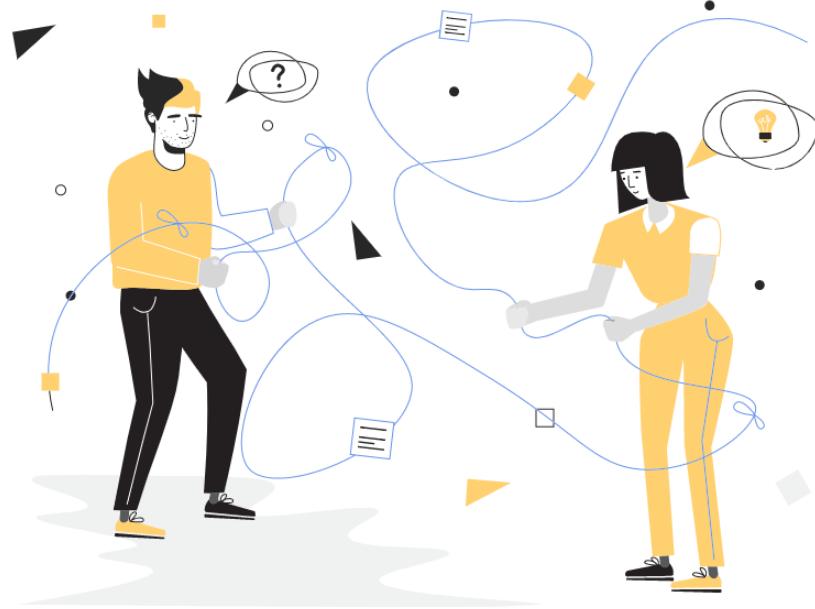




Universidade do Minho
Escola de Engenharia
Departamento de Informática

MÉTODOS DE RESOLUÇÃO DE PROBLEMAS E DE PROCURA Procura em contextos competitivos (Jogos)

LICENCIATURA EM ENGENHARIA INFORMÁTICA
MESTRADO integrado EM ENGENHARIA INFORMÁTICA
Inteligência Artificial
2022/23

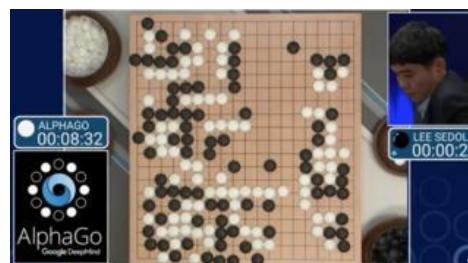


Fonte: <https://milanwittphol.com/projects/adversarial/index.html>

Jogos como Problemas de Procura



Fonte: <https://www.wired.com/2017/05/what-deep-blue-tells-us-about-ai-in-2017/>



Fonte: <https://www.bbc.com/news/technology-35785875>

- Jogos
- Jogos como Problemas de Procura
- Algoritmo Minimax
- Poda Alpha-beta (pruning)
- Decisões Imperfeitas em Tempo Real
- Jogos Determinísticos vs Estocásticos
- Jogos Estocásticos
 - Informação perfeita
 - Informação imperfeita (parcial)
- Conclusões



Jogos como Problemas de Procura

- Em tópicos anteriores, vimos que as estratégias de procura estão apenas associadas a uma entidade (agente) que pretende encontrar uma solução (por vezes, expressa numa sequência de ações);
- Porém, podem existir situações em que há mais que um agente na procura de soluções num espaço de procura (o que ocorre habitualmente em jogos);
- Em ambientes multiagentes como estes, cada agente necessita considerar as ações dos outros agentes e como estas o afetam.

Jogos como Problemas de Procura

- A imprevisibilidade destes agentes pode colocar **contingências** no processo de resolução de problemas por procura do agente.
- Estes ambientes competitivos, nos quais os objetivos dos agentes estão em conflito, dão lugar a problemas de procura com adversários (**adversarial search problems**).
- A Teoria de Jogos considera qualquer ambiente multiagente como um **jogo**, em que o impacto de um agente nos outros agentes é “significativo”, independentemente se os agentes são competitivos ou cooperativos.

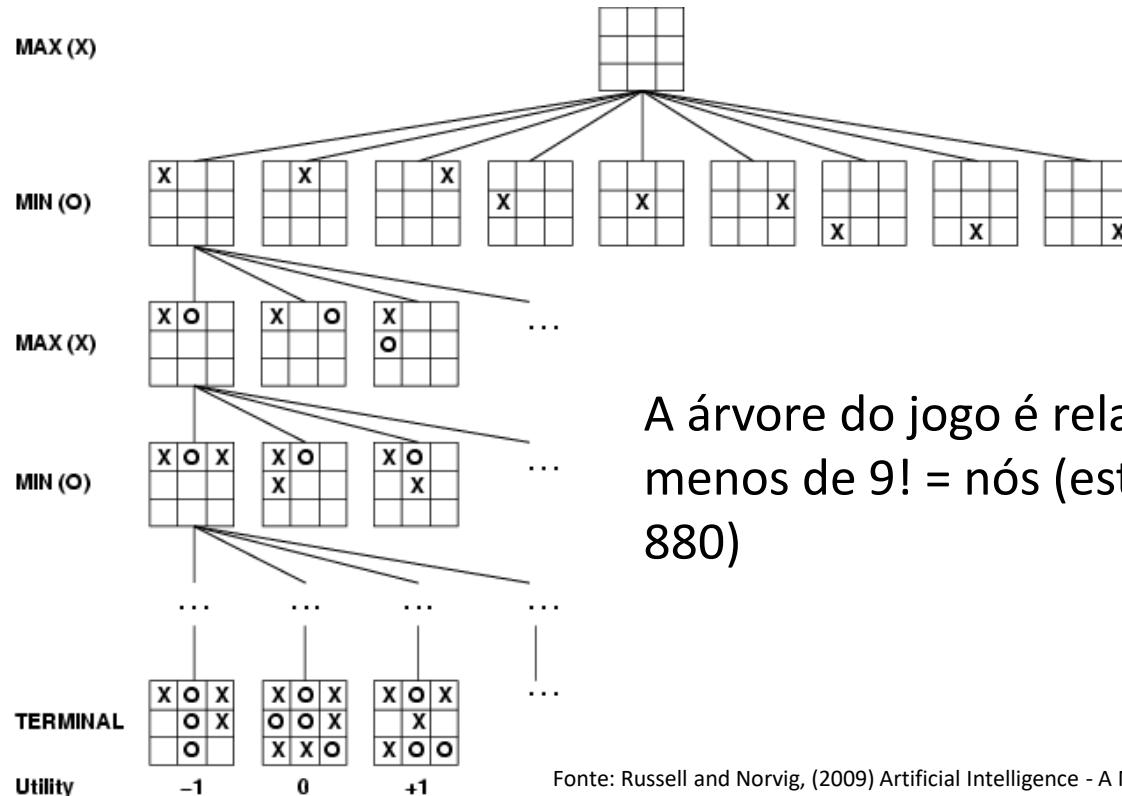
Porquê estudar jogos?

- Os jogos são tradicionalmente uma marca da inteligência;
- Os jogos são (relativamente) fáceis de formalizar;
- Os jogos podem ser um bom modelo de atividades competitivas ou cooperativas do mundo real
 - e.g, confrontos militares, negociações, leilões.

Jogos vs. Procura de agente único

- Não sabemos como o adversário irá agir
 - A solução não é uma sequência fixa de ações do estado inicial para o estado objetivo, mas uma estratégia ou política (um mapeamento do estado para a melhor jogada nesse estado);
- Eficiência é fundamental para jogar bem
 - O tempo para fazer um movimento é limitado;
 - O fator de ramificação, a profundidade da procura e o número de configurações do término são enormes.

Um jogo do galo entre dois jogadores, “max” e “min”



A árvore do jogo é relativamente pequena - menos de $9! =$ nós (estados) terminais (362 880)

Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

■ Tipos de Jogos:

- Informação:

- Perfeita: Xadrez, Damas, Go, Otelo, Gamão, Monopólio
- Imperfeita: Poker, Scrabble, Bridge, King

- Sorte/Determinístico:

- Determinístico: Xadrez, Damas, Go, Otelo
- Jogo de Sorte: Gamão, Monopólio, Poker, Scrabble, Bridge, King

■ Plano de “Ataque”:

- Algoritmo para o jogo perfeito
- Horizonte finito, avaliação aproximada
- Poda das árvores para reduzir custos (pruning)

■ Características:

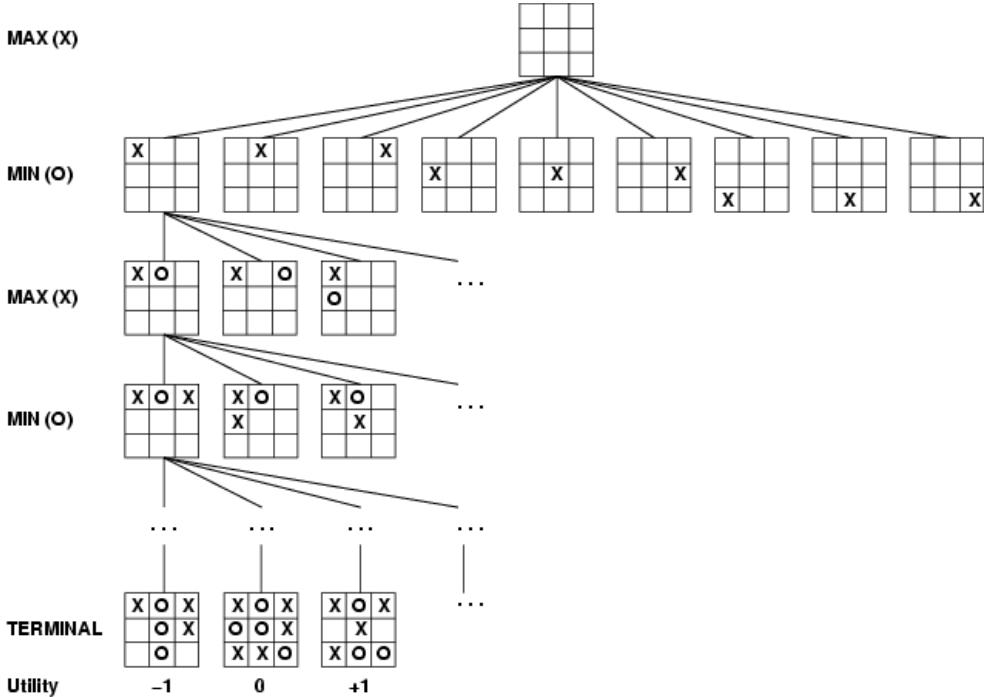
- Agente Hostil (adversário) incluído no mundo
- Oponente Imprevisível => Solução é um Plano de Contingência
- Tempo Limite => Pouco provável encontrar o objetivo. É necessário uma aproximação
- Uma das áreas mais antigas da IA. Em 1950 Shannon e Turing criaram os primeiros programas de Xadrez.
- Xadrez:
 - Todos consideram que é necessário ser “inteligente” para o jogar
 - Regras simples mas o jogo é complexo
 - Mundo totalmente acessível ao agente
 - Fator de ramificação médio de 35, partida com 50 jogadas => 35^{100} folhas na árvore de pesquisa (embora só existam 10^{40} posições legais)

- Considerando um jogo de informação perfeita com dois adversários podemos defini-lo formalmente como um problema de procura através dos seguintes elementos:
 - **Estado Inicial** (posição no tabuleiro e qual o próximo jogador a jogar)
 - **Conjunto de Operadores** (que definem os movimentos legais)
 - **Teste Terminal** (que determina se o jogo acabou - estado terminal)
 - **Função de Utilidade** (que dá um valor numérico para o resultado do jogo, por exemplo 1 (vitória), 0 (empate), -1 (derrota))
- O algoritmo MiniMax pode ser aplicado como estratégia de resolução neste tipo de jogos (com dois adversários e informação perfeita)

Jogos como Problemas de Procura

- Minimax: Jogo do Galo

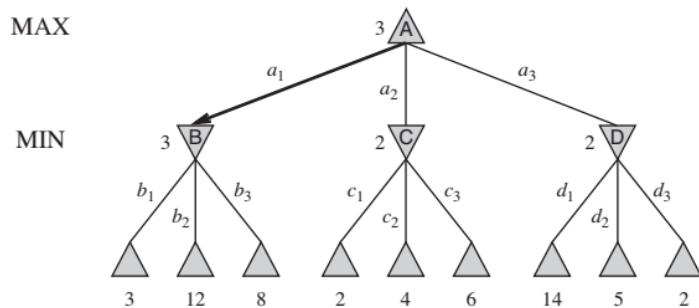
- O nodo do topo é o estado inicial
- MAX joga primeiro, colocando um X no quadrado vazio
- Na imagem é destacada uma parte da árvore de procura, em que se mostra as possíveis jogadas alternativas para MIN(o) e MAX(x) até atingirem os estados terminais (em que pode ser atribuído valores de utilidade mediante as regras do jogo)



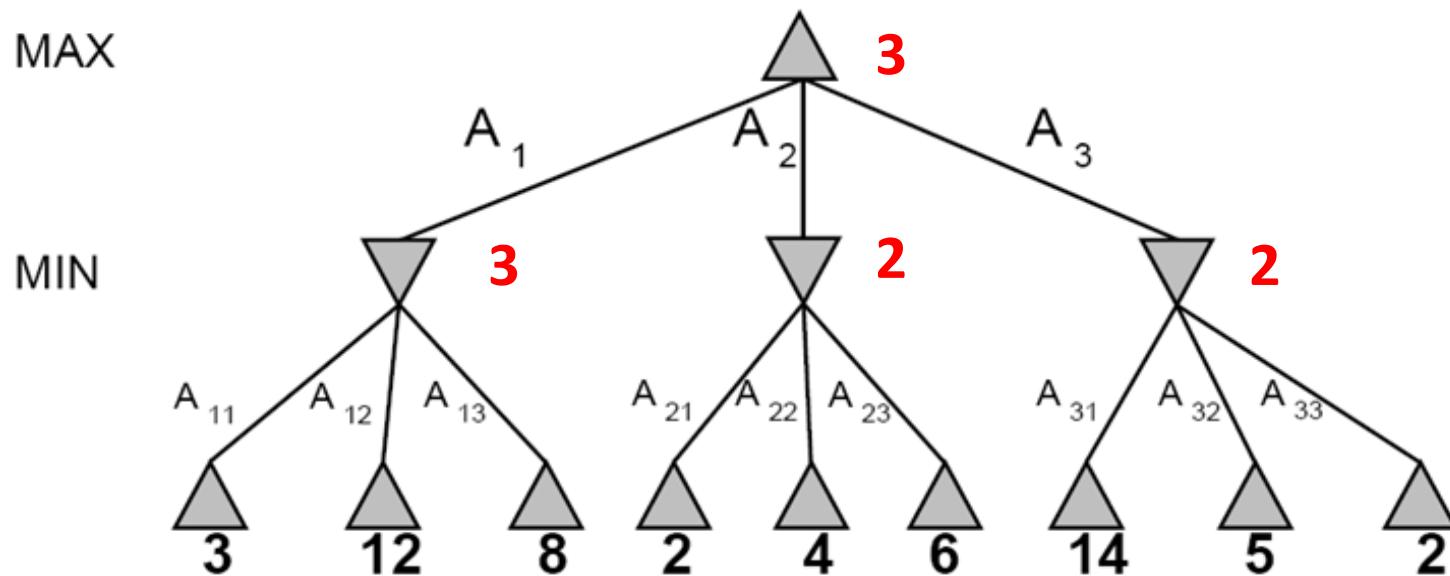
Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Num problema de procura normal, a solução ideal seria uma sequência de movimentos que levam a um estado objetivo (um estado terminal que é uma vitória).
- Em um jogo, o MIN tem algo a dizer sobre isso e, portanto, o MAX deve encontrar uma estratégia contingente, que especifique:
 - o movimento do NAX no estado inicial,
 - de seguida, os movimentos do MAX nos estados resultantes de todas as respostas possíveis do MIN,
 - os movimentos de MAX nos estados resultantes de toda resposta possível de MIN a esses movimentos
- Uma estratégia ideal leva a resultados pelo menos tão bons quanto qualquer outra estratégia quando se está a jogar com um oponente infalível.

- O algoritmo MiniMax pode ser aplicado como estratégia de resolução neste tipo de jogos (determinísticos, com dois adversários e informação perfeita) e consiste em:
 - Gerar a árvore completa até aos estados terminais
 - Aplicar a função utilidade a esses estados
 - Calcular os valores da utilidade até a raiz da árvore, uma camada de cada vez
 - Escolher o movimento com o valor mais elevado.

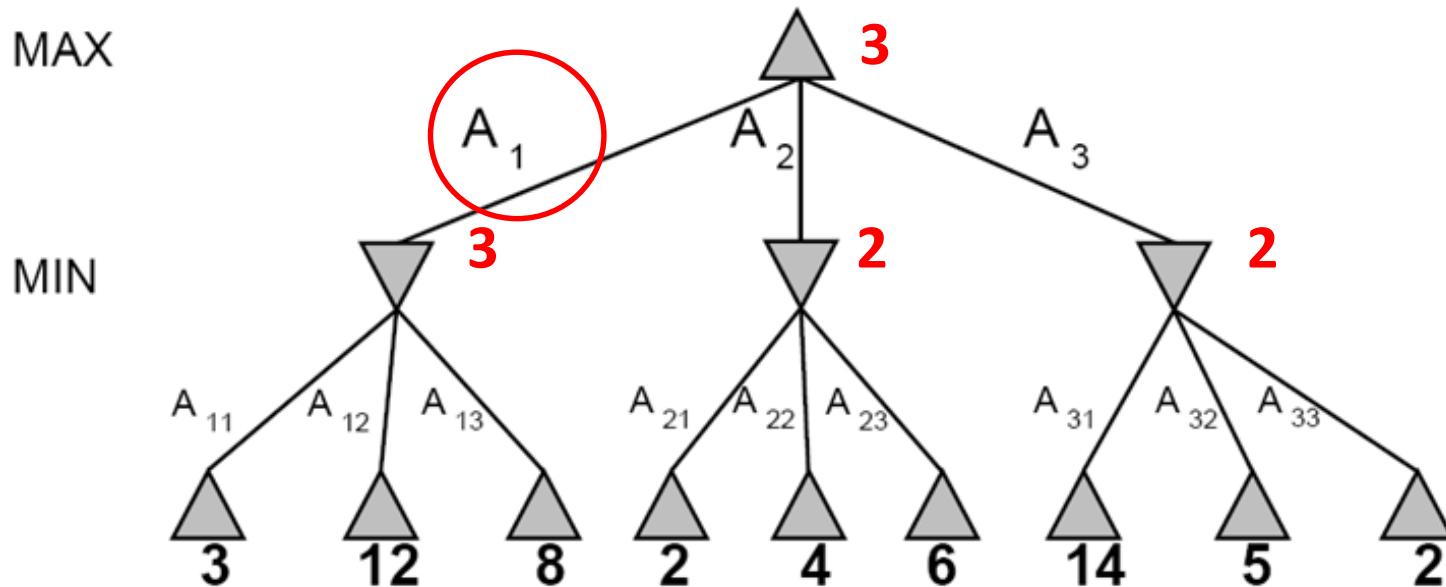


Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.



- **Valor minimax de um nó:** a utilidade (para MAX) de estar no estado correspondente, assumindo o desempenho perfeito dos dois lados
- **Estratégia Minimax:** escolher a jogada que oferece o melhor retorno do pior caso

Calculando o valor minimax de um nó



- Minimax (nó) =
 - utilidade (nó) se o nó for terminal
 - $\text{Max}_{\text{action}} \text{ Minimax}(\text{Sucessor(nó, ação)})$ se jogador = MAX
 - $\text{min}_{\text{action}} \text{ Minimax}(\text{Sucessor(nó, ação)})$ se jogador = MIN

- Dada uma árvore de jogo, a estratégia ideal pode ser determinada analisando o valor minimax de cada nó (**MINIMAX-VALUE (n)**)
- O valor minimax de um nó é a utilidade de estar no estado correspondente, supondo que os dois jogadores jogam da melhor forma dali até ao final do jogo.
- O MAX prefere passar para um estado de valor máximo, enquanto que o MIN prefere um estado de valor mínimo.

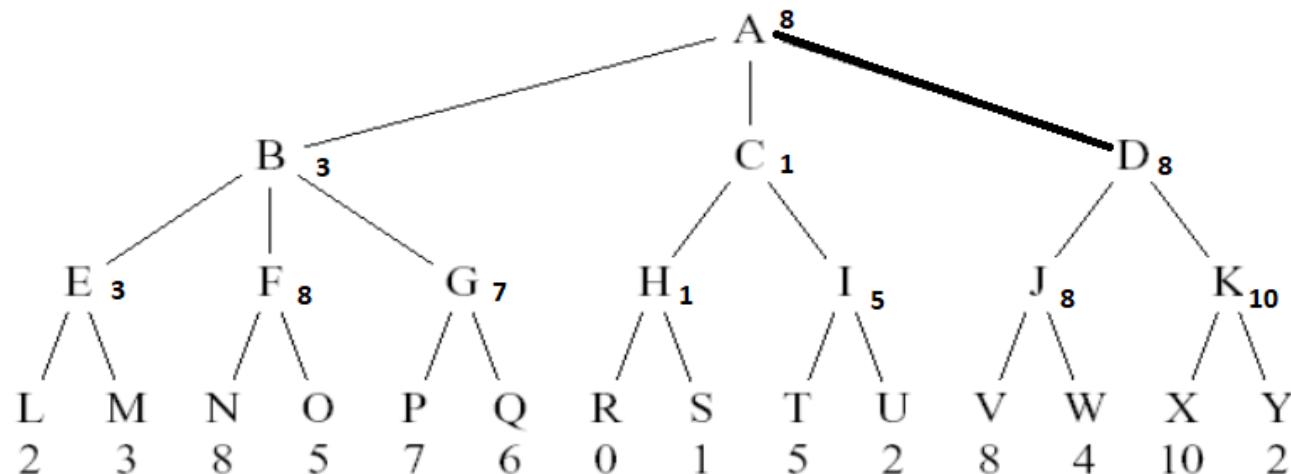
```
function MINIMAX-DECISION(state) returns an action
    v  $\leftarrow$  MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v
```

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v  $\leftarrow -\infty$ 
    for a, s in SUCCESSORS(state) do
        v  $\leftarrow$  MAX(v, MIN-VALUE(s))
    return v
```

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v  $\leftarrow \infty$ 
    for a, s in SUCCESSORS(state) do
        v  $\leftarrow$  MIN(v, MAX-VALUE(s))
    return v
```

Solução:

- Supondo que MAX é o primeiro a jogar, aplique o Algoritmo Minimax à seguinte árvore, indicando o movimento selecionado pelo algoritmo e o respetivo valor estimado.



■ Propriedades:

- Completo? Sim, se a árvore for finita!
- Ótimo? Sim, contra um adversário ótimo! Senão?
- Complexidade no Tempo? $O(b^m)$
- Complexidade no Espaço? $O(bm)$ (exploração primeiro em profundidade)

■ Problema:

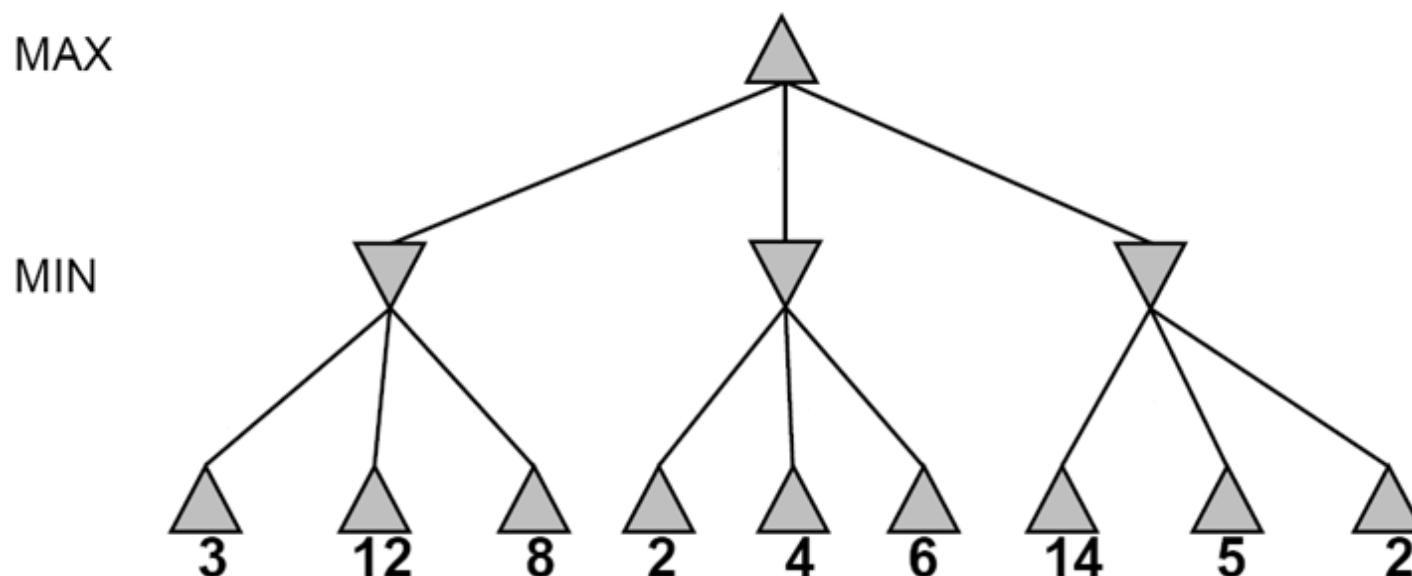
- Inviável para qualquer jogo minimamente complexo

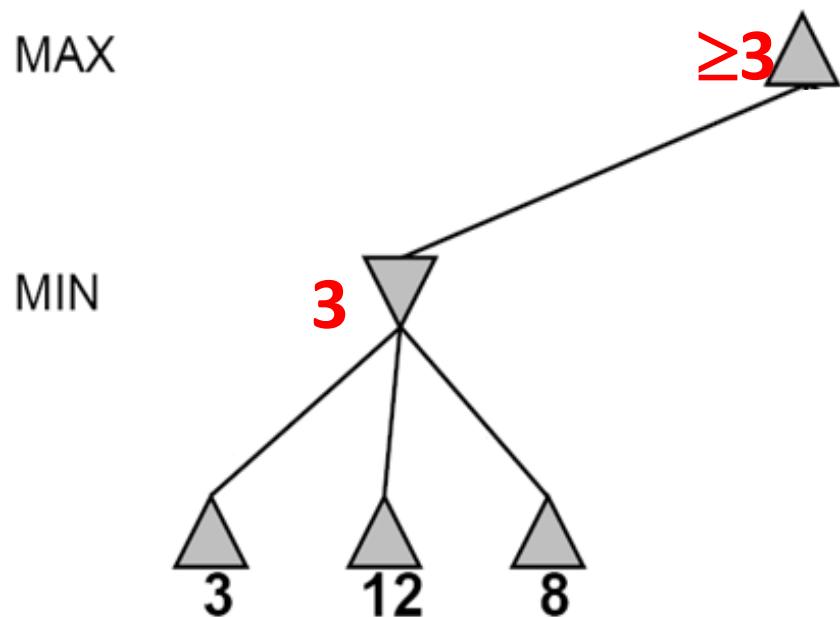
- b: o máximo fator de ramificação (o número máximo de sucessores de um nó) da árvore de procura
- d: a profundidade da melhor solução
- m: a profundidade máxima do espaço de estados

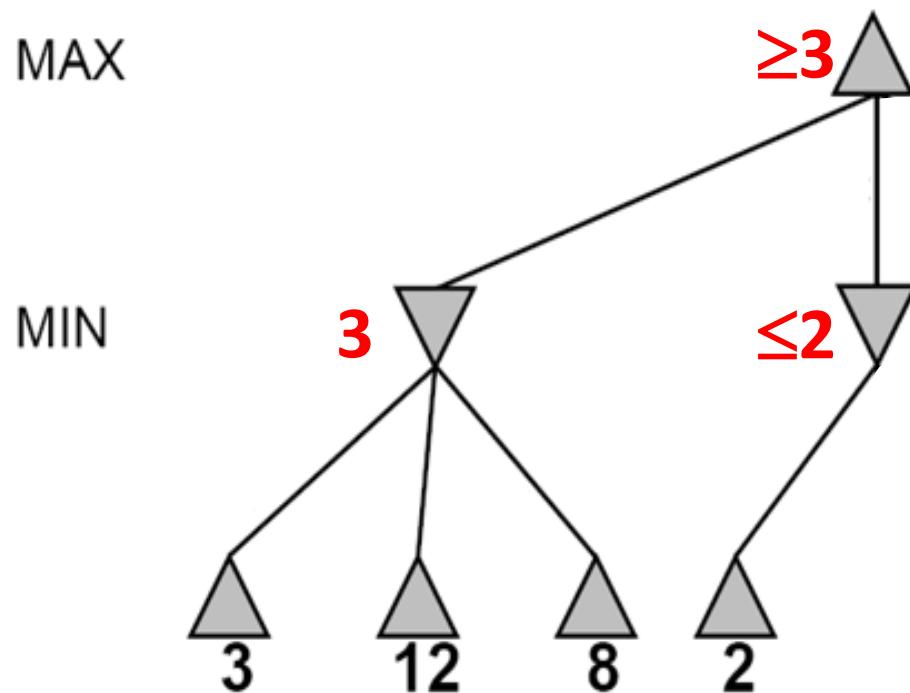
■ Exemplo:

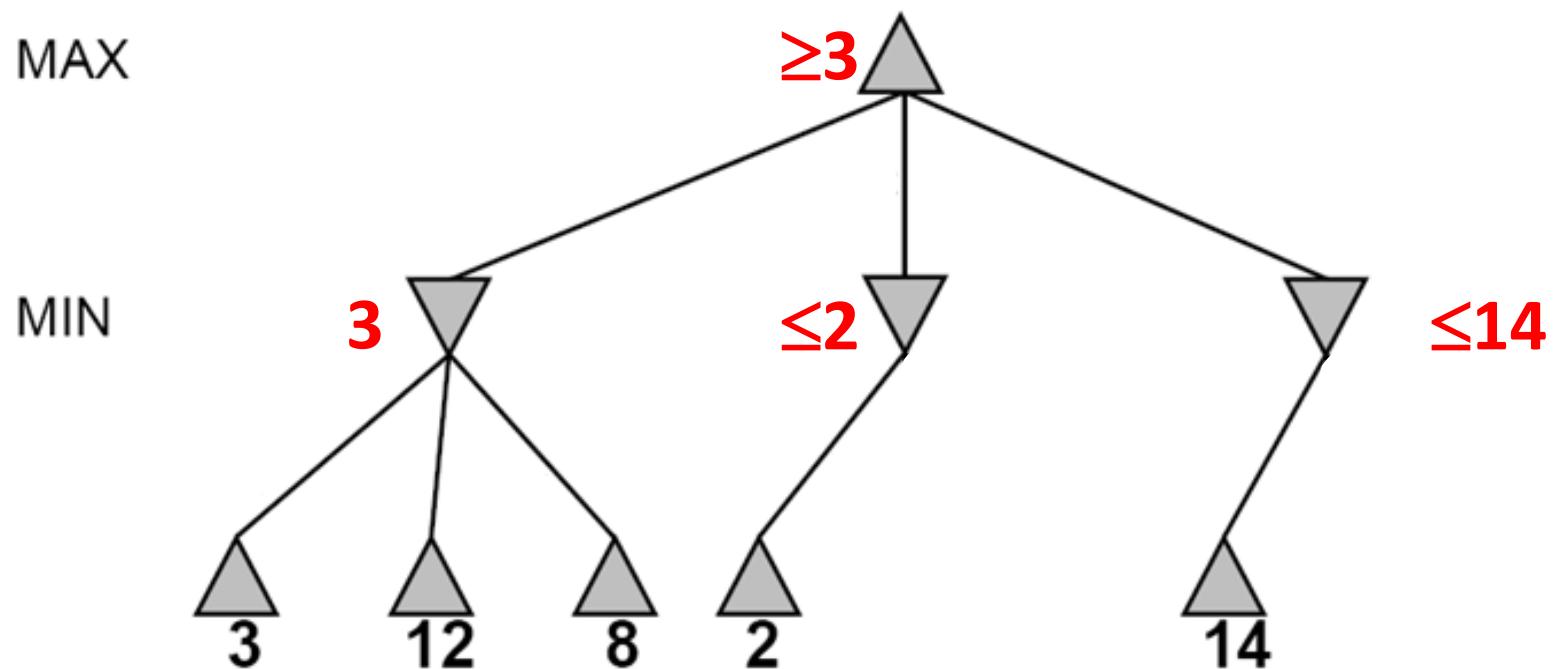
- Para o xadrez ($b=35$, $m=100$), $b^m = 35^{100} = 2.5 \cdot 10^{154}$
- Supondo que são analisadas 450 milhões de hipóteses por segundo $\Rightarrow 2 \cdot 10^{138}$ anos para chegar à solução!

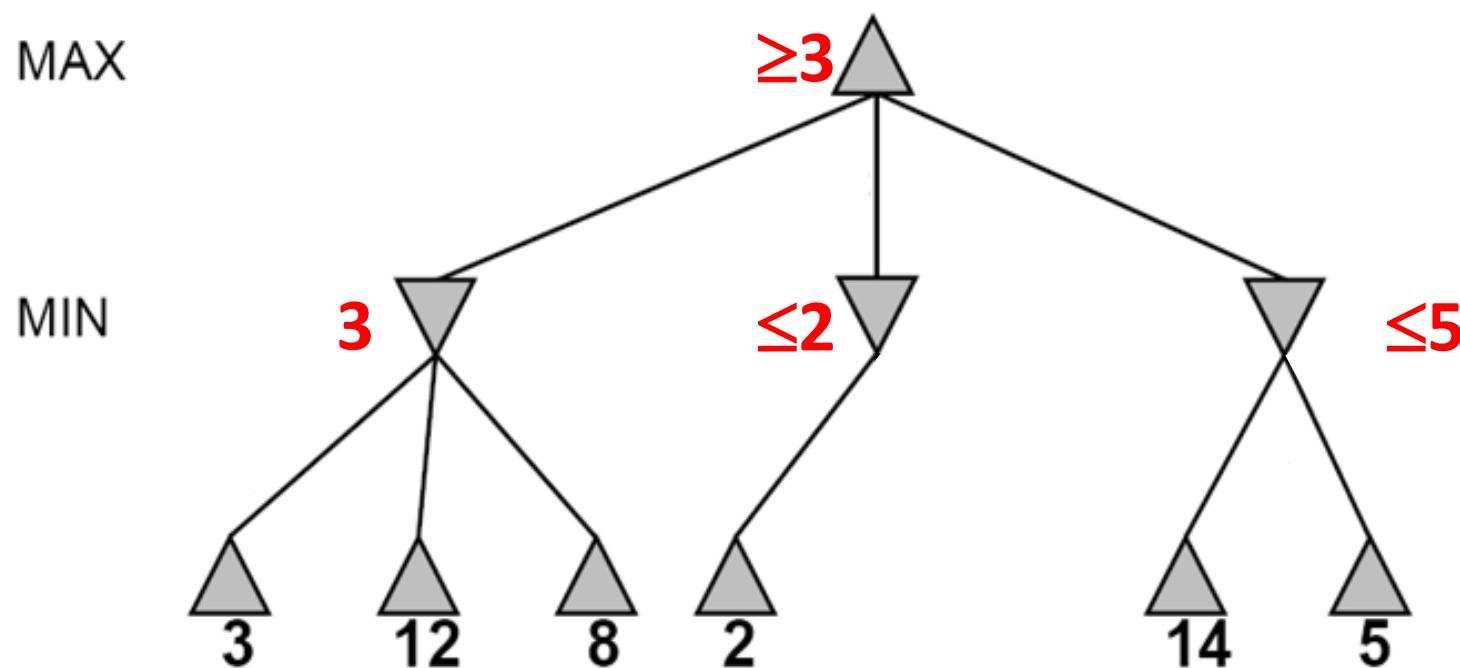
- É possível calcular a decisão minimax exata sem expandir todos os nós na árvore do jogo...

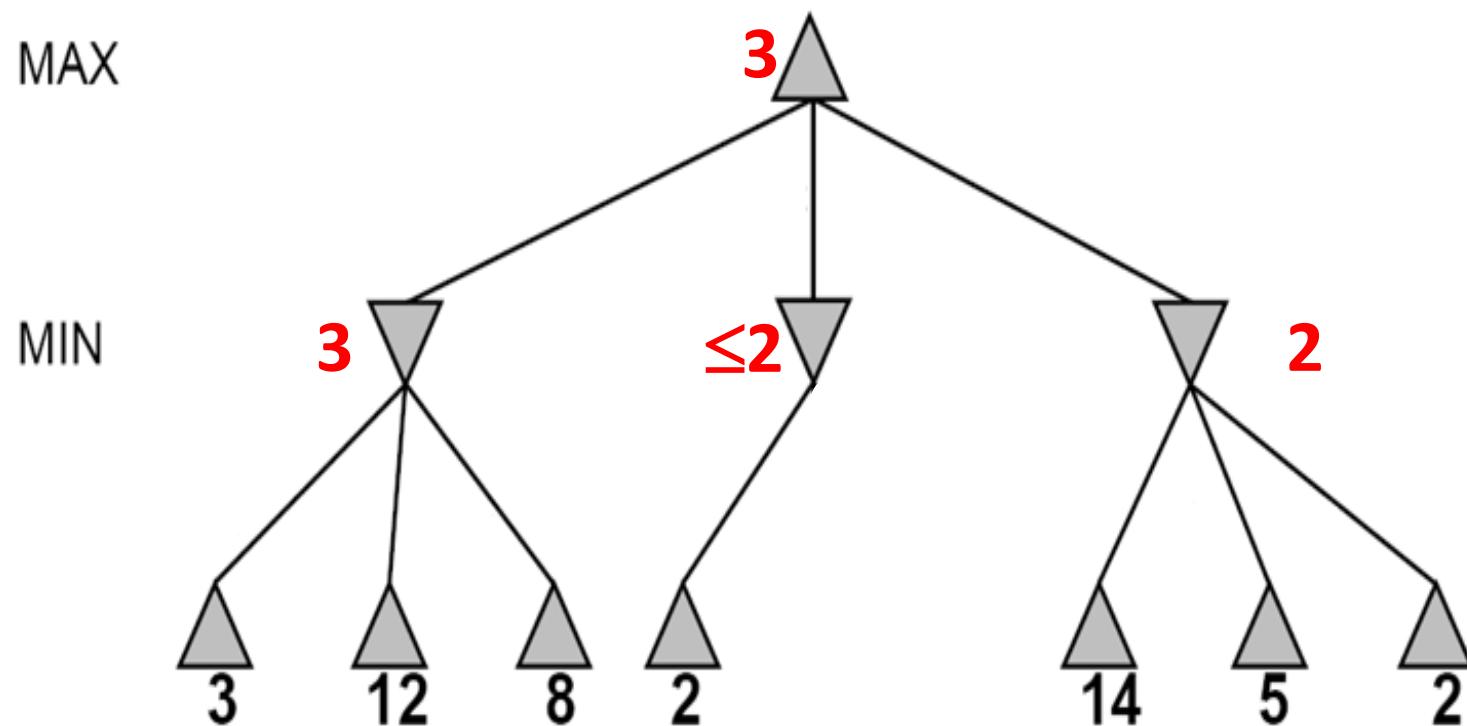








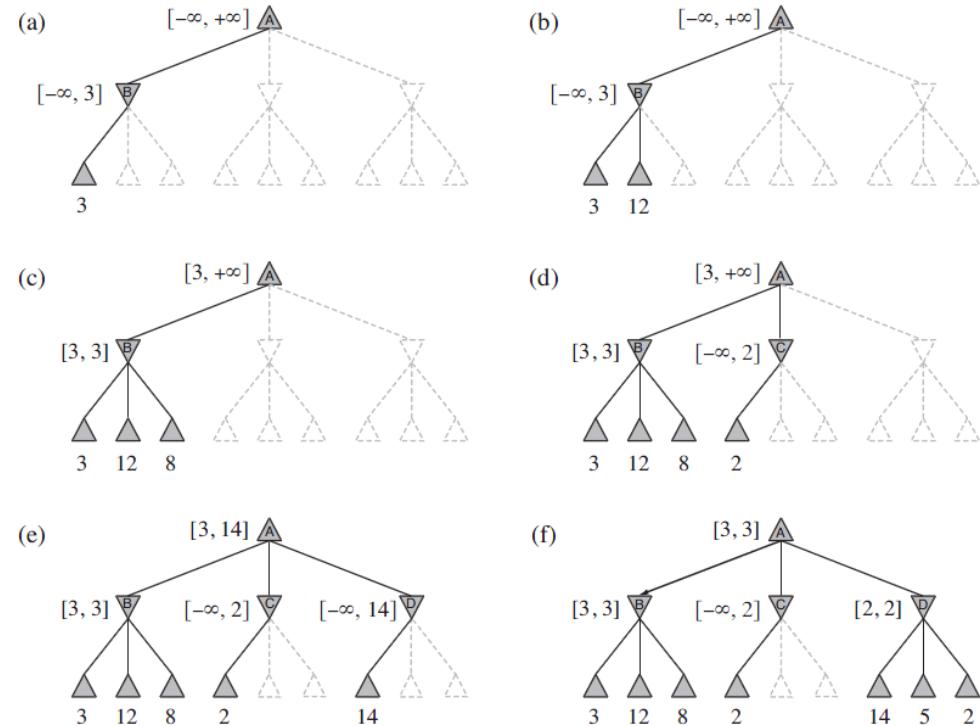




- α é o melhor valor (para Max) encontrado até agora no caminho correto
- Se V for pior do que α , Max deve evitá-lo => cortar o ramo
- β é definido da mesma forma para Min

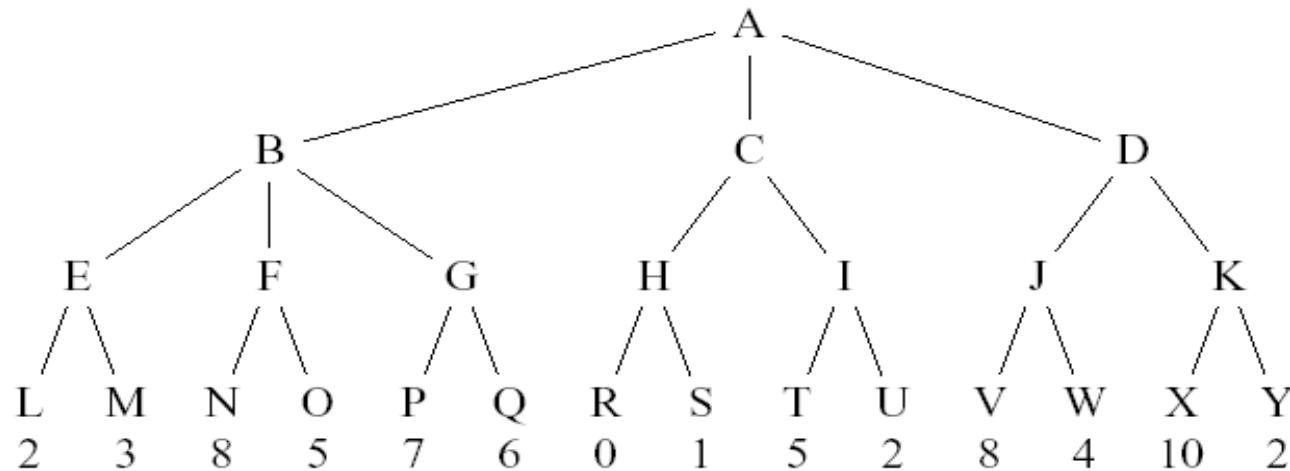
O Alfa-Beta não afeta o resultado final
Uma boa ordenação melhora a eficiência da poda

Alpha-beta pruning Poda Alfa-Beta



Exercício – MINIMAX com poda

- Supondo que MAX é o primeiro a jogar, aplique o Algoritmo Minimax com poda Alfa-Beta à seguinte árvore, indicando o movimento selecionado pelo algoritmo. Indique graficamente e justifique a poda que efetuou na aplicação do algoritmo Minimax

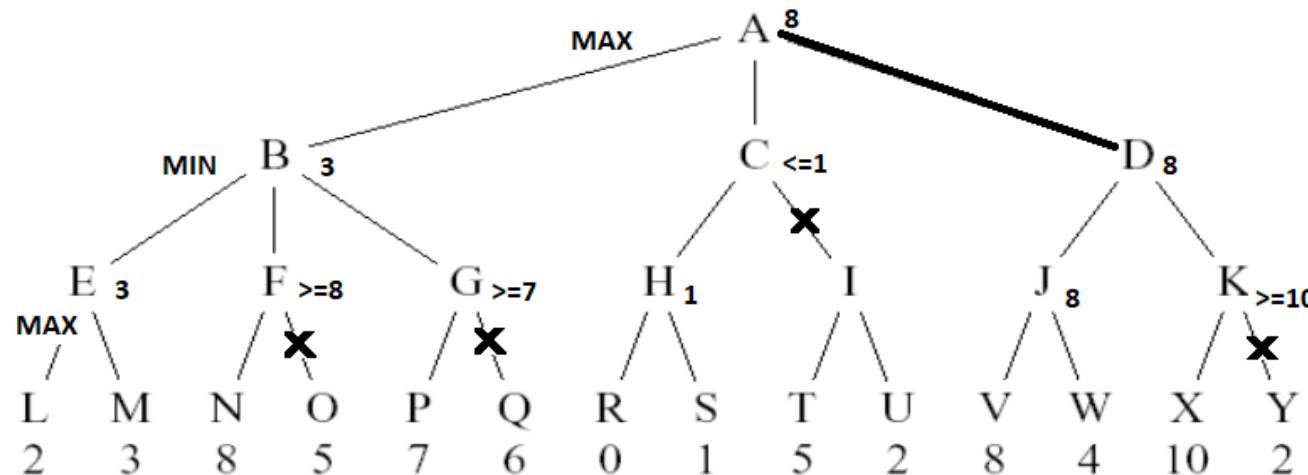


Fonte: Luís Paulo Reis, Artificial Intelligence (2019), Universidade do Porto

Exercício – MINIMAX com poda

Solução:

- Supondo que MAX é o primeiro a jogar, aplique o Algoritmo Minimax com poda Alfa-Beta à seguinte árvore, indicando o movimento selecionado pelo algoritmo. Indique graficamente e justifique a poda que efetuou na aplicação do algoritmo Minimax



Alpha-beta pruning Poda Alfa-Beta

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v  $\leftarrow$   $-\infty$ 
  for each a in ACTIONS(state) do
    v  $\leftarrow$  MAX(v, MIN-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\geq \beta$  then return v      <----- ponto de poda
     $\alpha \leftarrow$  MAX( $\alpha$ , v)
  return v
```

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v  $\leftarrow$   $+\infty$ 
  for each a in ACTIONS(state) do
    v  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v      <----- ponto de poda
     $\beta \leftarrow$  MIN( $\beta$ , v)
  return v
```

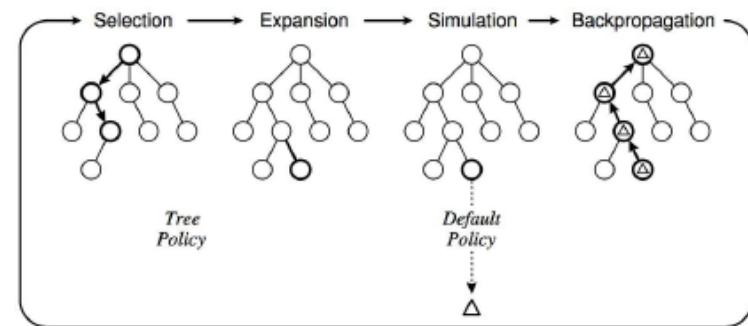
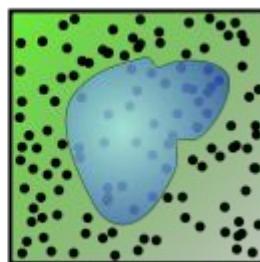
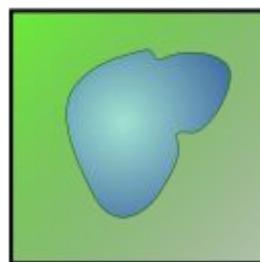
Problema Decisões Imperfeitas em Tempo Real

- O algoritmo minimax gera todo o espaço de procura do jogo, enquanto o algoritmo alfa – beta nos permite remover grande parte dele. No entanto, o alfa – beta ainda necessita executar uma procura até os estados terminais em pelo menos uma parte do espaço de procura;
- Essa profundidade geralmente não é prática, porque as mudanças devem ser feitas em um período de tempo razoável - normalmente alguns minutos no máximo;
 - Função de Avaliação: Utilidade (interesse) estimada para a posição
 - Teste de Corte: Profundidade Limite
- Surge eventualmente aqui outro problema: Problema do horizonte!
 - Tabela de transposição para armazenar estados expandidos anteriormente;
 - Poda (*pruning*) direta para evitar considerar todos os movimentos possíveis;
 - Tabelas de pesquisa para movimentos de abertura e jogos finais

- Na base: 200 milhões de avaliações de nós por movimento (3 min), minimax com uma função de avaliação decente e procura por quiescência
 - 5 jogadas ≈ humano novato
- Adicione poda alfa-beta
 - 10 jogadas, jogador experiente
- Deep Blue: 30 bilhões de avaliações por jogada, função de avaliação com 8000 recursos, grandes bases de dados de movimentos de abertura e final de jogo
 - 14 jogadas ≈ Garry Kasparov
- Estado da arte atual (Hydra, et al., 2006): 36 bilhões de avaliações por segundo, técnicas avançadas de poda
 - 18 jogadas - melhor do que qualquer ser humano vivo?

Monte Carlo Tree Search

- Aplica-se em os jogos com árvores profundas, grande fator de ramificação e sem boas heurísticas - como o Go?
- Em vez de procura com profundidade limitada com uma função de avaliação, deve-se usar simulações aleatórias
- Começando no estado atual (raiz da árvore de procura), iterar:
 - Selecionar um nó folha para expansão usando uma política de árvore (conflito de descoberta e exploração)
 - Executar uma simulação usando uma política padrão (por exemplo, movimentos aleatórios) até que um estado terminal seja alcançado
 - Propagar novamente o resultado para atualizar as estimativas de valor dos nós internos da árvore



- Damas:

- Chinook acabou com o reinado de 40 anos do campeão humano Marion Tinsley em 1994. Usava uma base de dados para finais de partida definindo a forma perfeita de vencer para todas as posições envolvendo 8 ou menos peças (num total de 443748401247 posições). Hoje em dia é um **problema resolvido**.

- Xadrez:

- Deep Blue derrotou o campeão do mundo humano **Gary Kasparov** num jogo com 6 partidas em 1997. Deep Blue procurava em 200 milhões de posições por segundo e usava uma função de avaliação extremamente sofisticada e métodos (não revelados) para estender algumas linhas de procura para além da profundidade 40!

- Go (2015):

- Os campeões humanos recusam-se a competir com computadores pois as **máquinas não conseguem jogar razoavelmente ($b>300$)**

- Go (2017):

- AlphaGo (Fan, Lee), AlphaGo Master, AlphaGo Zero and AlphaZero! As máquinas vencem 100-0 os campeões humanos e máquinas de gerações anteriores.

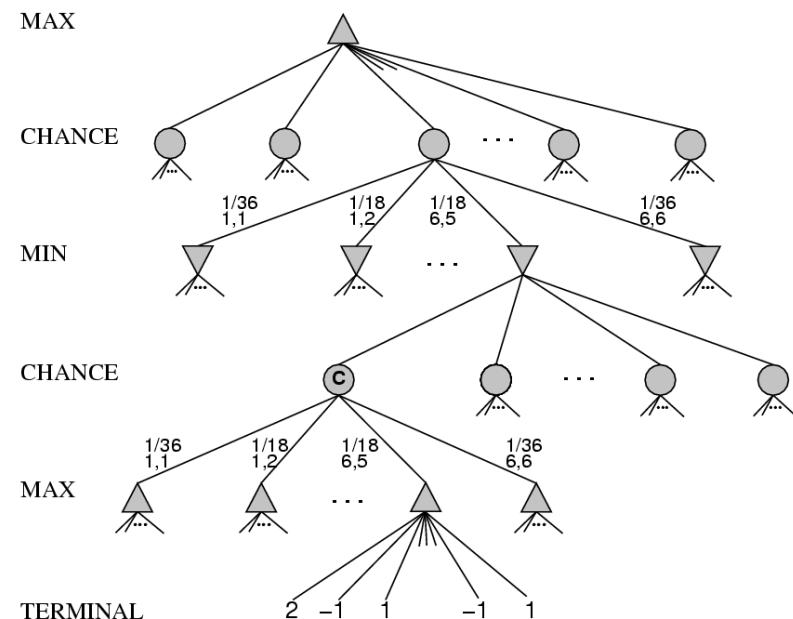
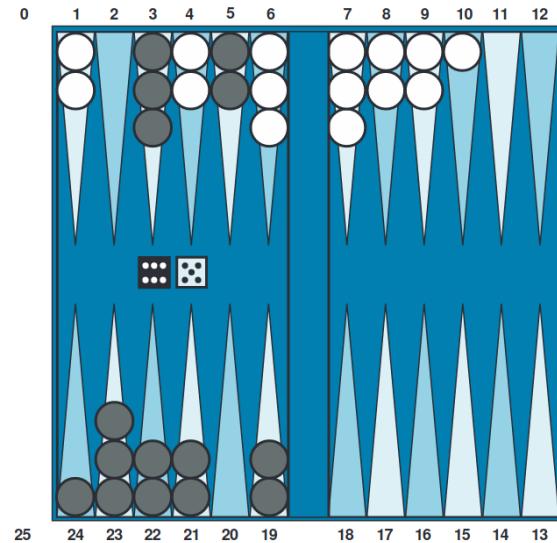
Jogos Estocásticos

- Na vida real, existem muitos eventos externos imprevisíveis que podem nos colocar em situações imprevistas. Muitos jogos refletem essa imprevisibilidade ao incluir um elemento aleatório (e.g., lançamento de dados).

	Determinístico	Estocástico
Informação perfeita (totalmente observável)	Xadrez Go, Damas	Gamão, Monopólio
Informação imperfeita (parcialmente observável)	Batalha Naval	Scrabble, Poker, Bridge

Jogos Estocásticos

- Jogos Estocásticos são jogos que tipicamente combinam habilidade e sorte;
 - Árvore de procura deverá incluir nós de probabilidade;
 - A decisão é efetuada com base no valor esperado;
 - *Algoritmo ExpectiMinimax.*



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Minimax vs. Expectiminimax

- **Minimax:**

- Maximize (over all possible moves I can make) the
- Minimum (over all possible moves Min can make) of the
- Reward

$$Value(node) = \max_{\text{my moves}} \left(\min_{\text{Min's moves}} (Reward) \right)$$

- **Expectiminimax:**

- Maximize (over all possible moves I can make) the
- Minimum (over all possible moves Min can make) of the
- Expected reward

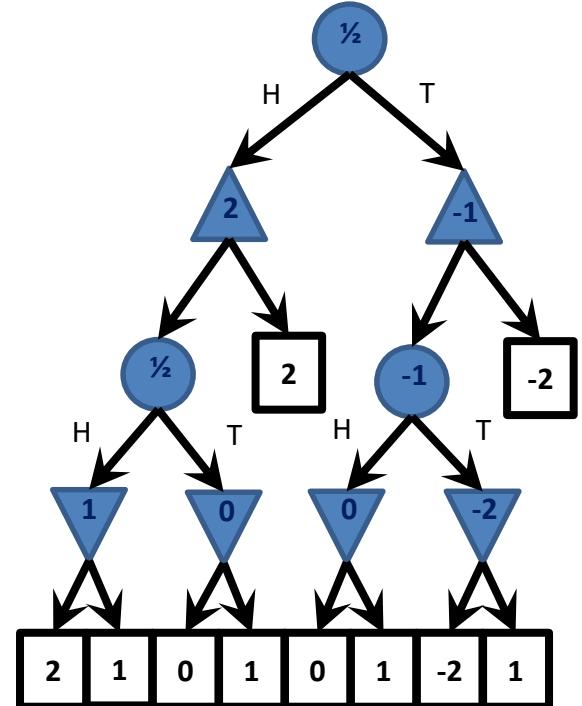
$$Value(node) = \max_{\text{my moves}} \left(\min_{\text{Min's moves}} (\mathbb{E}[Reward]) \right)$$

$$\mathbb{E}[Reward] = \sum_{outcomes} \text{Probability}(outcome) \times \text{Reward}(outcome)$$

- **Expectimax:** para nós sorte (chance) somar os valores dos estados sucessores ponderados pela probabilidade de cada sucessor.
- **Value(node) =**
 - Utility(*node*) if *node* is terminal
 - $\max_{action} \text{Value}(\text{Succ}(node, action))$ if *type* = MAX
 - $\min_{action} \text{Value}(\text{Succ}(node, action))$ if *type* = MIN
 - $\sum_{action} P(\text{Succ}(node, action)) * \text{Value}(\text{Succ}(node, action))$ if *type* = CHANCE

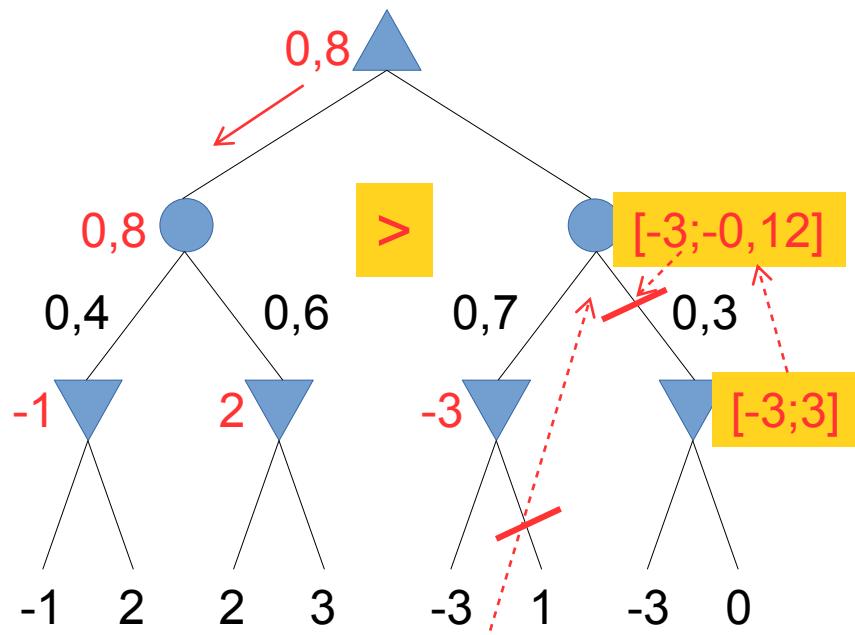
Jogos Estocásticos Exemplo

- ALEATÓRIO: Max lança uma moeda. Ou é Cara (H) ou coroa (T).
 - Para nas Cara: o jogo termina, Max vence (valor = € 2).
 - Para no Coroa: o jogo termina, Max perde (valor = € -2).
 - Continuar: o jogo continua
- ALEATÓRIO: Min lança uma moeda.
 - CARA-CARA: valor = € 2
 - Coroa-Coroa: valor = € - 2
 - Cara-Coroa or Coroa-Cara: valor = 0
- MIN: Min decide se deve manter o resultado atual (valor acima) ou pagar uma penalização (valor = € 1).



- Todos os métodos anteriores são úteis:
 - Poda (pruning) alfa-beta
 - Função de avaliação
 - etc
- A complexidade computacional é muito complicada
 - O fator de ramificação da escolha aleatória pode ser alto
 - O dobro dos "níveis" na árvore
- No Expectiminimax: para nós sorte (Chance), ele soma os valores dos estados sucessores ponderados pela probabilidade de cada sucessor.
 - Fator de ramificação pode ser muito desagradável, definindo funções de avaliação e algoritmos de poda mais difíceis
- Eventualmente, usar a Simulação de Monte Carlo: quando se chega a um nó sorte, simula-se um grande número de jogos com jogadas aleatórias de dados e usa-se a percentagem de ganhos como uma função de avaliação.
 - Pode funcionar bem para jogos como o Gamão.

Poda em jogos estocásticos



- se o ganho $\in [-3; 3]$ podem calcular-se limites e fazer podas exatas

Jogos Estocásticos com informação imperfeita (parcial)

Os jogos de cartas fornecem um dos melhores exemplos de jogos estocásticos com informação imperfeita (parcial), onde as informações não conhecidas (imperfeitas) advêm de aleatoriedade.

Exemplo: em muitos jogos, as cartas são distribuídas aleatoriamente no início do jogo, cada jogador recebe uma naipes de cartas que não é nem visível nem conhecida pelos os outros jogadores.

Jogos: Bridge, Copas e Poker.



Não se conhece completamente o estado do jogo

ex: póquer, bridge, king,... (aleatoriedade inicial e depois informação incompleta)

Soluções

- abstração – considerar diversos estados como semelhantes o que varia entre eles é pouco relevante
- meta-raciocínio – raciocínio acerca de valer ou não a pena raciocinar mais em certos estados/ramos do jogo

- **Minimax:**

- **Maximize** (over all possible moves I can make) the
- **Minimum**
 - (over all possible states of the information I don't know,
 - ... over all possible moves Min can make) the
- **Reward.**

$$Value(node) = \max_{my\ moves} \left(\min_{\substack{missing\ info, \\ Min's\ moves}} (Reward) \right)$$

Jogos Estocásticos com informação imperfeita Técnicas

- Se conhecemos as probabilidades de diferentes configurações e desejamos maximizar os ganhos médios (por exemplo, se podemos jogar o jogo muitas vezes): usa-se o **expectimax**
- Se não temos ideia das probabilidades de diferentes configurações; ou, se podemos apenas jogar uma vez e não podemos (nem queremos) perder: usa-se o **minimax**
- Se a informação desconhecida foi selecionada intencionalmente pelo oponente: usa-se a **teoria dos jogos**

- Trabalhar com jogos é extremamente interessante porque é:
 - fácil testar novas ideias
 - fácil comparar agentes com outros agentes
 - fácil comparar agentes com humanos
- Os jogos ilustram diversos pontos interessantes da IA:
 - Perfeição é inatingível => é necessário aproximar!
 - É uma boa ideia pensar sobre o que pensar
 - A Incerteza restringe a atribuição de valores aos estados
- Jogos funcionam para a IA como a Fórmula 1 para a construção de automóveis...

Bibliografia Recomendada

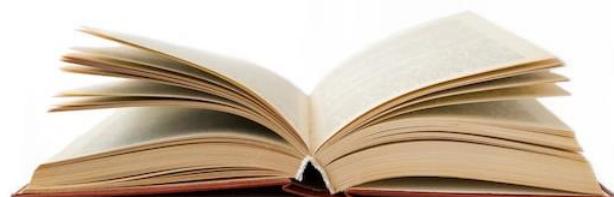
- Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach, 3rd edition, ISBN-13: 9780136042594, Chapter 5.

Outro material

- Svetlana Lazebnik, Lecture notes Fall 2017 Artificial Intelligence, University of Illinois.
- Luís Paulo Reis, Lecture notes Artificial Intelligence (2019), Universidade do Porto

Para quem quer ir mais longe

AlphaGo: Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016). <https://doi.org/10.1038/nature16961>





Universidade do Minho
Escola de Engenharia
Departamento de Informática

MÉTODOS DE RESOLUÇÃO DE PROBLEMAS E DE PROCURA Procura em contextos competitivos (Jogos)

LICENCIATURA EM ENGENHARIA INFORMÁTICA
MESTRADO integrado EM ENGENHARIA INFORMÁTICA
Inteligência Artificial
2022/23