INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL
22-23

CAP. 8 ALGORITMOS PARA JOGOS

Carlos Pereira
ISEC

Índice

- Índice
  - Introdução
  - MiniMax
  - Alpha-Beta Pruning
  - Outras abordagens
    - Funções de Avaliação
    - Jogos com Elemento Sorte

Introdução

3

- Algoritmos para Jogos
  - Os jogos diferenciam-se pela inclusão de um <u>factor de incerteza devido à presença de um adversário</u>
    - Incerteza do tipo não probabilística
      - O adversário (B) tentará a melhor jogada para ele, o que implica a pior jogada para o oponente (A).
      - A aplicação de algoritmos de pesquisa para encontrar a melhor solução para A não funciona! Pois é necessário contar com os movimentos de B.

Introdução

- …
  - Tipos de jogos

	Determinístico	*Não Determinístico (factor sorte)*
Observável	Xadrez Damas …	Monopólio Gamão …
Parcialmente Observável	Batalha Naval …	Cartas …

    - O Minimax aplica-se a jogos determinísticos e observáveis

Introdução

5

- ...
  - Os jogos constituem <u>problemas complexos</u> e de difícil resolução.
    - O exemplo mais conhecido é **o jogo de xadrez**:
      - Fator de ramificação b≈35
      - Em média desenrola-se ao longo de 50 lances por jogador.
      - A árvore de pesquisa (completa) teria **35^{100}** nós!

Introdução

6

- ...
  - Perspetiva Histórica
    - O HITECH foi o primeiro sistema a vencer um mestre de xadrez.
      - gera cerca de 10 milhões de estados antes de decidir um movimento.
    - O Deep Thought 2 foi implementado pela IBM em parceria com Carnegie Mellon University (CMU).
      - Situava-se ente os 100 melhores jogadores humanos.
    - Deep Blue (IBM) gera cerca de 100 a 200 biliões de posições por movimento.
      - http://www.research.ibm.com/deepblue/
      - Em 1997, venceu o Kasparov.

Introdução

- …
  - X3D Deep Fritz
    - Em 2006 venceu o campeão do mundo Vladimir Kramik
    - 2 milhões de movimentos por segundo!
    - Ambiente Multi-processador
      - https://www.chessprogramming.org/Kasparov_versus_X3D_Fritz_2003

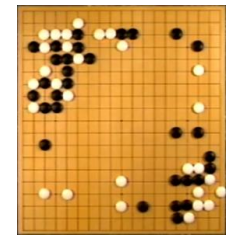
Introdução

- Jogo de Damas
  - 1950,Strachey, M.A., National Research Development Corporation,
  - 1956 Arthur Samuel, IBM
  - 1990, Chinook
    - Derrotou o campeão mundial
      - http://webdocs.cs.ualberta.ca/~chinook/
    - "Checkers is solved"
      - Um jogo sem erros conduz a empate!
        - http://www.sciencemag.org/content/317/5844/1518.abstract?keytype=ref&siteid=sci&ijkey=jVmVcXy2%2FNTnY

Introdução

- Jogo Go
  - Características do jogo
    - Inventado na China, 2000 AC. Tabuleiro 19×19, alternadamente, os jogadores vão colocando peças nas intersecções
    - Objetivo: Cercar as peças do oponente
  - Ainda mais difícil que Xadrez!
    - Avaliação de posições é difícil e o número de jogadas possível é muito elevado. O número de posições é superior ao número de átomos no universo observável $= 10^{80}$.
  - AphaGo
    - O primeiro a derrotar um campeão mundial de Go.

https://www.deepmind.com/research/highlighted-research/alphago



MiniMax

10

- Algoritmo Minimax
  - Dois jogadores, designados por **MAX** e **MIN**
    - Jogam alternadamente - MAX joga primeiro
    - No final do jogo pode acontecer:
      - MAX ganha (MIN perde)
      - Max perde
      - Empatam
        - Por exemplo no xadrez a avaliação pode ser +1 (MAX ganha), -1 (MAX perde) ou 0 (empatam). Noutros jogos a avaliação pode ser traduzida por pontos (cartas, etc..)
    - Baseia-se no princípio de "seleção da melhor jogada por parte de cada jogador"

MiniMax

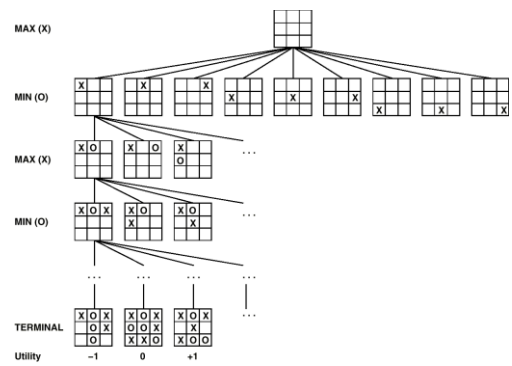
11

- ...
  - Definição formal do jogo
    - Estado Inicial: Posição inicial, valor das "peças" e indicação de quem inicia o jogo.
    - Operadores(Ações): Funções que definem as jogada permitidas
    - Teste de Final: Função que determina se o jogo acabou.
      - Os estados atingidos no final do jogo designam-se por Estados Terminais
    - Função de Utilidade: mede o "proveito" que o estado terminal alcançado representa para cada um dos jogadores.

MiniMax

12

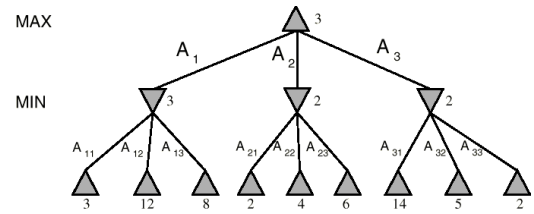
- Como resolver o problema?
  - https://scratch.mit.edu/projects/133865314
  - Construção da Árvore do jogo –
    - Exemplo para o jogo do galo



MiniMax

13

- ...
  - Considere-se um jogo que termina ao fim de duas meias-ações - Uma para MAX e outra para MIN.
    - A árvore correspondente poderá ser:

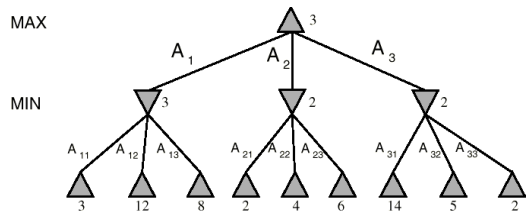


    - MAX joga A1, A2 ou A3
    - MIN joga A11, A12,...,A33
    - Estão representados a avaliação dos estados (na perspetiva de MAX)
  - Qual a melhor estratégia?

MiniMax

14

- ...



  - MAX, para decidir se joga A1, A2 ou A3, deve conhecer previamente os respetivos valores, o que implica:
    - Determinar os valores de todos os estados terminais
    - Partir do princípio de que MIN jogará de forma a prejudicar MAX
  - **Neste caso, qual a melhor jogada?**

MiniMax

15

- ...

1. Gerar a árvore do jogo
2. Determinar a Utilidade de cada estado terminal (valor para MAX)
3. Progredir para o nível anterior (neste nível é MIN que joga)

   A cada nó assinalar o valor mínimo dos nós seus filhos (isto traduz que MAX espera que MIN jogue de modo a minimizar a pontuação de MAX)
4. Progredir para o nível anterior (neste nível é MAX que joga):

   A cada nó assinalar o valor máximo dos nós seus filhos (isto traduz que MAX jogará da melhor forma)
5. Prosseguir assim até ser atingida a raiz da árvore.

MiniMax

16

- ...
  - Algoritmo (de Decisão) MiniMax:

```
function MINIMAX-DECISION(state) returns an action
  inputs: state, current state in game
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

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
```

MiniMax

17

- ...
  - A árvore é toda construída inicialmente.
    - Toda a árvore é percorrida
    - Travessia em profundidade
  - Algoritmo recursivo
    - Atribuição de valores é feita dos nós terminais para a raiz
  - Impraticável para jogos complexos
    - Porquê?

Alpha-Beta Pruning

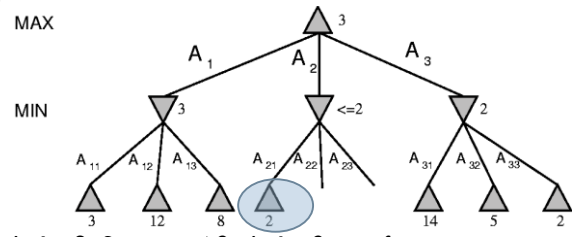
18

- Recursos: Tempo e Memória
  - O algoritmo Minimax necessita de memória e tempo consideráveis, mesmo para jogos relativamente simples!
  - Será viável aplicar o Minimax ao jogo de xadrez?
    - Num torneio de xadrez, cada jogada demora, em média, 150 segundos.
    - Com recursos razoáveis, admitamos que será possível pesquisar cerca de 1000 posições por segundo, o que implica 150.000 posições por jogada.
    - Assim, o programa só teria tempo para avaliar 4 meias-jogadas, o que corresponde a um nível de principiante! (com um factor de ramificação de 35, 35^4=1.500.625)

Alpha-Beta Pruning

19

- ...
  - Será possível obter a decisão correcta sem ter de gerar toda a árvore?



  - Se jogar A1 obtém 3. Se jogar A2 obtém 2 ou inferior
    - A valorização de A22 e A23 deixou de interessar!!
    - Considerando que o algoritmo é recursivo, a avaliação de todos os possíveis descendentes de A22 e A23 deixa também de ser necessária.

Alpha-Beta Pruning

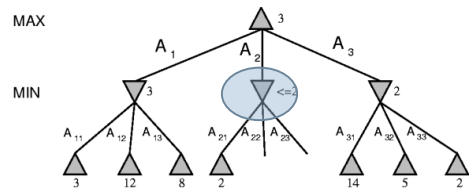
20

- ...
  - O algoritmo baseia-se na utilização de dois parâmetros, "Alpha" e "Beta":
    - Alpha: representa o valor mínimo garantido que MAX poderá obter.
      - Como representa um limite inferior é inicializado a -∞ e vai crescendo, sendo actualizado num nó MAX.
    - Beta: representa o valor máximo que MIN consegue impor a MAX
      - MAX nunca conseguirá jogar para obter um valor superior a beta
      - Sendo um limite superior, é inicializado a +∞ e posteriormente vai decrescendo (actualizado num nó MIN)

Alpha-Beta Pruning

21

- ..



  - Alpha=3
    - provém da visita já realizada ao ramo esquerdo da árvore, que provou que MAX, optando por A1, pode obter o valor 3 (mesmo que MIN jogue o melhor possível)
  - Beta=2
    - provém da análise dos filhos do nó atingível por A2, que prova que MIN pode impor um limite de 2 (ou eventualmente inferior, dependendo de A22 e A23)

Alpha-Beta Pruning

22

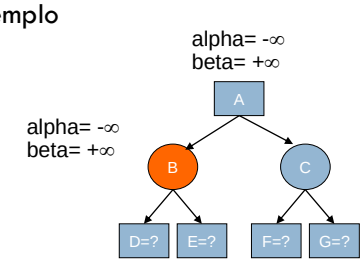
- ...

```
function ALPHA-BETA-DECISION(state) returns an action
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  same as MAX-VALUE but with roles of α, β reversed
```

Alpha-Beta Pruning

23

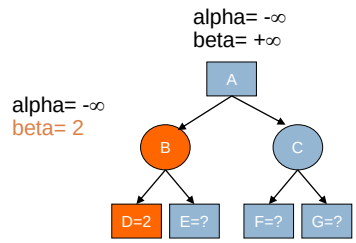
- ...
  - Ao atingir-se um nó em que alfa >= beta, pode cortar-se o ramo
    - Exemplo



Alpha-Beta Pruning

24

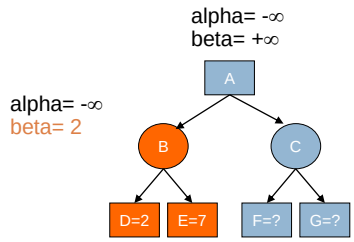
- ...



Alpha-Beta Pruning

25

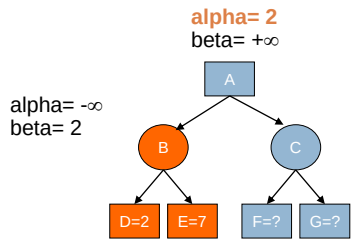
- ...



Alpha-Beta Pruning

26

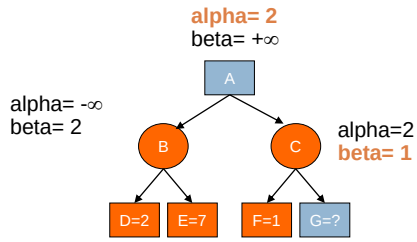
- ...



Alpha-Beta Pruning

27

- …



- **O ramo G pode ser excluído – processo de pruning**
  - porque qualquer que seja o seu valor, MAX nunca optará pelo ramo C!

Alpha-Beta Pruning

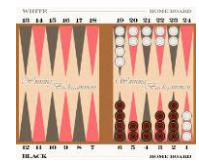
- …
  - Características
    - Algoritmo **ótimo**
      - A estratégia sugerida é igual à que seria incutida pelo MiniMax (sem pruning)
    - A Eficácia do algoritmo depende da ordem pela qual os sucessores são avaliados

Funções de Avaliação

- Corte e Funções de Avaliação
  - Outra forma de evitar a expansão completa da árvore de jogo consiste no seguinte processo:
    - Expandir a árvore até um limite pré-determinado
    - Avaliar cada uma das folhas (usando uma heurística – função de avaliação)
    - Proceder como no MiniMax
      - os valores dados pelas funções de avaliação são retornados como se as folhas correspondessem a estados terminais.
  - Retorna uma estimativa, de natureza heurística, da função utilidade
    - Problema: Qual a heurística mais adequada?

Jogos com Elemento Sorte

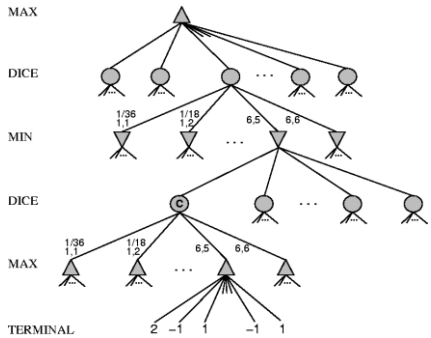
- Elemento Sorte
  - Muitos jogos contêm o elemento sorte.
    - Exemplo:
      - Gamão: O jogador lança dois dados e a jogada é efetuada em função do resultado do lançamento.



    - Será possível aplicar o algoritmo Minimax também nestes jogos?
      - Sim, contudo a **árvore deverá incluir também nós que traduzam o fator sorte**.

Jogos com Elemento Sorte

- …



Os nós "sorte" alternam com "Max" e "Min"

- Representam-se os valores de probabilidade de ocorrência:
  - p=1/36 para os lançamentos duplos (1,1), (2,2), …, (6,6)
  - p=1/18 para os restantes

Jogos com Elemento Sorte

- …
  - Algoritmo:
    - Calcula-se a utilidade nos estados terminais
    - Nos nós superiores Max obtém-se o maior valor (como no MimiMax)
    - Nos nós superiores Min obtém-se o menor valor (como no MimiMax)
    - Nos nós sorte, calcular o valor esperado:

$$\text{para Max} : E = \sum_i P(d_i) \max(utilidade(s))$$

$$\text{para Min} : E = \sum_i P(d_i) \min(utilidade(s))$$

- Exemplo - **Nó C:**
  - *E=1/36 * x(1,1) + 1/18 * x(1,1) + … **1/18** * **max(2, -1…1)** + … + 1/18 * x(6,6)*

Jogos com Elemento Sorte

33

- *expectminimax*

```
function expectiminimax(node, depth)
    if node is a terminal node or depth = 0
        return the heuristic value of node
    if the adversary is to play at node
        // Return value of minimum-valued child node
        let α := +∞
        foreach child of node
            α := min(α, expectiminimax(child, depth-1))
    else if we are to play at node
        // Return value of maximum-valued child node
        let α := -∞
        foreach child of node
            α := max(α, expectiminimax(child, depth-1))
    else if random event at node
        // Return weighted average of all child nodes' values
        let α := 0
        foreach child of node
            α := α + (Probability[child] * expectiminimax(child, depth-1))
    return α
```

$$\textsc{Expectiminimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \\ \sum_r P(r)\textsc{Expectiminimax}(\textsc{Result}(s,r)) & \text{if } \textsc{Player}(s) = \textsc{chance} \end{cases}$$

Jogos com Elemento Sorte

34

- …
  - Se usarmos **funções de avaliação e corte**, deve-se ter em conta que:
    - O algoritmo comporta-se de forma diferente se fizermos uma mudança na escala!
    - Para evitar este problema, a função de avaliação deve representar a probabilidade de ganhar de uma posição (ou, mais geralmente, da utilidade esperada da posição).