Jogos

Jogos vs. problemas de busca

Jogos são mais difíceis do que problemas usuais de busca.

Oponente é "imprevisível" $\Rightarrow$ problema de contingência

Limites de tempo $\Rightarrow$ improvável achar meta, tem que aproximar

Uma longa história:

- Algoritmo para jogo perfeito (Von Neumann, 1944)
- Horizonte finito, avaliação aproximada (Zuse, 1945; Shannon, 1950; Samuel, 1952–57)
- Poda para reduzir custos (McCarthy, 1956)

Tipos de jogos

Jogo de informação perfeita: ambiente acessível.

Jogo de informação imperfeita: ambiente parcialmente inacessível

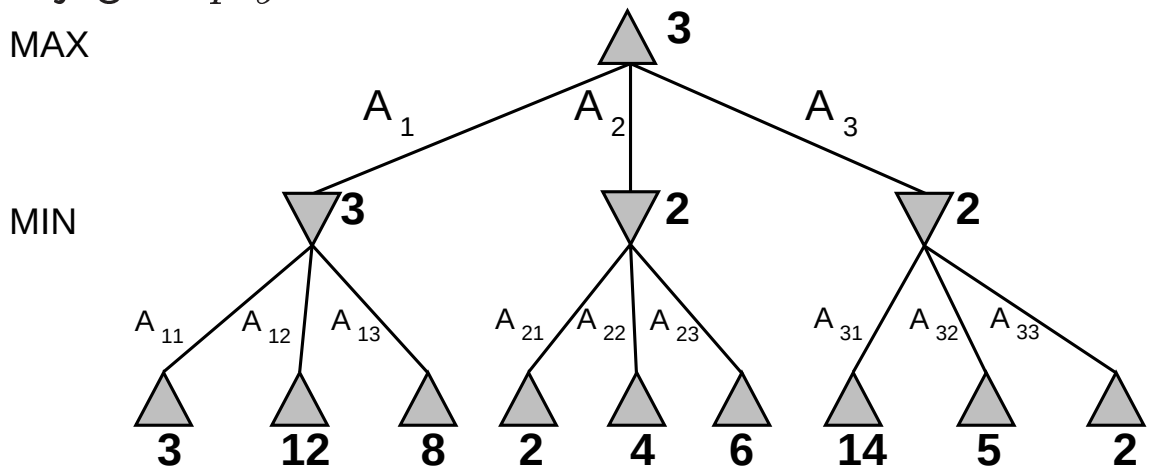
	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information		bridge, poker, scrabble nuclear war

Minimax

Propõe jogadas para jogos determinísticos de informação perfeita

Idéia: escolha movimento para posição com mais alto valor $minimax$ = **melhor** jogada contra a **melhor** jogada do adversário

Por exemplo, jogo 2-ply:



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*) **then**
return UTILITY[*game*](*state*)
else if MAX is to move in *state* **then**
return the highest MINIMAX-VALUE of SUCCESSORS(*state*)
else
return the lowest MINIMAX-VALUE of SUCCESSORS(*state*)

Propriedades (e problemas) de Minimax

Completo?? Sim, se árvore for finita (xadrez têm regras específicas para garantir isto)

Ótimo?? Sim, contra um oponente ótimo. Caso contrário?

Complexidade no tempo?? $O(b^m)$

Complexidade no espaço?? $O(bm)$ (exploração *depth-first*)

Para xadrez, $b \approx 35$, $m \approx 100$ para um jogo "razoável"
$\Rightarrow$ solução exata impraticável

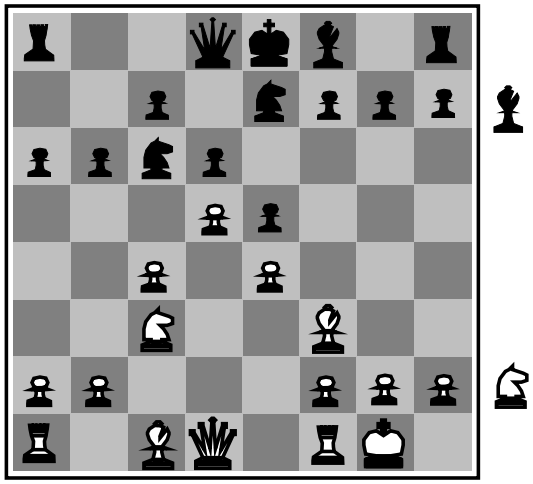
Enfoque usual para lidar com limitações de recursos computacionais:

teste de corte, e.g., limite de profundidade com uso de *função de avaliação* = "desejabilidade" estimada da posição

Busca quiescente: avaliação heurística de posição não é recomendável em posições "instáveis" $\Rightarrow$ melhor usá-la apenas se posições forem estáveis com respeito à heurística. Exemplo ...

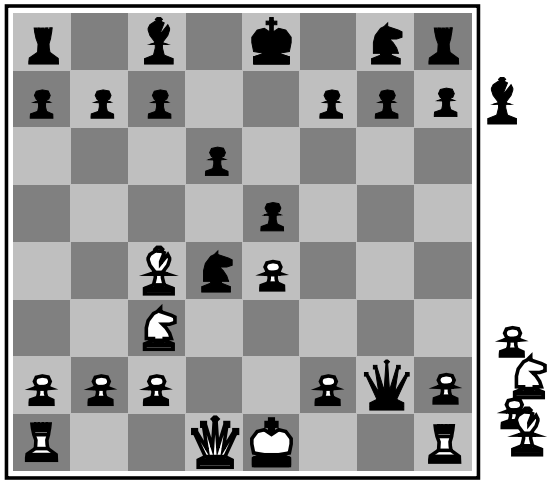
Problema do horizonte: Avaliação com teste de corte pode ignorar consequências no futuro distante, a não ser que heurística seja muito sofisticada.

Funções de avaliação



Black to move

White slightly better



White to move

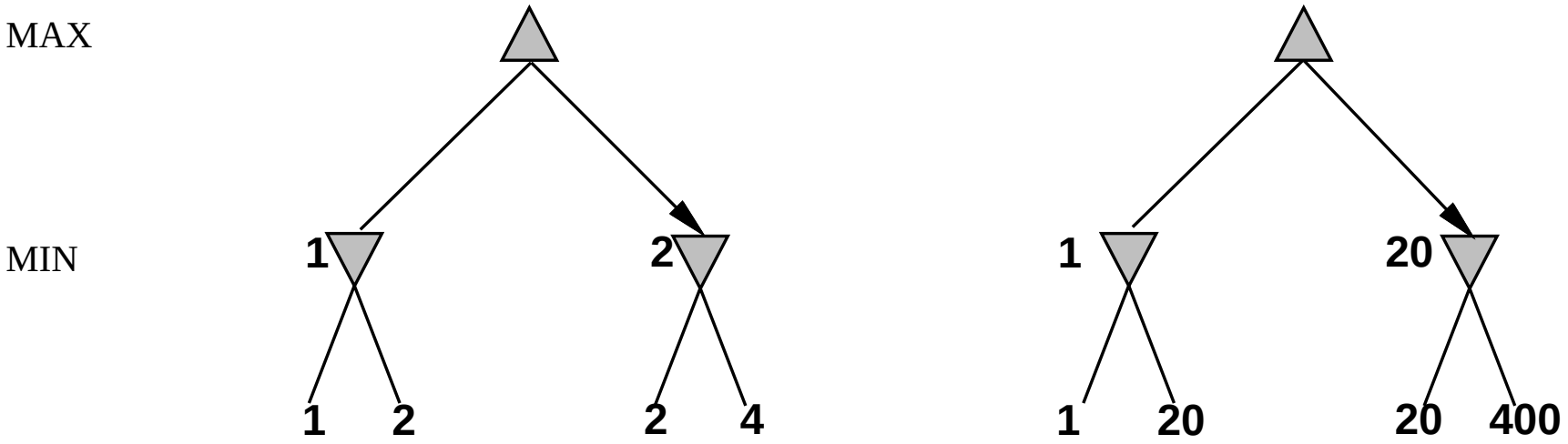
Black winning

Para xadrez, tipicamente uma soma ponderada de *features*

$$\textsc{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

por exemplo, $w_1 = 9$ com
$f_1(s) =$ (número de rainhas brancas)–(número de rainhas pretas)
etc.

Observação: Valores exatos não importam



O comportamento é preservado sob qualquer transformação *monotônica* de Eval

Só a ordem importa:

A avaliação em jogos determinísticos atua como uma função de *ordenamento*

Eliminando a busca

MinimaxCutoff é idêntico a MinimaxValue exceto por

1. Terminal? é substituído por Cuttof?
2. Utility é substituído por Eval

Funciona na prática? Suponha capacidade de processar 10^6 nós por jogada, jogo de xadrez (fator de ramificação aprox. 35):

$$b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4$$

4-*ply lookahead* é um fraquíssimo jogador de xadrez!

4-*ply* $\approx$ um iniciante
8-*ply* $\approx$ PC típico, mestre humano
12-*ply* $\approx$ Kasparov, Deep Blue

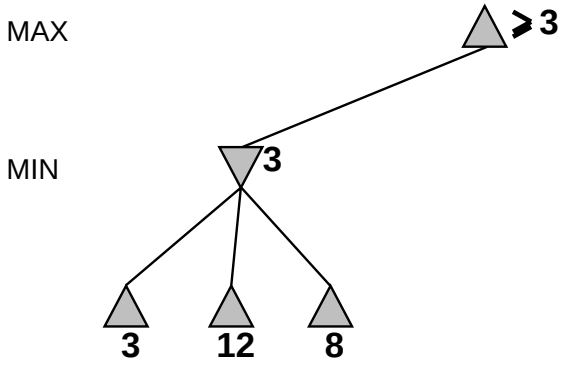
Poda alpha-beta

Observe a ineficiência de MINIMAX: completa geração da árvore e só então avaliação de posições.

Idéia da poda alpha-beta: geração e avaliação simultâneas.

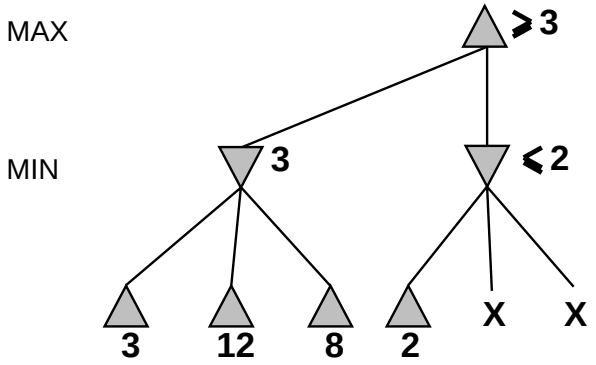
Benefício: posso evitar gerar ramos da árvore originários de nós com avaliação baixa (do ponto de vista de MAX) ou alta (do ponto de vista de MIN).

Exemplo:

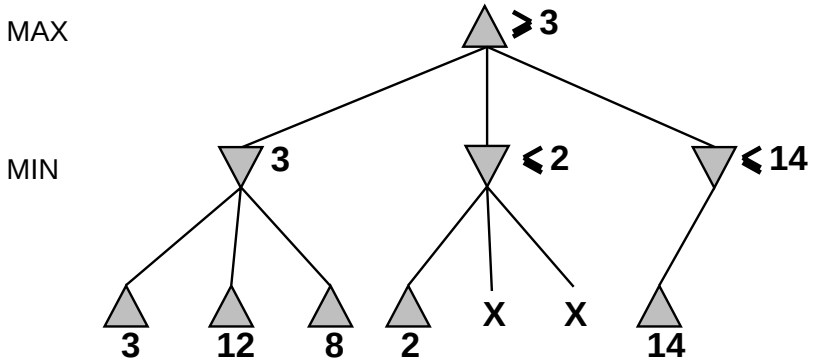


MAX garante uma avaliação de no mínimo 3 para o nó inicial. Este **limite inferior** é chamado α.

Poda alpha-beta: exemplo

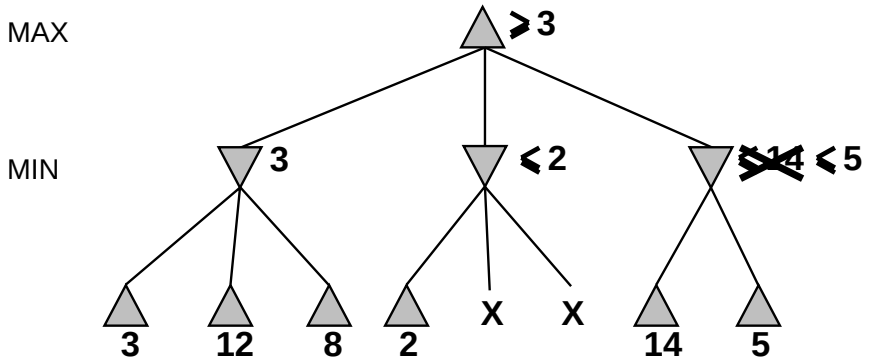


MIN garante uma avaliação de no máximo 2 para o nó. Este **limite superior** é chamado β. Poda alpha-beta: nó ancestral com $\alpha > \beta$ de nó descendente $\rightarrow$ Ramo ≤ 2 é "podre", MAX já garantiu jogar em um ramo melhor. Nó que origina ramo podre recebe β como seu valor estático.

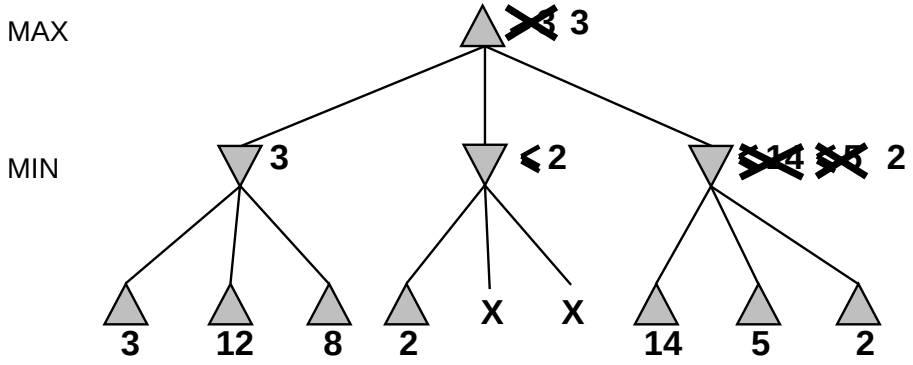


MIN garante uma avaliação de no máximo 14 para o nó. Este nó tem portanto $\beta = 14$ (por enquanto). Poda alpha-beta: nó ancestral com $\alpha < \beta$ de nó descendente $\rightarrow$ nada se pode afirmar, talvez este ramo seja melhor para MAX.

Poda alpha-beta: exemplo



Novo valor $\beta = 5$: MIN agora garante uma avaliação de no máximo 5. Poda alpha-beta: nó ancestral com $\alpha < \beta$ de nó descendente $\rightarrow$ nada se pode afirmar, talvez este ramo seja mais proveitoso para MAX.



Novo valor $\beta = 2$: MIN agora garante uma avaliação estática de no máximo 2. Poda alpha-beta: nó ancestral com $\alpha > \beta$ de nó descendente $\rightarrow$ MAX já garantiu ramo melhor. Valor estático final do nó inicial assume o valor α.

Poda alpha-beta: descrição

- Valores α de nós MAX nunca diminuem.
- Valores β de nós MIN nunca aumentam.

Então, a busca pode ser descontinuada:

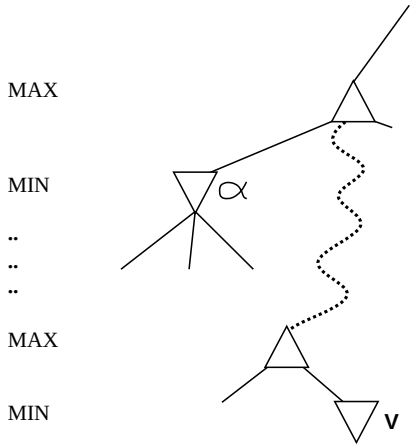
- Abaixo de qualquer nó MIN com valor $\beta \leq$ ao valor α de algum de seus ancestrais MAX.
- Abaixo de qualquer nó MAX com valor $\alpha \geq$ ao valor β de algum de seus ancestrais MIN.

Durante a busca:

- O valor α de um nó MAX é o maior valor dos seus sucessores atuais.
- O valor β de um nó MIN é o menor valor dos seus sucessores atuais.

Observe que poda alpha-beta usa busca *em profundidade* ...

Ilustração da poda alpha-beta



α é o melhor valor (para MAX) achado até o momento. Se β é pior que α, MAX o evitará $\Rightarrow$ poda do ramo "podre"

Propriedades:

$\diamondsuit$ Poda **não afeta** resultado final

$\diamondsuit$ Boa ordenação aumenta efetividade da poda: com "ordenamento perfeito," complexidade no tempo $= O(b^{m/2})$

$\Rightarrow$ **Dobra** profundidade da busca

$\Rightarrow$ Pode alcançar profundidade 8 e jogar xadrez de alto nível

Jogos determinísticos na prática

Damas: Chinook terminou o reinado de 40 anos do campeão do mundo Marion Tinsley em 1994. Usou um banco de dados de finais de jogos definindo o jogo perfeito para todas as posições envolvendo 8 ou menos peças no tabuleiro, num total de 443.748.401.247 posições.

Xadrez: Deep Blue derrotou o campeão de mundo Gary Kasparov num $match$ de seis jogos em 1997. Deep Blue procura 200 milhões de posições por segundo, usa uma avaliação de posições muito sofisticada, e métodos não-revelados para estender algumas linhas de busca até 40 ply.

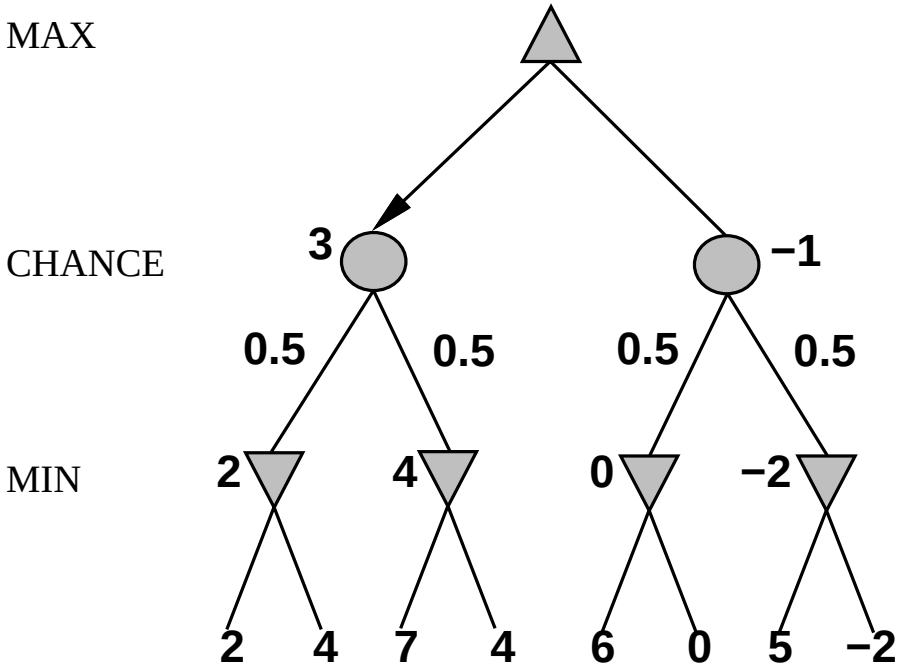
Go: campeões humanos se recusam a competir contra computadores, que são ruins demais. Em Go, $b > 300$, assim a maioria dos programas usa reconhecimento de padrões para sugerir movimentos plausíveis.

Jogos não-determinísticos

Aparece um "terceiro jogador": o fator aleatório.

E.g., no gamão os dados determinam os movimentos legais $\Rightarrow$ um "jogador" chamado "dados" interpõe-se a jogadas de MAX e MIN.

Exemplo simplificado com moeda em vez de dado:



Algoritmo para jogos não-determinísticos

EXPECTIMINIMAX proporciona jogo perfeito

Igual à MINIMAX, mas considerando-se nós fortuitos:

...
se *estado* é um nó fortuito **então** uso
EXPECTIMINIMAX-VALUE de SUCCESSORS(*estado*)
...

$$\text{expectmax}\ (C) = \sum_i P(d_i) \max_{s \in S(C,d_i)} (\text{utilidade}(s)) \quad (1)$$

$$\text{expectmin}\ (C) = \sum_i P(d_i) \min_{s \in S(C,d_i)} (\text{utilidade}(s)) \quad (2)$$

Uma versão de poda alpha-beta é possível
mas só se os valores de folha são limitados (*bounded*) por um valor conhecido.
Como??

Jogos não-determinísticos na prática

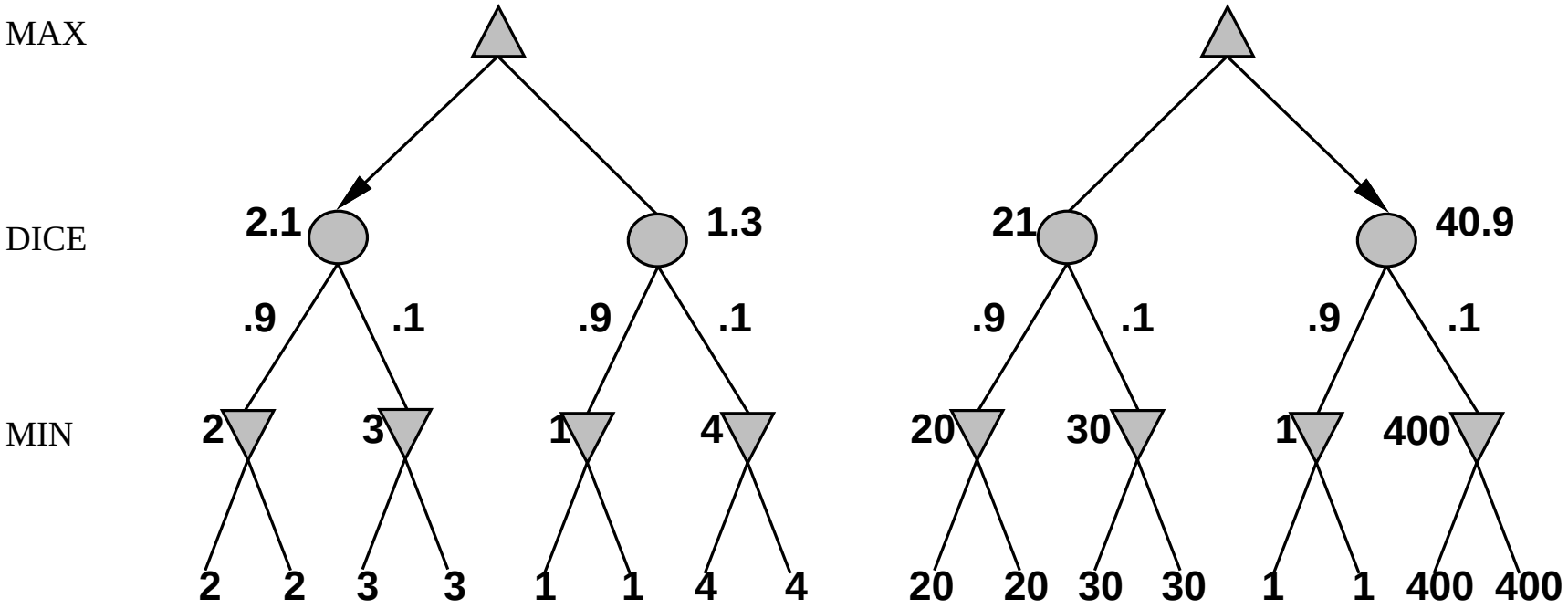
Lançar dados aumenta b: 21 possíveis resultados com 2 dados
Gamão tem $\approx$ 20 movimentos legais por posição

Com o aumento da profundidade, a probabilidade de se alcançar um determinado nó diminui $\Rightarrow$ valor do $lookahead$ é diminuído.

Poda alpha-beta é muito menos efetiva.

TDGammon usa busca de profundidade 2 + muito boa Eval
$\approx$ nível do campeão do mundo. É baseado em Aprendizado por Reforço: começa sabendo apenas as regras, e melhora gradualmente jogando consigo mesmo várias vezes e usando como indicativo de desempenho apenas o resultado final de cada jogo (vitória ou derrota), propagada a situações anteriores que dividem a responsabilidade pelo valor final.

Observação: Valores exatos importam



O comportamento não é mais preservado se apenas garantirmos um reescalonamento que preserve a ordem.

O comportamento só é preservado por uma transformação *linear positiva* de EVAL.