Inteligência Artificial

Busca Competitiva
Jogos

Prof. Paulo Martins Engel

Busca Competitiva

- Num ambiente multiagente, é necessário considerar as ações de outros agentes e o modo como essas ações nos afetam.
- A imprevisibilidade de outros agentes pode introduzir *contingências* no processo de resolução de problema.
- Em ambientes competitivos, as metas dos agentes estão em conflito, dando origem a problemas de *busca competitiva*, onde se enquadram os jogos.

Jogos em IA

- Em IA, os jogos são determinísticos, de revezamento de dois jogadores, com informações perfeitas.
- A posição (favorável ou desfavorável) de um jogador num determinado instante (*estado*) do jogo pode ser medida por uma *função de utilidade*.
- Os valores de utilidade dos agentes no fim do jogo são iguais e opostos (simétricos): +1 (ganha), ou –1 (perde).
- O objetivo da busca competitiva é planejar com antecedência num mundo em que outros agentes estão fazendo planos contra nós.

Jogos

- Entre os primeiros domínios de aplicação, pois:
  - É fácil representar o estado de um jogo.
  - Em geral, os agentes estão restritos a um pequeno número de ações com resultados definidos por regras precisas.
  - Constituem uma tarefa estruturada em que é fácil medir o sucesso ou fracasso.
  - Supunha-se que os jogos podiam ser solucionados por uma busca direta do estado inicial para a posição vencedora, sem grandes quantidades de conhecimento.
- Exceção aos jogos simulados: O futebol de robôs é um jogo físico, com descrições muito mais complicadas envolvendo ações bastante imprecisas.

Jogos

- Por volta de 1950, o xadrez foi estudado por Konrad Zuse, Claude Shannon, Norbert Wiener e Alan Turing.
- Atualmente, as máquinas ultrapassaram os seres humanos nos jogos de damas e Othello, derrotaram campeões do mundo (embora não todas as vezes) em xadrez (Deep Blue × Kasparov 1997) e gamão, e são competitivas em muitos outros jogos.
- Damas: Arthur Samuel (IBM) desenvolveu um programa que aprendia a sua própria função de avaliação, derrotando o campeão humano em 1962 (graças a um erro de Robert Nealy).
- O programa Chinahook (Jonathan Schaeffer) se tornou campeão mundial em 1994.
- A principal exceção é Go, em que os computadores se enquadram no nível amador.

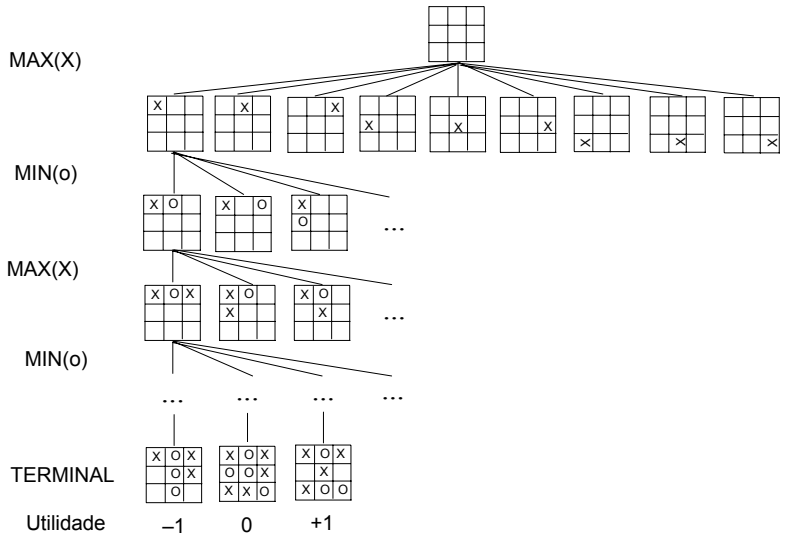
Exemplo: xadrez

- Fator médio de ramificação: 35
- Número médio de jogadas: 50 para cada jogador.
- Assim, a árvore completa de busca de um jogo terá aproximadamente 35^{100} ou 10^{154} nós.
- Portanto, uma busca cega é inviável, mesmo para realizar o primeiro movimento.
- Se deve fazer o melhor uso possível do tempo disponível para uma jogada: tomar alguma decisão, mesmo que a jogada ótima não seja determinada em tempo.

MINIMAX

- Algoritmo mais usado em jogos com dois jogadores, chamados MAX e MIN.
- MAX faz o primeiro movimento, e depois eles se revezam até o jogo terminar.
- Um jogo como problema de busca com os componentes:
  - **Estado inicial:** posição do tabuleiro, identifica jogador que fará o movimento.
  - **Função sucessor:** retorna lista de pares (*movimento*, *estado*).
  - **Função utilidade** (ou objetivo): dá valor numérico aos estados terminais. No xadrez: +1 (*vitória*), 0 (*empate*), –1 (*derrota*).

Uma árvore de busca do jogo da velha

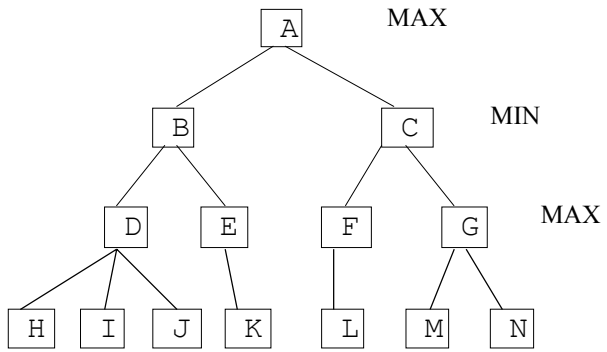


MINIMAX

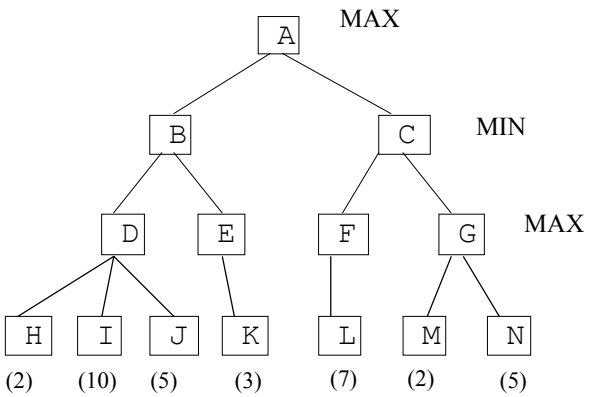
- MINIMAX é uma estratégia de contingência para MAX que especifica o movimento de MAX de modo a otimizar o valor de utilidade, quando está enfrentando um oponente com estratégia ótima.
- A estratégia ótima é determinada pelo exame do *valor minimax* de cada nó *n*: valor-minimax(*n*).
- O valor minimax de um nó é a utilidade (para MAX) de se encontrar naquele estado, supondo-se que ambos os jogadores têm desempenho ótimo.
- O valor minimax é obtido, recursivamente, pela propagação dos valores dos nós sucessores:

$$\text{valor}-\text{minimax}(n) = \begin{cases} \text{utilidade}(n), & \text{se } n \text{ é um estado terminal} \\ \max_{s \in \text{sucessores}} \text{valor}-\text{minimax}(s), & \text{se } n \text{ é um nó MAX} \\ \min_{s \in \text{sucessores}} \text{valor}-\text{minimax}(s), & \text{se } n \text{ é um nó MIN} \end{cases}$$

Uma árvore de jogo

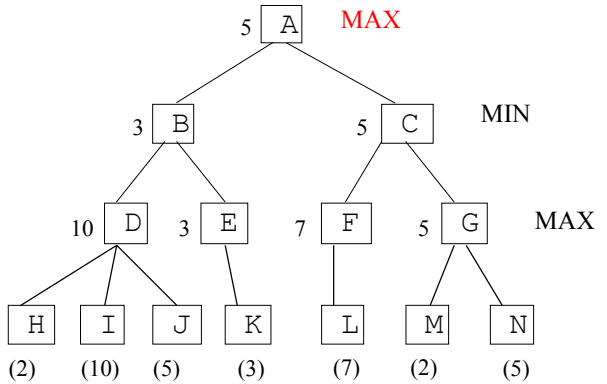


Aplicando uma função de avaliação nos nós folhas



A função de avaliação é em relação ao jogador de maximização

Propagando as avaliações



A jogada seria de “A” para “C”

Algoritmo Minimax

função DECISÃO-MINIMAX(*estado*) **retorna** *uma ação*
  entradas: *estado*, estado corrente no jogo
  v ← VALOR-MAX(*estado*)
retornar a *ação* em SUCESSORES(*estado*) com valor *v*

função VALOR-MAX(*estado*) **retorna** *um valor de utilidade*
  se TESTE-TERMINAL(*estado*) **então retornar** UTILIDADE(*estado*)
  v ← −∞
  para cada *s* em SUCESSORES(*estado*) **faça**
  v ← MAX(*v*, VALOR-MIN(*s*))
retornar *v*

função VALOR-MIN(*estado*) **retorna** *um valor de utilidade*
  se TESTE-TERMINAL(*estado*) **então retornar** UTILIDADE(*estado*)
  v ← ∞
  para cada *s* em SUCESSORES(*estado*) **faça**
  v ← MIN(*v*, VALOR-MAX(*s*))
retornar *v*

Algoritmo Minimax

- GERMOV (posição, jogador) - gera todas as jogadas válidas a partir da situação do jogo definida por *posição* e considerando que a jogada atual é de *jogador*.
- ESTÁTICA (posição, jogador) - retorna um valor que quantifica o estado atual das peças do jogo.
  - A função de avaliação é em relação ao jogador que faria a jogada
  - Quanto maior o valor melhor a situação
- PROFUNDO_SUFICIENTE (posição, profundidade) - pode considerar vários fatores:
  - número de níveis na árvore
  - um jogador ganhou
  - quão promissor é o caminho
  - quanto tempo ainda há disponível para a jogada

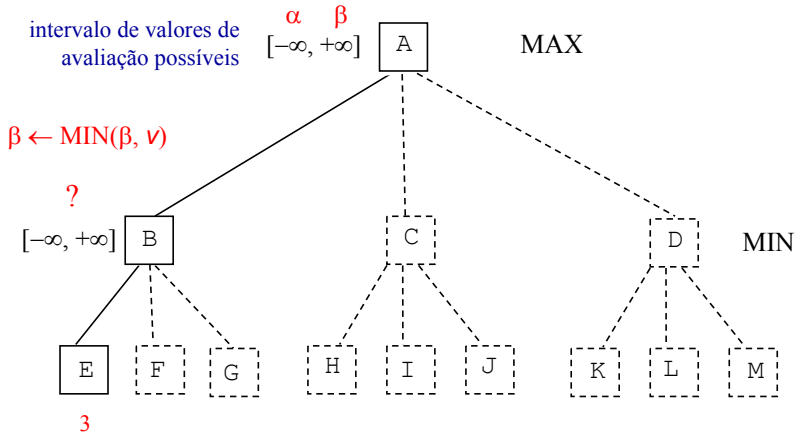

```
MINIMAX (POSIÇÃO, PROFUNDIDADE, JOGADOR)
1- se PROFUNDO_SUFICIENTE (POSIÇÃO, PROFUNDIDADE)
   então
        retorna estrutura:
          VALOR ← ESTÁTICA (POSIÇÃO, JOGADOR)
          CAMINHO ← nil
   senão
      SUCESSORES ← GERMOV (POSIÇÃO, JOGADOR)
   fim_se
2- se SUCESSORES = ∅
   então
        retorna estrutura:
          VALOR ← ESTÁTICA (POSIÇÃO, JOGADOR)
          CAMINHO ← nil
   senão
      MELHOR_CONTAGEM ← VALOR MÍNIMO DE ESTÁTICA (X,Y)
      para cada elemento SUC de SUCESSORES faça:
        RESULTADO_SUC←MINIMAX(SUC,PROFUNDIDADE+1,OPOSTO(JOGADOR))
        NOVO_VALOR ← - VALOR.RESULTADO_SUC
        se NOVO_VALOR > MELHOR_CONTAGEM
        então
          MELHOR_CONTAGEM ← NOVO_VALOR
          MELHOR_CAMINHO ← SUC + CAMINHO.RESULTADO_SUC
        fim_se
      fim_paracada
   fim_se
3- retorna estrutura:
     VALOR ← MELHOR_CONTAGEM
     CAMINHO ← MELHOR_CAMINHO
```


Poda Alfa-Beta

- Um aperfeiçoamento do algoritmo minimax corresponde a não pesquisar um ramo da árvore que comprovadamente não pode levar a um resultado melhor que o atual.
- α: o valor da melhor escolha (mais alto), até o momento, ao longo do caminho para MAX.
- β: o valor da melhor escolha (mais baixo), até o momento, ao longo do caminho para MIN.

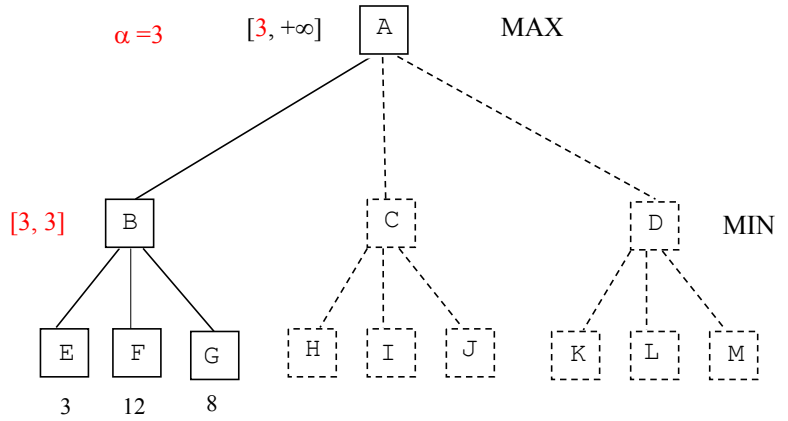
Exemplo de poda alfa-beta

- Busca em profundidade

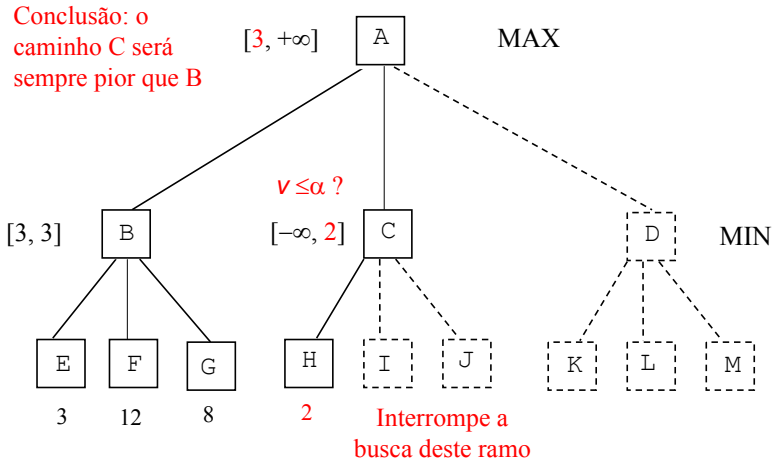


Exemplo de poda alfa-beta

α: o valor da melhor escolha (mais alto), até o momento, ao longo do caminho para MAX.

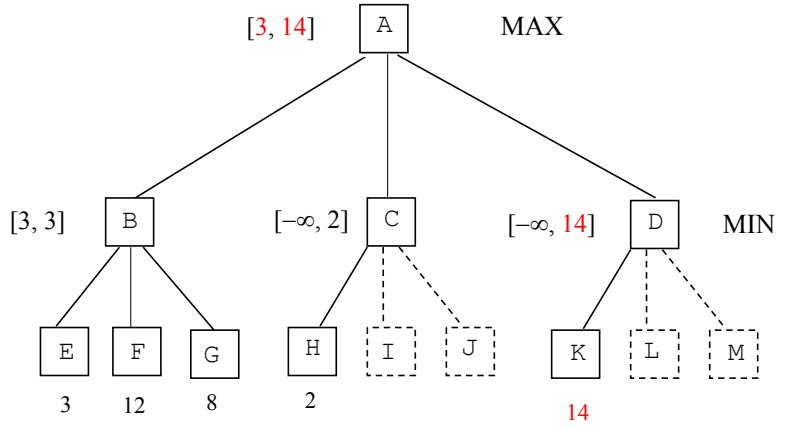


Exemplo de poda alfa-beta



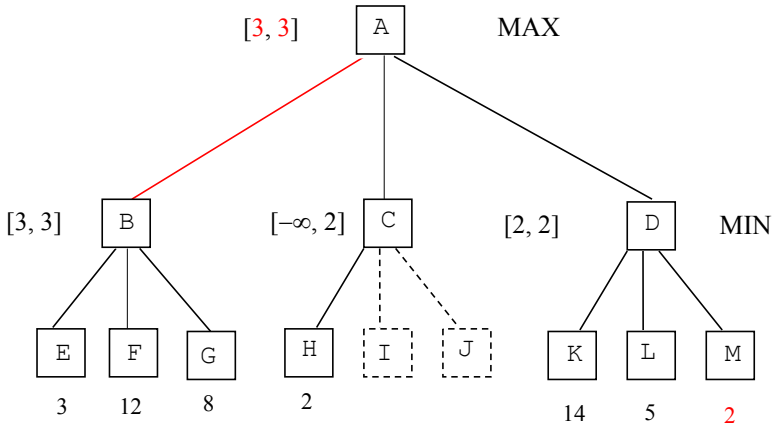
Exemplo de poda alfa-beta

β: o valor da melhor escolha (mais alto), até o momento, ao longo do caminho para MIN.



Exemplo de poda alfa-beta

- Escolhe o ramo com o maior valor.



Poda alfa-beta

função BUSCA-ALFA-BETA (*estado*) **retorna** *uma ação*
  entradas: *estado*, estado corrente no jogo
  v ← VALOR-MAX(*estado*, −∞, +∞)
retornar a *ação* em SUCESSORES(*estado*) com valor *v*

função VALOR-MAX (*estado*, α, β) **retorna** *um valor de utilidade*
  entradas: *estado*, estado corrente no jogo
    α, valor da melhor alternativa para MAX ao longo do caminho até *estado*
    β, valor da melhor alternativa para MIN ao longo do caminho até *estado*
  se TESTE-TERMINAL(*estado*) **então retornar** UTILIDADE(*estado*)
  v ← −∞
  para cada *s* em SUCESSORES(*estado*) **faça**
  v ← MAX(*v*, VALOR-MIN(*s*, α, β))
  se v ≥ β **então retornar** v
  α ← MAX(α, *v*)
retornar *v*

função VALOR-MIN (*estado*, α, β) **retorna** *um valor de utilidade*
  entradas: *estado*, estado corrente no jogo
    α, valor da melhor alternativa para MAX ao longo do caminho até *estado*
    β, valor da melhor alternativa para MIN ao longo do caminho até *estado*
  se TESTE-TERMINAL(*estado*) **então retornar** UTILIDADE(*estado*)
  v ← +∞
  para cada *s* em SUCESSORES(*estado*) **faça**
  v ← MIN(*v*, VALOR-MAX(*s*, α, β))
  se v ≤ α **então retornar** v
  β ← MIN(β, *v*)
retornar *v*

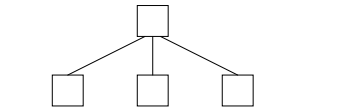
Refinamentos adicionais [RIC 94]

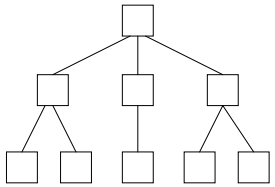
- Esperando por quietude: não parar a busca no meio de uma troca de peças, por exemplo
- Busca secundária: após decidir qual o melhor movimento, investigar este movimento 2 jogadas além do verificado inicialmente, para ter certeza que não há uma "armadilha"
- Usar movimentos de livros: por exemplo, aberturas e encerramentos

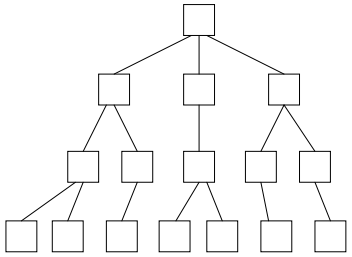
Aprofundamento progressivo

- Procedimento usado para evitar o problema de jogar com tempo definido
- base: encontrar a melhor solução com profundidade 1, depois com profundidade 2 e assim sucessivamente.
- Ao terminar o tempo disponível, apresenta-se a melhor solução encontrada até o momento

Aprofundamento progressivo







Problemas do Minimax

- Baseia-se fortemente em que o oponente escolherá sempre o melhor movimento
  - aceitável em situação de vitória
  - em situação de derrota pode ser melhor arriscar que o oponente cometerá um erro

Função de Avaliação

- Retorna uma estimativa da utilidade a partir da posição.
- Em geral, calcula diversas características do estado, $f_i(s)$.
- Agrupa estas características por uma função linear ponderada:

$$AVAL(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

- No xadrez $f_i(s)$ poderia representar o número de peças *i* (peão, cavalo, etc.) ou ainda: domínio do centro, mobilidade, etc
- w_i é o peso da característica *i* e pode ser ajustado por aprendizado
- Ex. xadrez: peão (1), cavalo (3), bispo (3), torre (5), rainha (9)
- Em muitos programas, a função de avaliação é não-linear, para refletir dependência de peças (dois bispos valem um pouco mais que um bispo).

Alguns jogos específicos

Os melhores programas de jogos jogam, em geral, muito bem, mas eles utilizam técnicas bastante diferentes das técnicas humanas:

- Nós determinamos as melhores posições a analisar por um processo de identificação de estruturas (*pattern matching*), que é um processo realizado em paralelo
- A exploração de um conjunto de jogadas possíveis, como realizado pelo computador, não é paralelizável