Inteligência Artificial - Notas de aula

Raoni F. S. Teixeira

Aula 4 - Busca Competitiva

1 Introdução

Essa aula apresenta o algoritmo de busca *Minmax* utilizado em jogos competitivos. Nesse tipo de ambiente, o agente não pode prever o comportamento do adversário, o que inviabiliza o uso dos algoritmos abordados anteriormente.

2 Jogo de dois jogadores

Jogos como xadrex ou damas ocorrem em ambientes determinísticos e observáveis. Os jogadores se alternam até o fim da partida. O vencedor recebe uma recompensa e o perdedor, uma penalidade. Em caso de empate, ambos recebem a mesma pontuação.

Formalmente, um jogo é descrito por seis componentes:

1. um estado inicial $s_i \in \mathcal{S}$ que especifica como o jogo começa;
2. uma função J, $J(s) \mapsto [\text{jogador}_1, \text{jogador}_2]$, que informa qual jogador deve agir em cada estado s;
3. uma função A, $A(s) \mapsto \mathcal{A}$, que devolve as ações válidas no estado s;
4. um modelo de transição T, $T(s,a) \mapsto s'$, que especifica o próximo estado (s') após aplicar a ação a no estado s;
5. uma função F, $F(s) \mapsto [\text{Verdadeiro}, \text{Falso}]$, que devolve o valor Verdadeiro se o jogo termina em s e Falso caso contrário e
6. Uma função utilidade $U(s) \mapsto [-\infty, \infty]$, que atribui valor numérico a estados terminais, do ponto de vista de um jogador. Se um jogador vence, $U(s) = 1$; se perde, $U(s) = -1$ e $U(s) = 0$ em caso de empate.

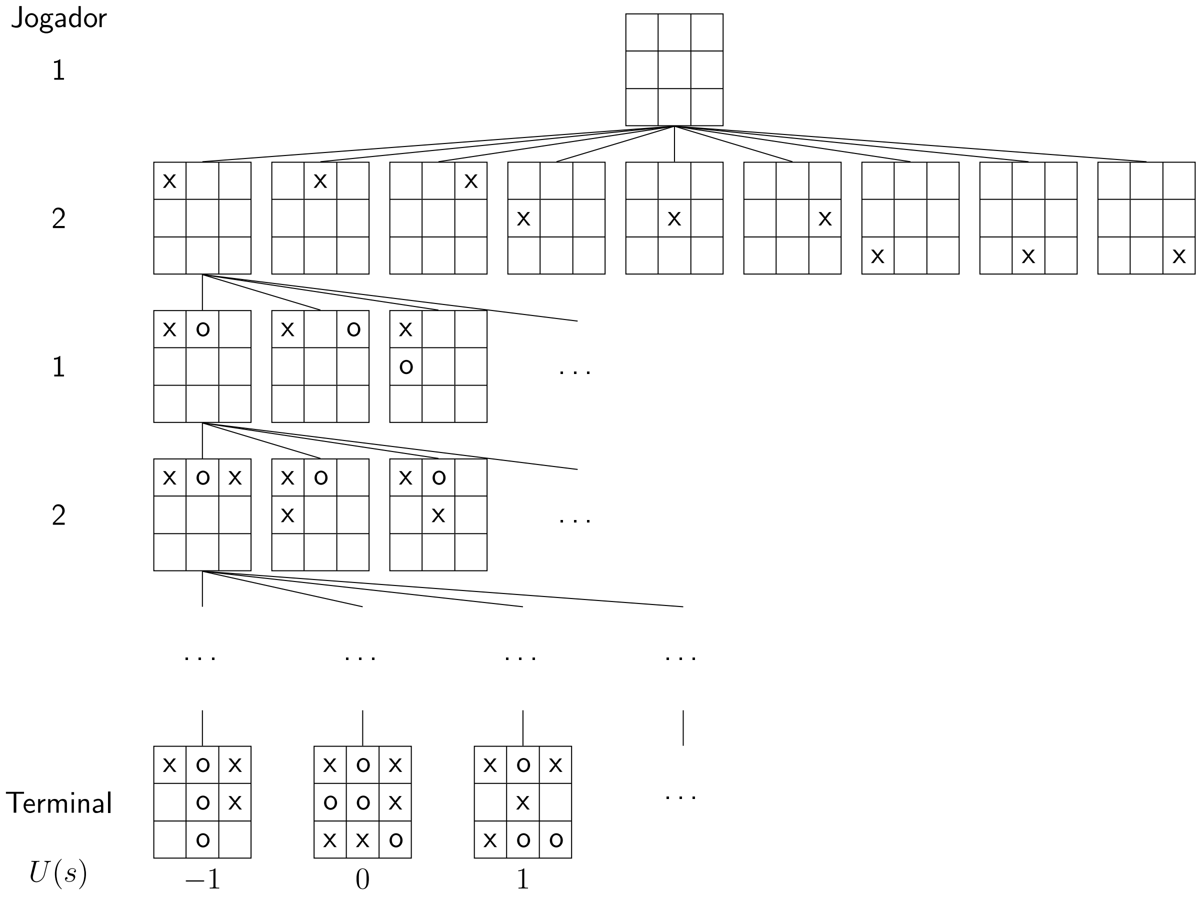


Figura 1: Árvore de partidas do jogo da velha. Os nós são os estados do jogo e cada caminho da raiz à folha é uma partida.

A Figura 1 mostra a árvore de partidas do jogo da velha, com o estado inicial na raiz e os estados terminais nas folhas. Cada caminho da raiz a uma folha é uma sequência completa de jogadas. A função U atribui valores aos nós terminais para indicar vitória (1), derrota (-1) ou empate (0).

3 Minmax

Nesse ambiente competitivo, o agente deve considerar que o adversário sempre escolherá a pior jogada possível para ele. Assim, o agente maximiza sua utilidade assumindo que o oponente a

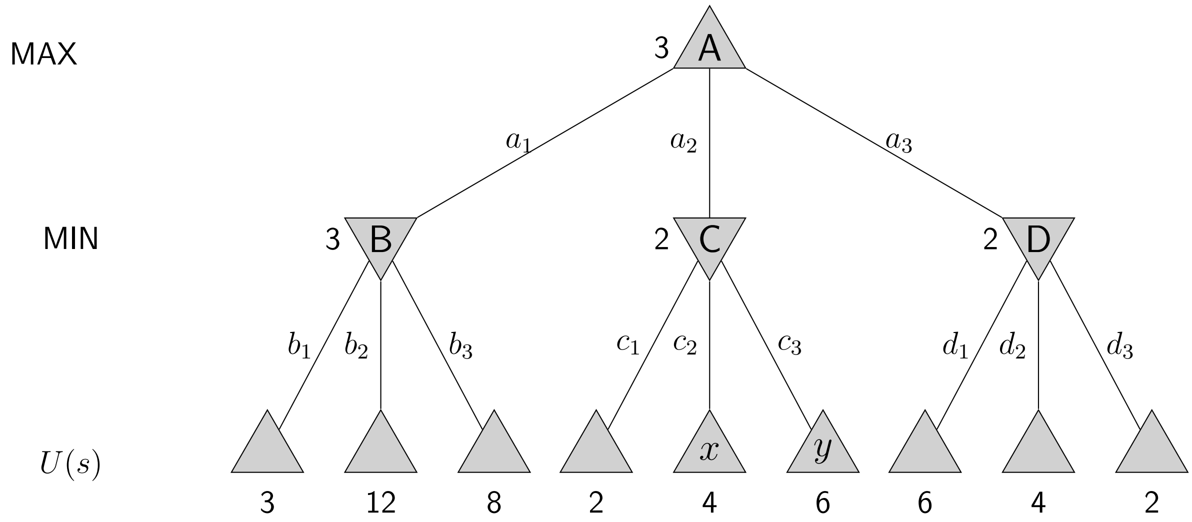


Figura 2: Árvore de um jogo de dois jogadores. $\triangle$ e $\bigtriangledown$ indicam o primeiro jogador (agente) e seu adversário. Os números nas folhas são os valores da função $U(s)$.

minimizará. Este é o problema de busca Minmax:

$$\text{Minmax}(s) = \begin{cases} U(s), & \text{se } F(s) \text{ é Verdadeiro.} \\ \max\limits_{a \in A(s)} \text{Minmax}(T(s,a)), & \text{se } J(s) \text{ é o agente.} \\ \min\limits_{a \in A(s)} \text{Minmax}(T(s,a)), & \text{se } J(s) \text{ é o adversário.} \end{cases} \tag{1}$$

A função Minmax avalia todas as combinações possíveis e devolve a jogada ótima (maior valor de U), assumindo que escolhas racionais de ambos os jogadores.

A Figura 2 ilustra a árvore de decisões de um jogo entre dois jogadores. Cada estado não terminal (A, B, C e D) possui três ações possíveis. Os valores nas folhas representam os resultados da função $U(s)$: o jogador $\triangle$ busca maximizar esses valores, enquanto seu oponente, $\bigtriangledown$, tenta minimizá-los.

Nesse cenário, $\triangle$ presume que o adversário escolherá as ações b_1, c_1 e d_3, que minimizam $U(s)$. Com base nessa suposição, $\triangle$ escolhe a ação a_1, pois ela leva ao maior valor possível de Minmax.

Como o número de estados cresce exponencialmente com a profundidade da árvore, é necessário aplicar estratégias para reduzir o tempo de execução do algoritmo. As duas mais comuns são:

- limitar a profundidade da árvore de busca e

- podar partes da árvore que não influenciam o resultado.

O controle de profundidade depende de uma função heurística que avalia estados não terminais. O algoritmo a seguir implementa a Equação 1 com esse controle. A função Eval(s) estima o quão favorável é um estado s para o agente. Dados um estado inicial s e um nível de profundidade p, o algoritmo retorna o valor Minmax correspondente à melhor jogada possível.

$\textsc{Minmax}(s, p)$

if $F(s) =$ *Verdadeiro* **ou** $p \leq 0$ **then**-
    return Eval(s)
if $J(s) =$ *agente* **then**-
    return $\textsc{MaxValor}(s, p)$
else-
    return $\textsc{MinValor}(s, p)$

$\textsc{MaxValor}(s, p)$

$v \leftarrow -\infty$
foreach $a \in A(s)$ **do**-
    $v \leftarrow \max(v, \textsc{Minmax}(T(s, a), p - 1))$
return v

$\textsc{MinValor}(s, p)$

$v \leftarrow +\infty$
foreach $a \in A(s)$ **do**-
    $v \leftarrow \min(v, \textsc{Minmax}(T(s, a), p - 1))$
return v

Embora o Minimax seja eficaz, a enumeração de todas as jogadas é cara computacionalmente. A poda alfa-beta resolve esse problema, como veremos a seguir.

4 Poda alpha-beta

A poda alfa-beta (α-β) otimiza o Minimax ao eliminar subárvores que não influenciam o resultado final e reduzir o número de nós avaliados.

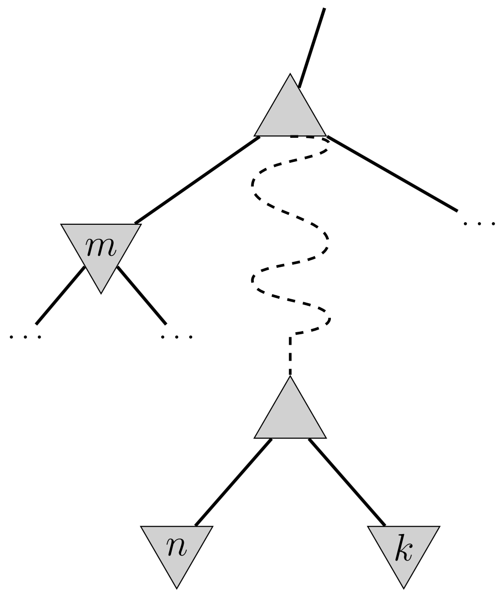


Figura 3: Poda alpha-beta. Se m ou n são melhores que k, então k pode ser podado.

Este é o caso das jogadas c_2 e c_3 na sub-árvore C da Figura 2 que não alteram o valor de Minmax:

$$\begin{aligned}
\mathrm{Minmax}(\mathsf{A}) &= \mathsf{max}(\mathsf{min}(\mathsf{B}), \mathsf{min}(\mathsf{C}), \mathsf{min}(\mathsf{D})) \\
&= \mathsf{max}(\mathsf{min}(3, 12, 8), \mathsf{min}(2, x, y), \mathsf{min}(6, 4, 2)) \\
&= \mathsf{max}(3, \mathsf{min}(2, x, y), 2) \\
&= 3 && (\forall \text{ valores de x, y})
\end{aligned}$$

A ideia da poda alpha-beta é otimizar a busca minmax evitando a exploração de subárvores que não influenciam a decisão final. Para isso, o algoritmo mantém dois valores: α, o melhor valor já encontrado para o jogador maximizador, e β, o melhor valor já encontrado para o minimizador. Esses limites permitem interromper precocemente a exploração de certos ramos da árvore de decisão quando se verifica que eles não podem melhorar a jogada já conhecida.

A Figura 3 ilustra esse princípio. Suponha que o jogador tenha a opção de seguir para o nó k. Se existir uma jogada anterior melhor no mesmo nível (por exemplo, n) ou em qualquer ponto acima na árvore (por exemplo, m), então k nunca será selecionado. Assim, ao identificarmos que k não pode superar as alternativas já conhecidas, podemos interromper a busca por seus descendentes e podar esse ramo da árvore.

A Figura 4 detalha passo a passo como os valores α e β evoluem durante a execução do algoritmo. No início da busca, $\alpha = -\infty$ e $\beta = \infty$. Cada subfigura (de **a** a **f**) representa um estágio da exploração da árvore e mostra como a poda reduz a quantidade de nós visitados.

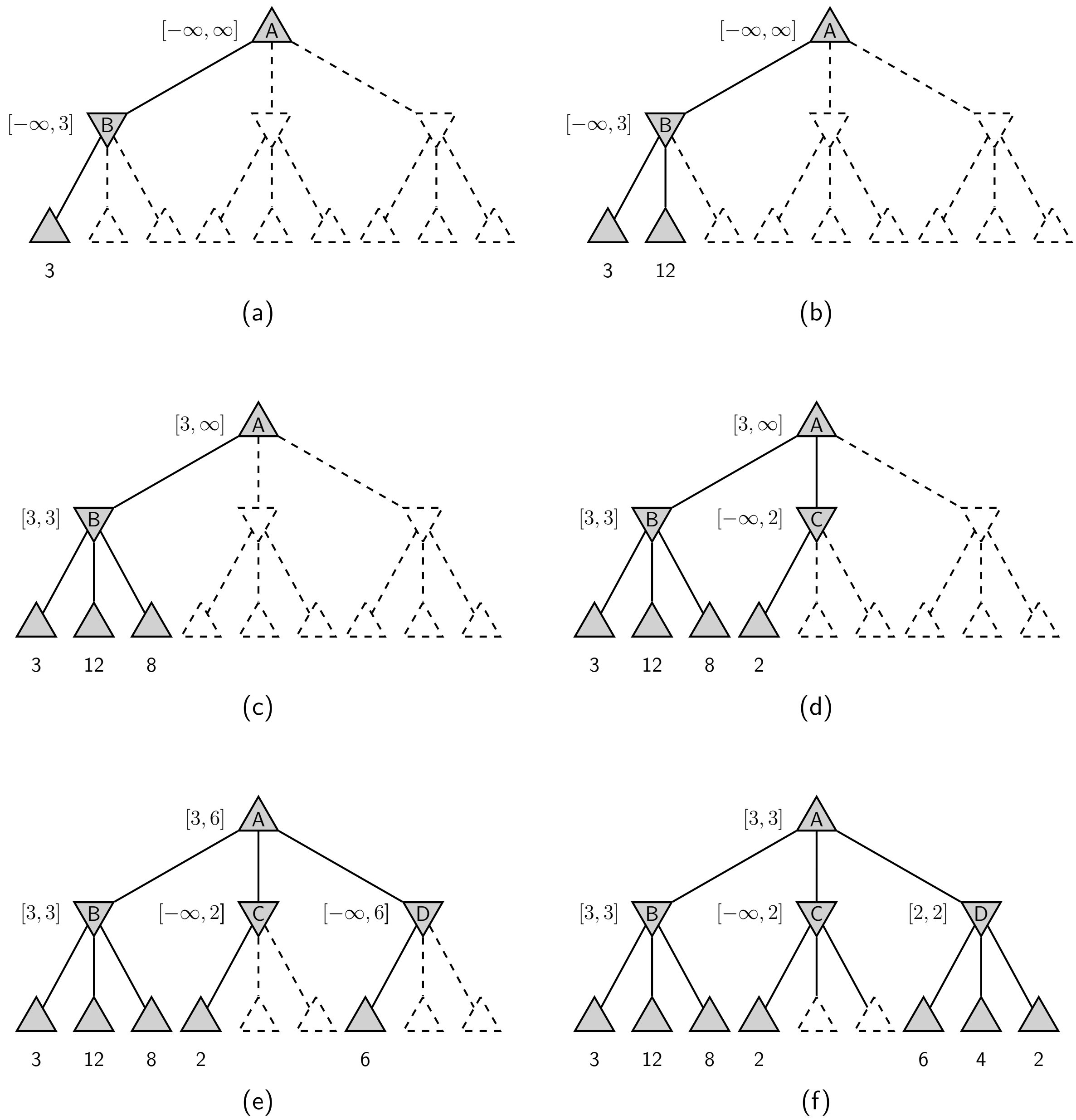


Figura 4: Simulação da poda alpha-beta para árvore da Figura 1. Os valores de alfa e beta estão indicados entre colchetes e são inicialmente $-\infty$ e ∞. A busca é interrompida na sub-árvore C — os estados tracejados na parte (f) foram podados.

Na Figura 4a, o valor 3 é descoberto na primeira folha da subárvore B, estabelecendo $\beta = 3$ naquele nó. Na Figura 4b, a segunda folha (valor 12) não altera β, pois o jogador $\bigtriangledown$ evita essa escolha. A Figura 4c mostra a terceira folha de B, com valor 8. Como todas as jogadas já foram exploradas, o valor final de B é 3.

Na Figura 4d, a primeira folha de C é 2, definindo $\beta = 2$. Como o valor de B (que é 3) já garante uma jogada melhor para $\bigtriangleup$, os demais nós de C são podados. Na Figura 4e, o valor 6 da primeira folha de D é maior que 3, o que exige continuar a busca. Como agora os três filhos da raiz foram parcialmente avaliados, já sabemos que o valor da raiz será no máximo 6.

Na Figura 4f, os valores 4 e 2 completam a subárvore D, cujo valor final é 2. A jogada ótima do jogador $\bigtriangleup$ em A é seguir para B (jogada a_1 na Figura 2).

As Figuras 4d, 4e e 4f também ilustram um aspecto importante: a ordem de descoberta das jogadas afeta diretamente a eficiência da poda. Na Figura 4d, a sequência de valores 2, 4 e 6 permite podar cedo. Já nas Figuras 4e e 4f, a ordem 6, 4, 2 não permite a mesma economia.

Os algoritmos a seguir implementam a poda alpha-beta com controle de profundidade. A função Eval representa uma heurística de avaliação de estados. Para utilizar o algoritmo, basta chamar:

$$\textsc{Minmax}(s_0, p, -\infty, \infty)$$

onde s_0 é o estado inicial e p a profundidade máxima de busca.

$\textsc{Minmax}(s, p, \alpha, \beta)$

```
if F(s) = Verdadeiro ou p ≤ 0 then-
    return Eval(s)
if J(s) = agente then-
    return MaxPoda(s, p, α, β)
else-
    return MinPoda(s, p, α, β)
```

$\textsc{MaxPoda}(s, p, \alpha, \beta)$

```
v ← −∞
foreach a ∈ A(s) do-
    v ← max(v, Minmax(T(s, a), p − 1, α, β))
    α ← max(α, v)
    if α ≥ β then-
        return α
return v
```

$\textsc{MinPoda}(s, p, \alpha, \beta)$

$v \leftarrow +\infty$
foreach $a \in A(s)$ **do**-
 $v \leftarrow \min(v, \textsc{Minmax}(T(s, a), p - 1, \alpha, \beta))$
 $\beta \leftarrow \min(\beta, v)$
 if $\alpha \geq \beta$ **then**-
  return β
return v

5 Discussão Final

O algoritmo Minmax tem origem na teoria dos jogos desenvolvida por John von Neumann e Oskar Morgenstern, que formalizaram a ideia de estratégias ótimas em jogos de soma zero no clássico *Theory of Games and Economic Behavior* [vNM44]. Em Inteligência Artificial, o Minmax foi popularizado com o trabalho pioneiro de Arthur Samuel [Sam59], em seu programa de damas, e posteriormente por Allen Newell e Herbert Simon, no sistema *Chess Program* [1].

O algoritmo permanece até hoje como base para sistemas de decisão em jogos de dois jogadores com turnos alternados, como xadrez, damas, jogo da velha, go e diversos jogos modernos. Combinado com heurísticas de avaliação e técnicas como poda alfa-beta, o Minmax é capaz de realizar buscas profundas de forma eficiente.

Uma das aplicações mais notáveis é o motor de xadrez *Stockfish*[2], considerado um dos mais fortes do mundo. Ele implementa variantes otimizadas do Minmax com extensões de busca, transposition tables e heurísticas sofisticadas para avaliar bilhões de posições. O algoritmo também está presente em IA de jogos clássicos como *Othello*, *Connect Four*, e diversos jogos de tabuleiro em ambientes acadêmicos e comerciais.

Outras aplicações incluem:

- planejamento adversarial em ambientes simulados;
- tomada de decisão em agentes multiagente;
- robótica competitiva (por exemplo, futebol de robôs);
- jogos eletrônicos com agentes NPC que antecipam ações do jogador [YT18].

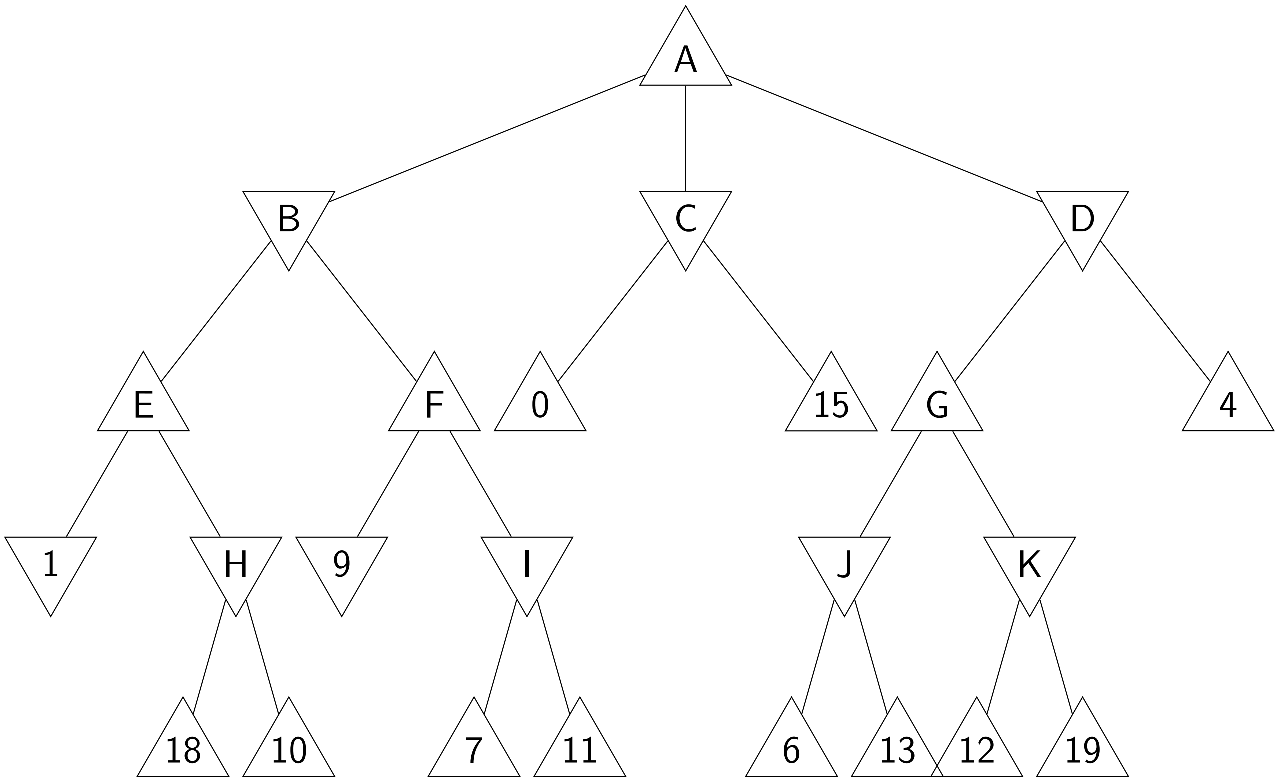
Para jogos com espaços de estados muito grandes, como go ou StarCraft, variantes mais avançadas como *Monte Carlo Tree Search* (MCTS) e redes neurais profundas substituem ou complementam o Minmax tradicional.

[1] https://www.chessprogramming.org/NSS
[2] https://github.com/official-stockfish/Stockfish

6 Exercícios

1. Como a busca minimax modela a competição em jogos de dois jogadores?

2. Calcule o valor de Minmax de cada nó árvore a seguir e indique os nós podados pela poda alfa-beta.



3. De que forma a poda alfa-beta melhora a eficiência da busca minimax?

Referências

[Sam59] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[vNM44] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[YT18] Georgios N. Yannakakis and Julian Togelius. *Artificial Intelligence and Games*. Springer, 2018.