jogo pode ser formalmente definido como um tipo de problema de busca com os seguintes componentes:
  - **Estado Inicial**: inclui a posição no tabuleiro e uma indicação de que movimento pode-se fazer.
  - **Conjunto de Operadores**: define os movimentos que um jogador pode fazer.
  - **Teste Terminal**: determina quando o jogo terminou.
  - **Função de Utilidade** (*payoff function*): fornece um valor numérico para a saída de um jogo. Em xadrez, a saída é ganhou, perdeu ou empatou, a qual podemos representar por +1, -1 ou 0.

Perfect Decision in Two-Person Games

- Se esse fosse um problema de busca normal, então o que todos os MAX deveriam fazer é procurar por uma sequência de movimentos que levem a um estado final (de acordo com a função de utilidade), e então fazer o primeiro movimento da sequência.
- Infelizmente, nesse caso, MIN tem algo a dizer sobre isso :)
- Então, MAX precisa encontrar uma **estratégia** que levará a um estado final vencedor **sem ser afetado por aquilo que MIM faz**.
- Essa estratégia inclui o movimento correto de MAX para cada movimento possível de MIN.

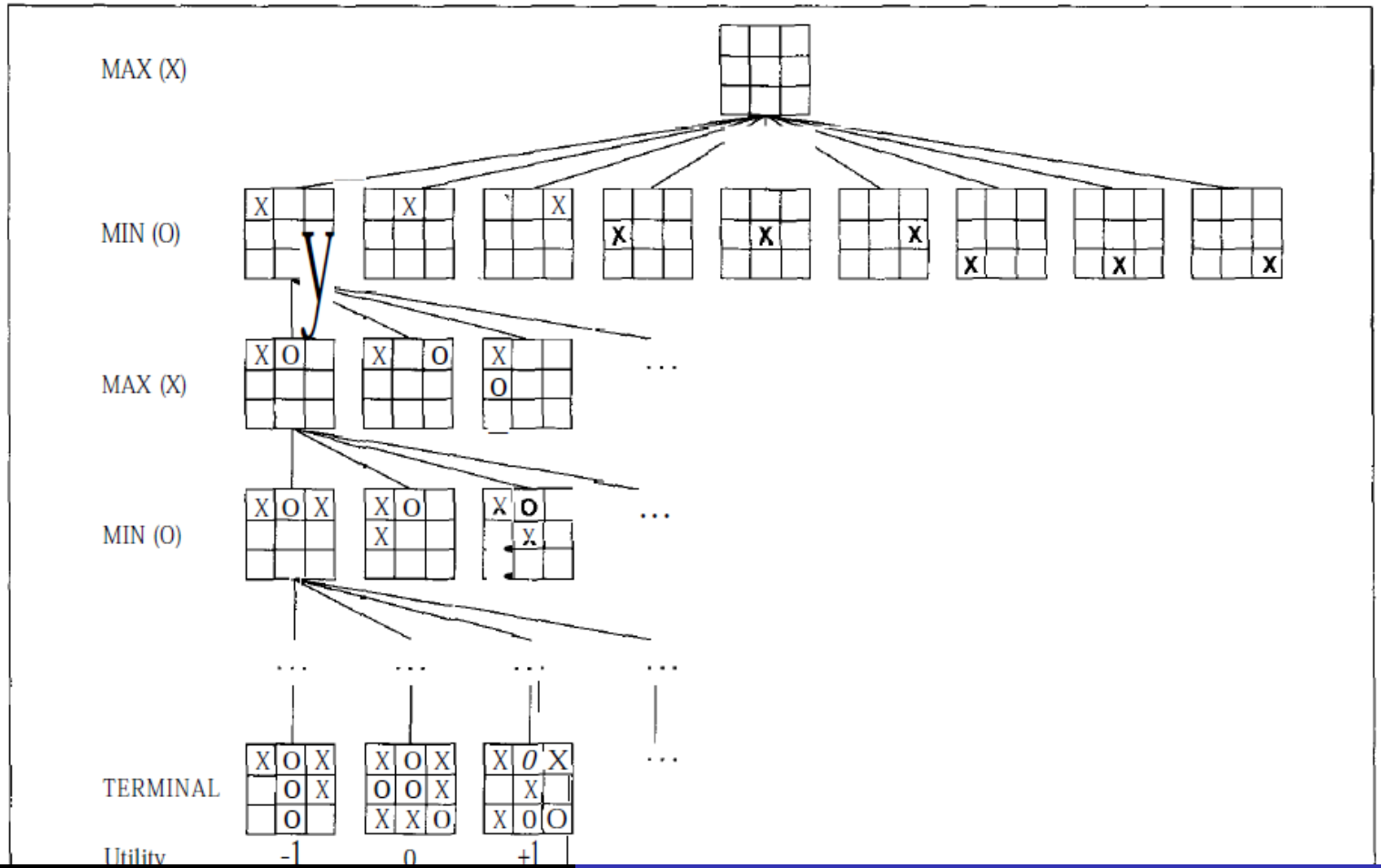
Perfect Decision in Two-Person Games

- Começamos estudando um estratégia ótima, mesmo que normalmente não tenhamos tempo suficiente para computá-la.

Perfect Decision in Two-Person Games

- **Exemplo Jogo da Velha**
- No estado inicial MAX tem uma escolha entre 9 movimentos possíveis.
- O jogo alterna entre MAX marcando um X e MIM colocando O até que chegaremos em um estado final.
- O número em cada nodo folha indica o **valor de utilidade** de cada estado terminal do ponto de vista de MAX.
- Valores atos são bons para MAX e ruins para MIN.
- O Trabalho de MAX é usar a árvore de busca (particularmente a utilidade dos estados terminais) para determinar o melhor movimento.

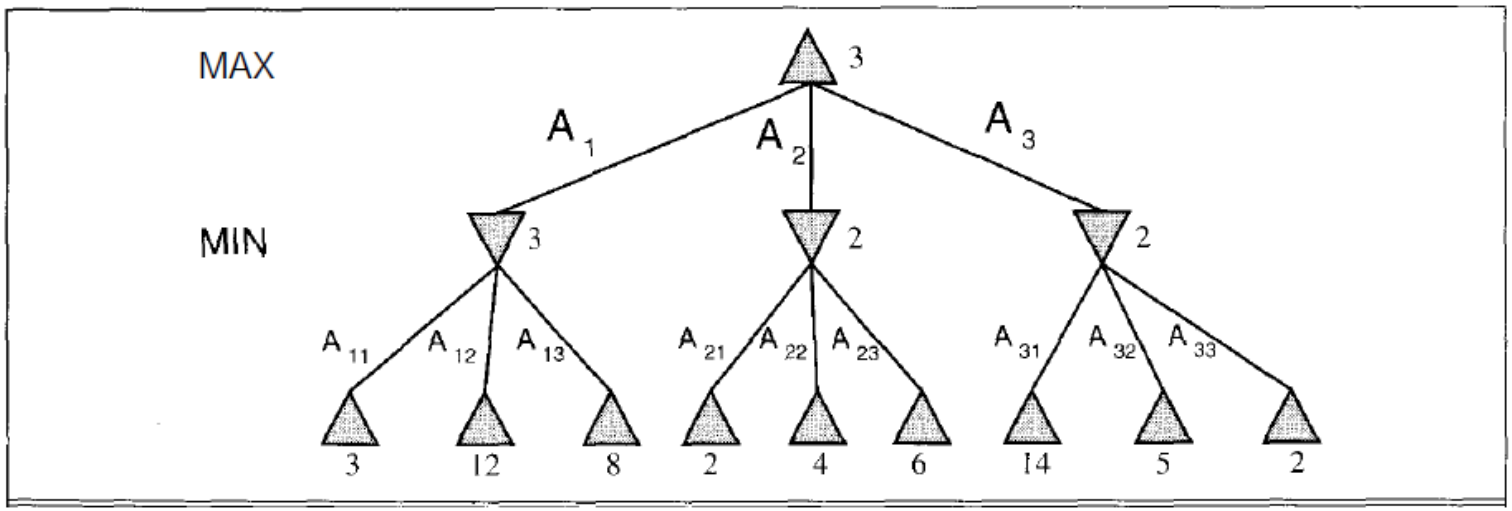
Perfect Decision in Two-Person Games



Perfect Decision in Two-Person Games

- Mesmo um jogo simples como o jogo da velha é muito complexo mostar toda a árvore de busca.
- Para ilustração vamos considerar um jogo trivial.
- Os possíveis movimentos de MAX são: A_1, A_2 e A_3.
- As possíveis respostas de MIN para A_1 são A_{11}, A_{12}, A_{13}, e assim por diante.

Perfect Decision in Two-Person Games



Perfect Decision in Two-Person Games

- O algoritmo **minimax** é projetado para determinar a estratégia ótima para MAX, e então decidir qual é o melhor primeiro movimento.

Algoritmo minimax

- Gere toda a árvore de jogo, até os estados terminais.
- Aplique a função de utilidade em cada estado terminal para obter seu valor.
- Use a função utilidade dos estados terminais para determinar a utilidade dos nodos um nível acima na árvore de busca.
- No exemplo acima, considere os nodos folhas mais à esquerda.
- No nodo MIN acima deles, MIN tem a opção de se mover e o melhor que MIN pode fazer é escolher A_{11}. que leva ao terminal mínimo 3.
- Então mesmo que a função de utilidade não seja diretamente aplicável a este nodo, podemos atribuir o valor de utilidade 3, considerando que MIN vai escolher o melhor.
- Continue colocando valores nos nodos dessa forma até a raiz, um nível de cada vez.

Algoritmo minimax

- Eventualmente, os valores chegam no topo da árvore, nesse ponto MAX escolhe o movimento que leva ao valor mais alto.
- Note o valor 3 no topo da árvore do exemplo trivial.
- **Esse método é chamado decisão minimax**, pois ele maximiza o valor de utilidade sob a condição de que o oponente jogará perfeitamente para minimizá-lo.

Algoritmo minimax

```
function MINIMAX-DECISION(game) returns an operator
   for each op in OPERATORS[game] do
     VALUE[op] <- MINIMAX-VALUE(APPLY(op,game),game)
   end
return the op with the highest VALUE[op]
----------
function MINIMAX-VALUE(state, game) returns a utility value
   if TERMINAL-TEST[game](state)
      return UTILITY[game](state)
   else if MAX is to move in state then
           return the highest MINIMAX-VALUE of SUCESSORS(st
         else
           return the lowest MINIMAX-VALUE of SUCESSORS(sta
```