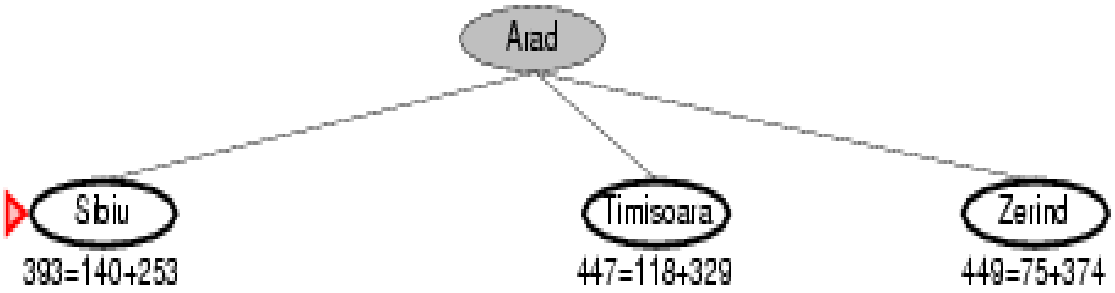
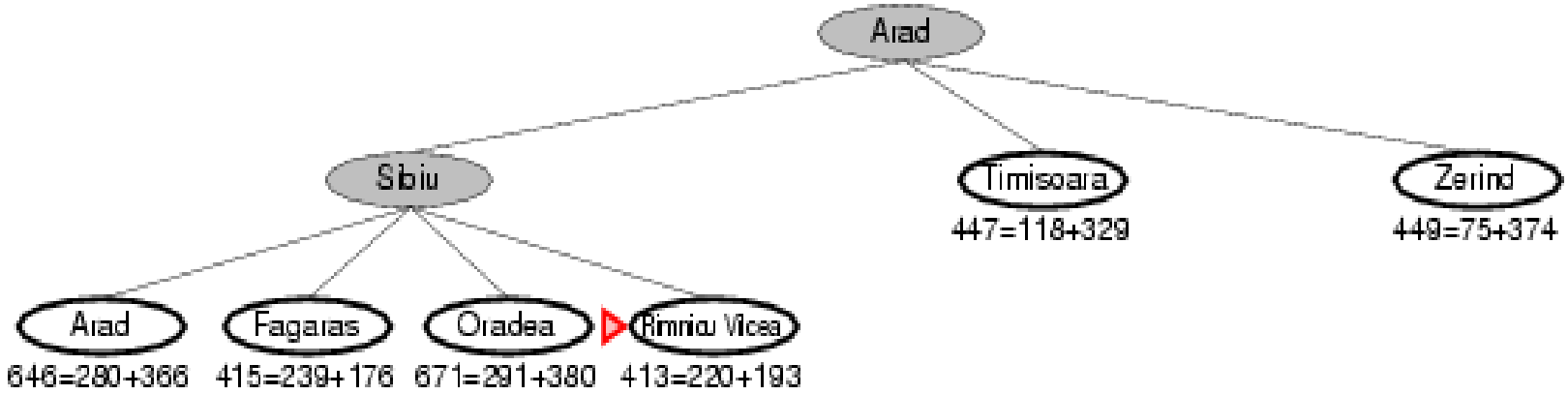
Busca A*: exemplo

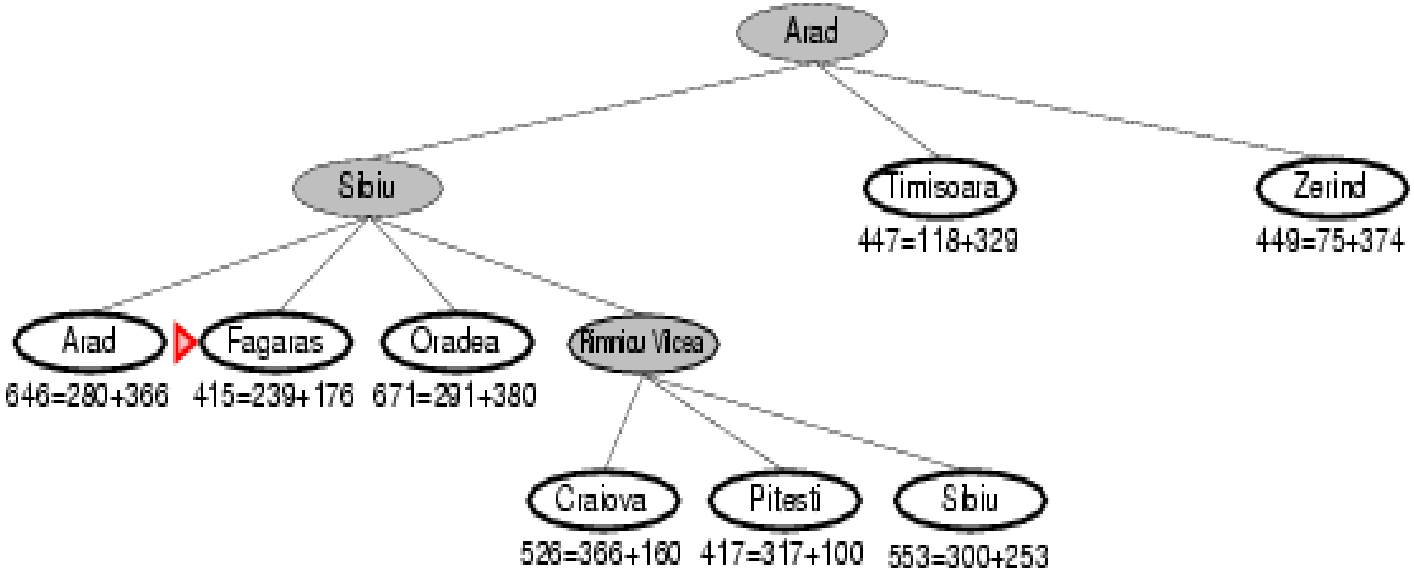
Busca A*: exemplo



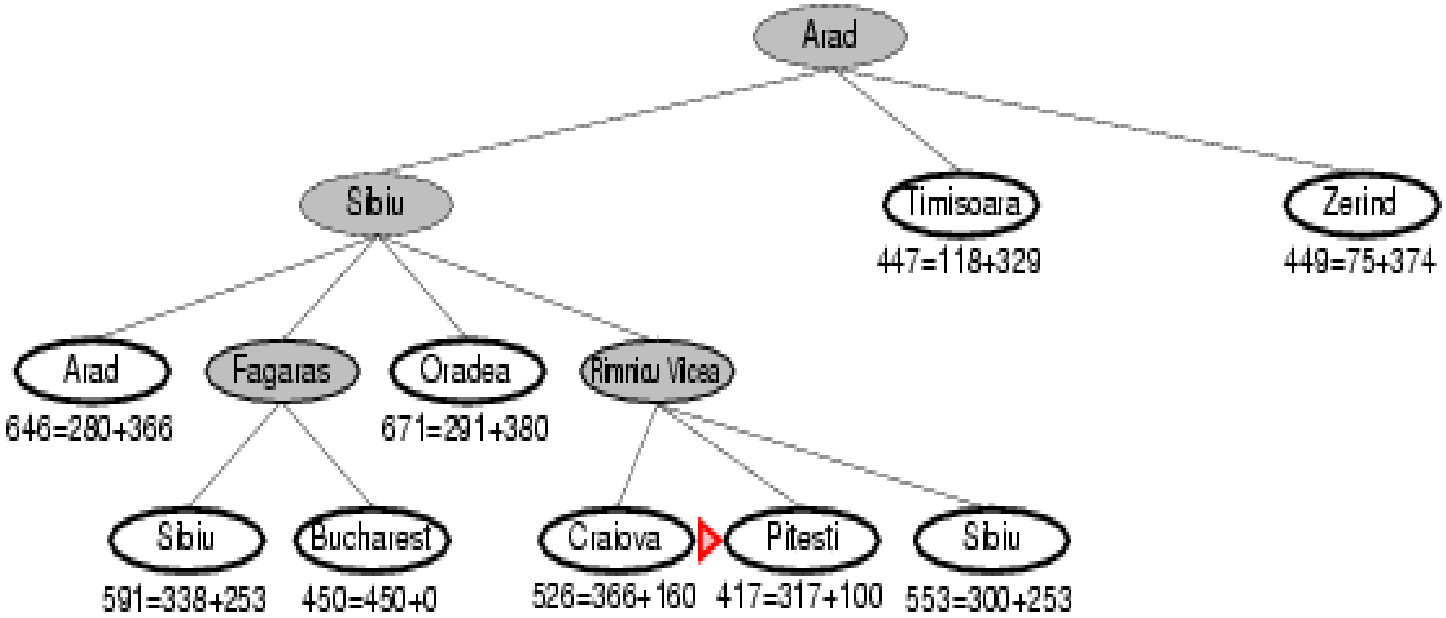
Busca A*: exemplo



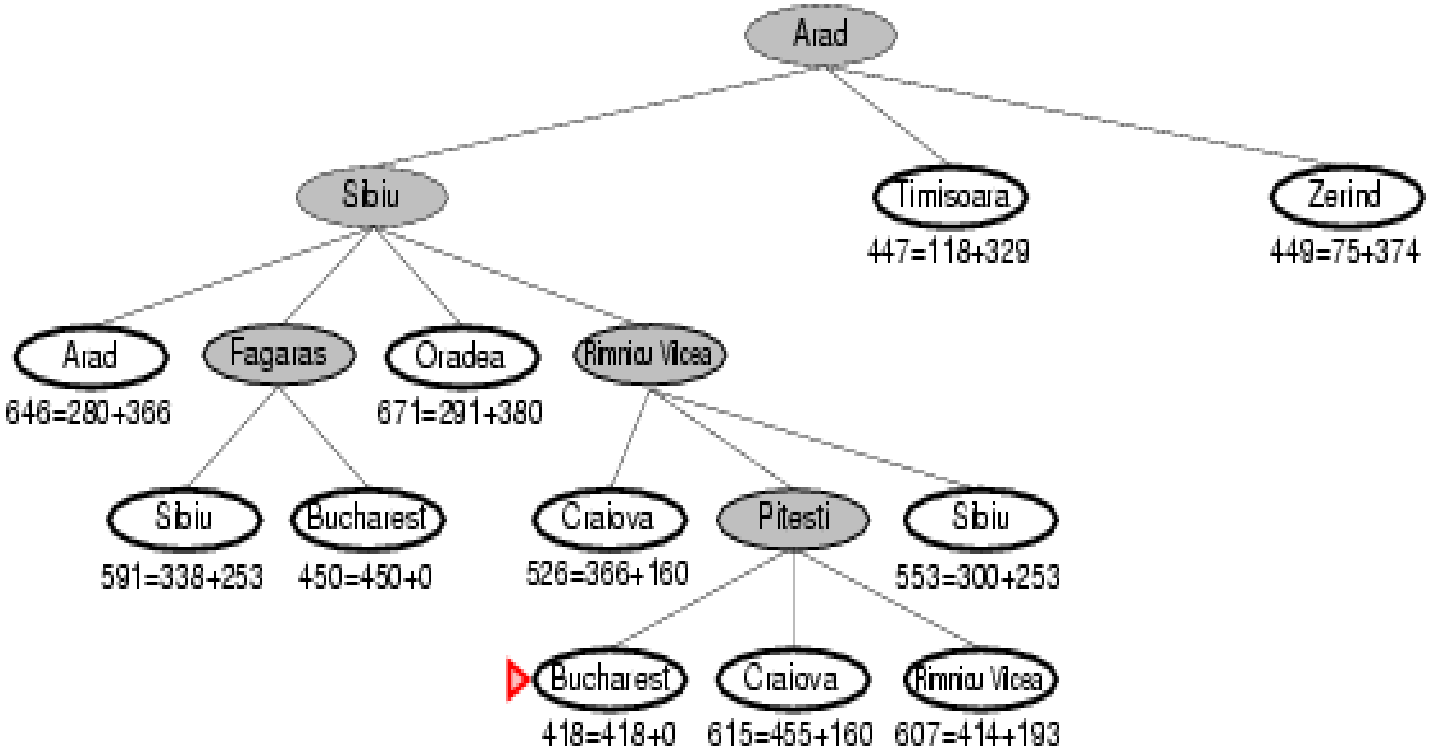
Busca A*: exemplo



Busca A*: exemplo

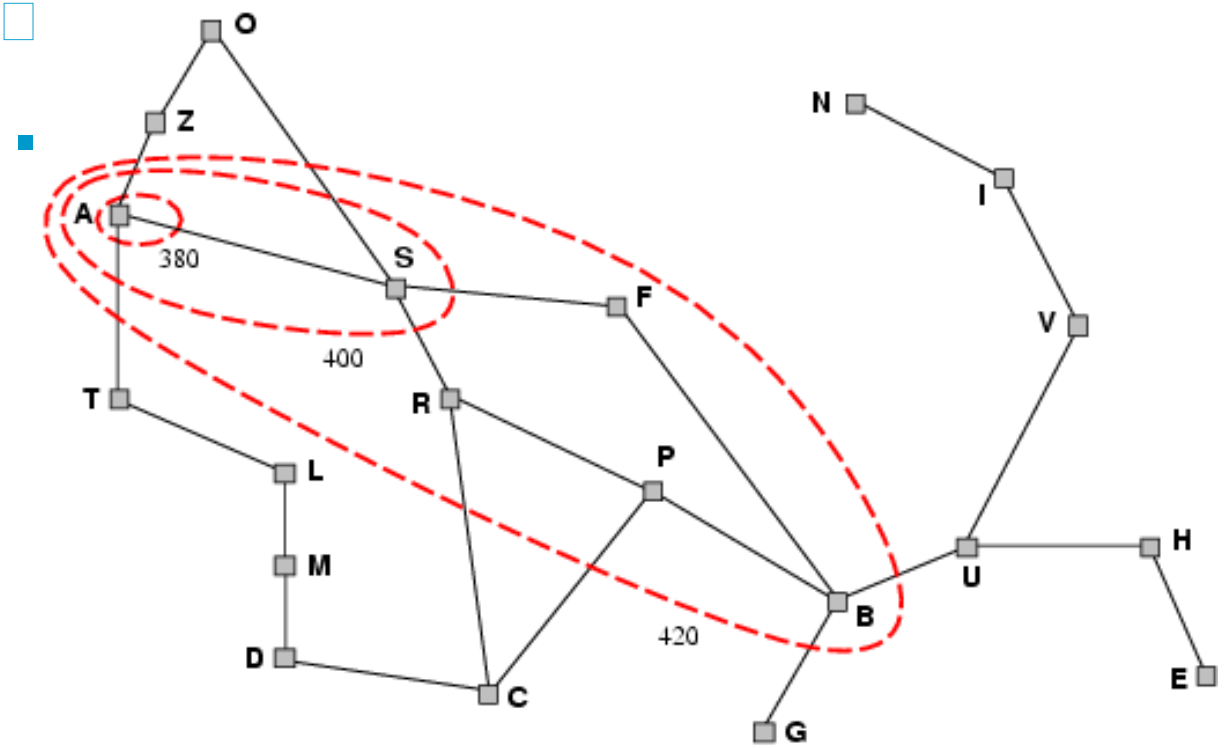


Busca A*: exemplo



Optimalidade do A*

- A* expande nós seguindo o maior valor de *f*
- Gradualmente adiciona ”contornos *f*" de nó
- Contorno *i* possui todos os nós f_i, onde $f_i < f_{i+1}$



Até aqui...

- Problemas sem interação com outro agente;
- o agente possui total controle sobre suas ações e sobre o efeito de suas ações;
- muitas vezes encontrar a solução ótima é factível.

Busca competitiva (jogos “adversariais”)

Aula 4 - Cap. 6 Russell & Norvig

Fundamentos da IA

Mestrado - FEI - 2005

Jogos

- São domínios clássicos em IA
  - abstratos: fáceis de formalizar e representar;
  - podem ter sua complexidade reduzida ou aumentada;
  - exigem a tomada de decisões (muitas vezes em um curto intervalo de tempo);
  - Há interação e pode haver não determinismo.

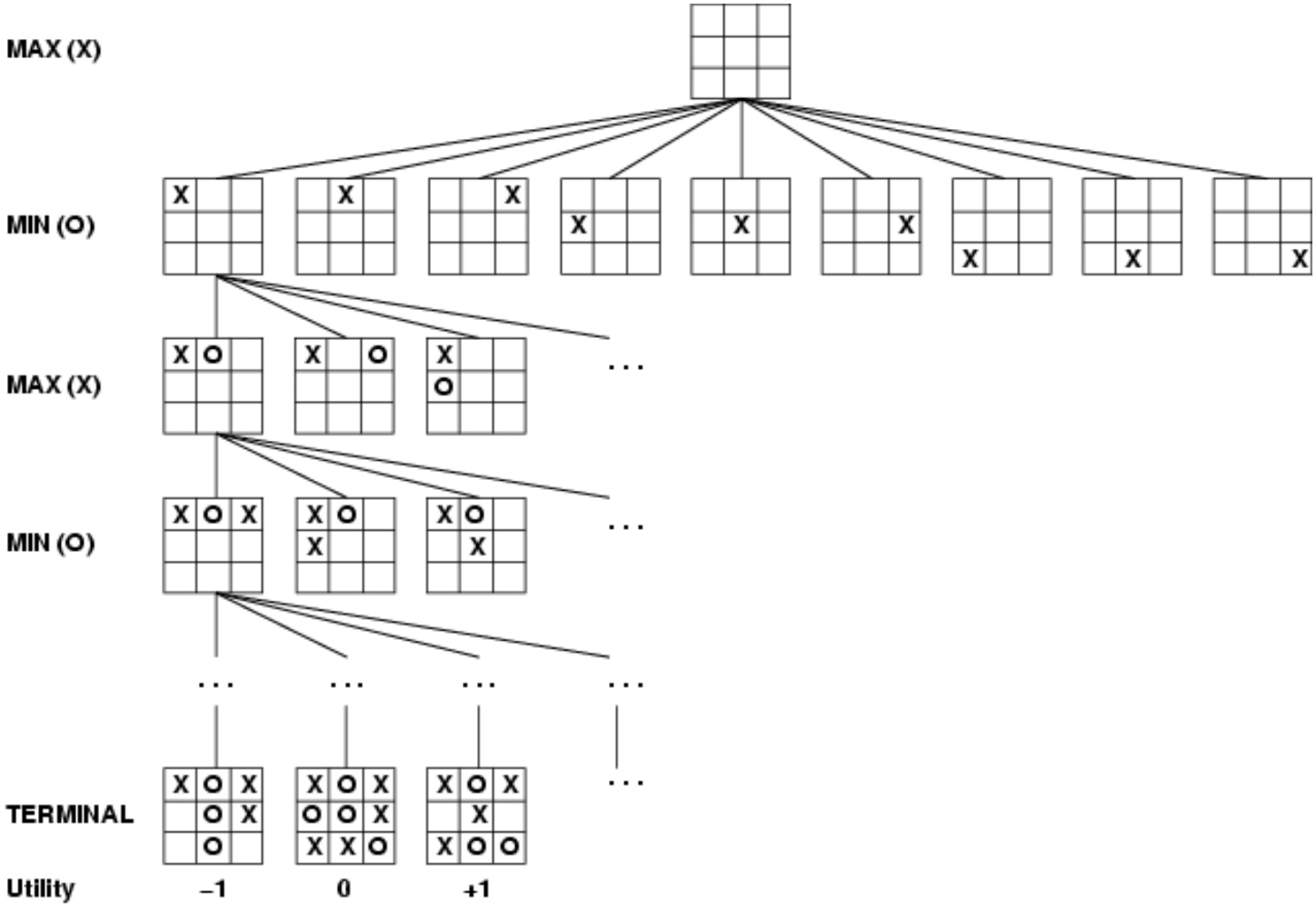
Jogos vs. busca

- O oponente é “imprevisível”
  - levar em consideração todos os movimentos possíveis de oponente;
- Limite de tempo
  - tomar uma decisão, mesmo que não seja ótima.

Decisões ótimas em jogos

- Inicialmente jogos com dois jogadores:
  - 'MAX' e 'MIN';
- Um jogo pode ser definido como uma árvore de busca:
  - estado inicial
  - função sucessor (-> movimento, estado)
  - teste de término
  - função utilidade: dá um valor numérico para os estados terminais

Árvore de jogo (2 jogadores)



Do ponto de vista de max, valores altos de utilidade são bons.

Estratégias ótimas

- A solução ótima para MAX depende dos movimentos de MIN, logo:
  - MAX deve encontrar uma *estratégia de contingência* que especifique o movimento de MAX no estado inicial, e depois o movimento de MAX nos estados resultantes de cada movimento de MIN e assim por diante...

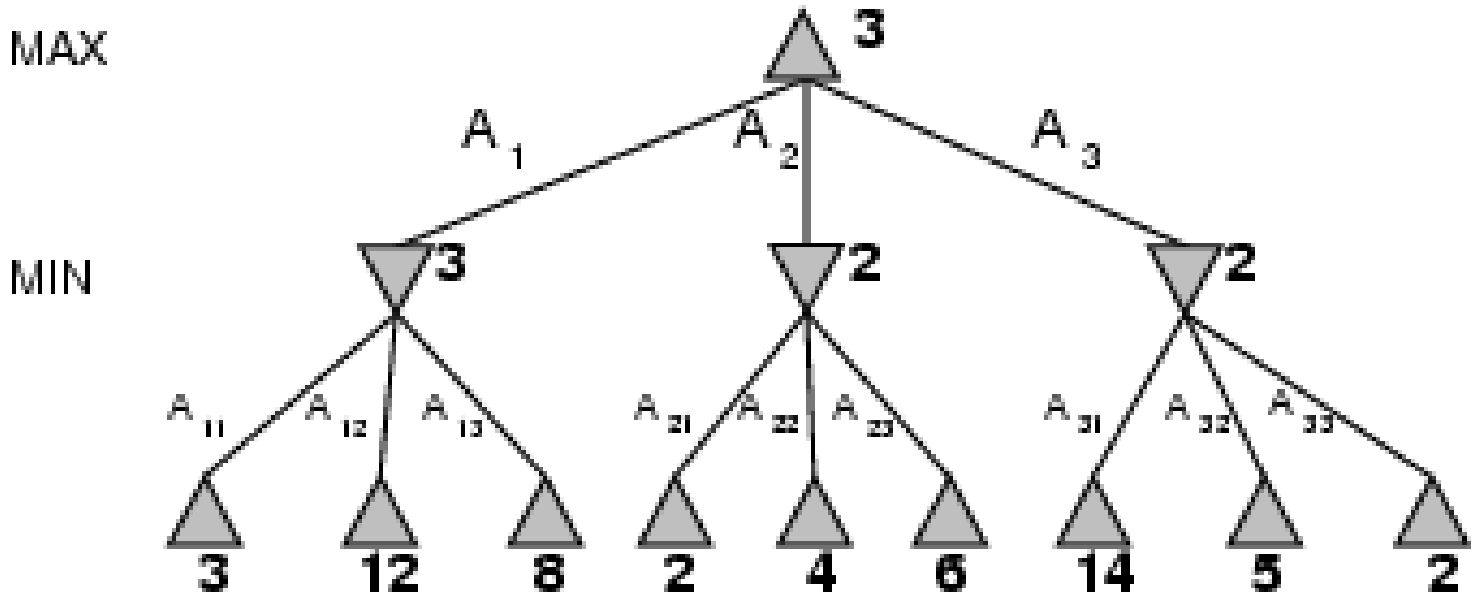
Estratégias ótimas

- Dada uma árvore de jogo, a estratégia ótima pode ser determinada a partir do valor ***minimax*** de cada nó.
- O valor minimax (para MAX) é a utilidade de MAX para cada estado, ***assumindo que MIN escolhe os estados mais vantajosos*** para ele mesmo (i.e. os estado com menor valor utilidade para MAX)

Valor minimax

- UTILIDADE(n) se n é terminal
- $\max_{x \in Succ(n)}$ Valor Minimax(s) se n é um nó de MAX
- $\min_{x \in Succ(n)}$ Valor Minimax(s) se n é um nó de MIN

Minimax



A ação *a1* é a escolha ótima para MAX, porque leva ao sucessor com mais alto valor minimax.

A melhor jogada para um jogo determinístico assumindo o melhor oponente.

Algoritmo minimax

```
function MINIMAX-DECISION(state) returns an action
    v ← 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 ← −∞
    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
```

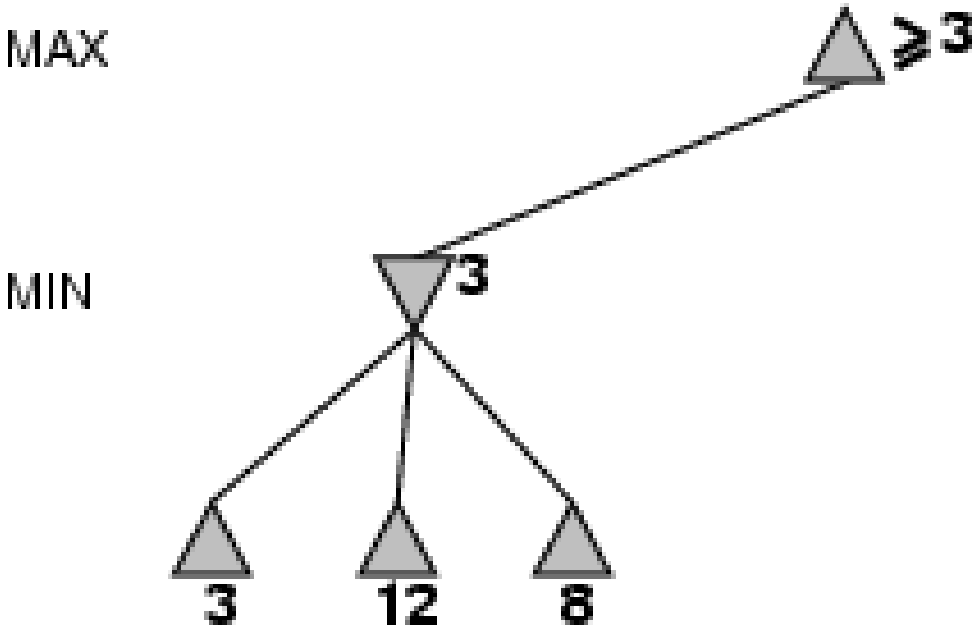
Algoritmo minimax

- Ótimo (para um oponente ótimo);
- Tempo: busca completa em profundidade na árvore do jogo: $O(b^m)$
  - m: profundidade
  - b: movimentos válidos em cada estado
- Espaço:
  - O(bm) se todos os sucessores são gerados
  - O(m) se gera um sucessor por vez

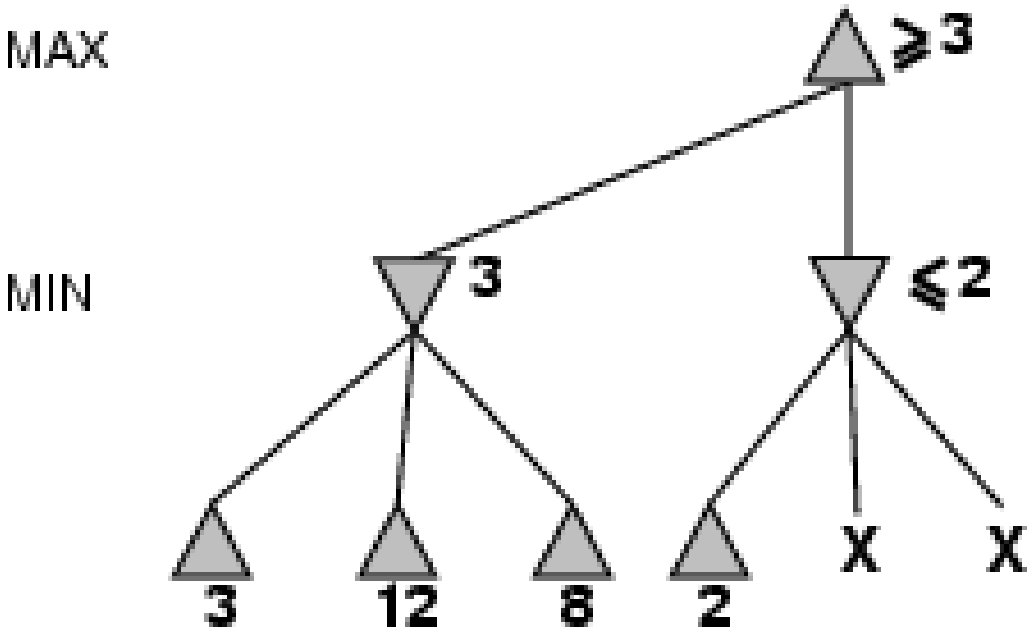
Poda α-β

- Busca minimax: nº de estados do jogo é exponencial em relação ao nº de movimentos;
- Poda α-β :
  - calcular a decisão correta sem examinar todos os nós da árvore;
  - retorna o mesmo que minimax, porém sem percorrer todos os estados.

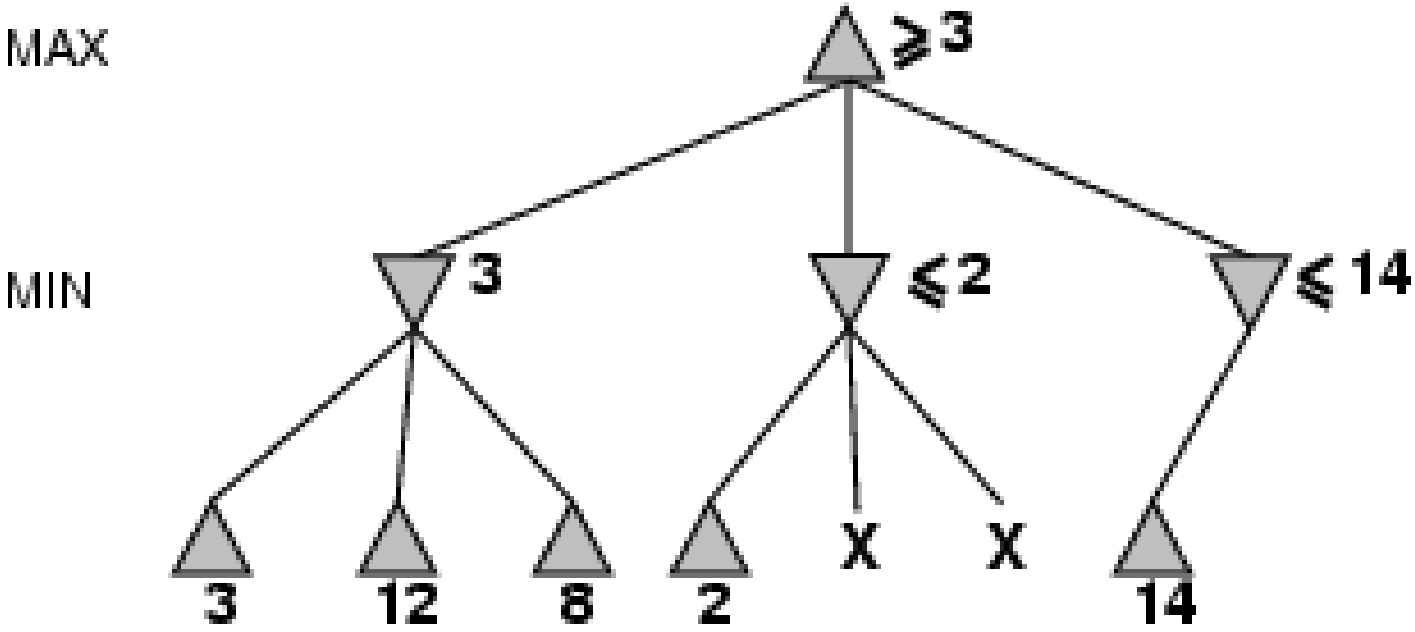
Poda α-β



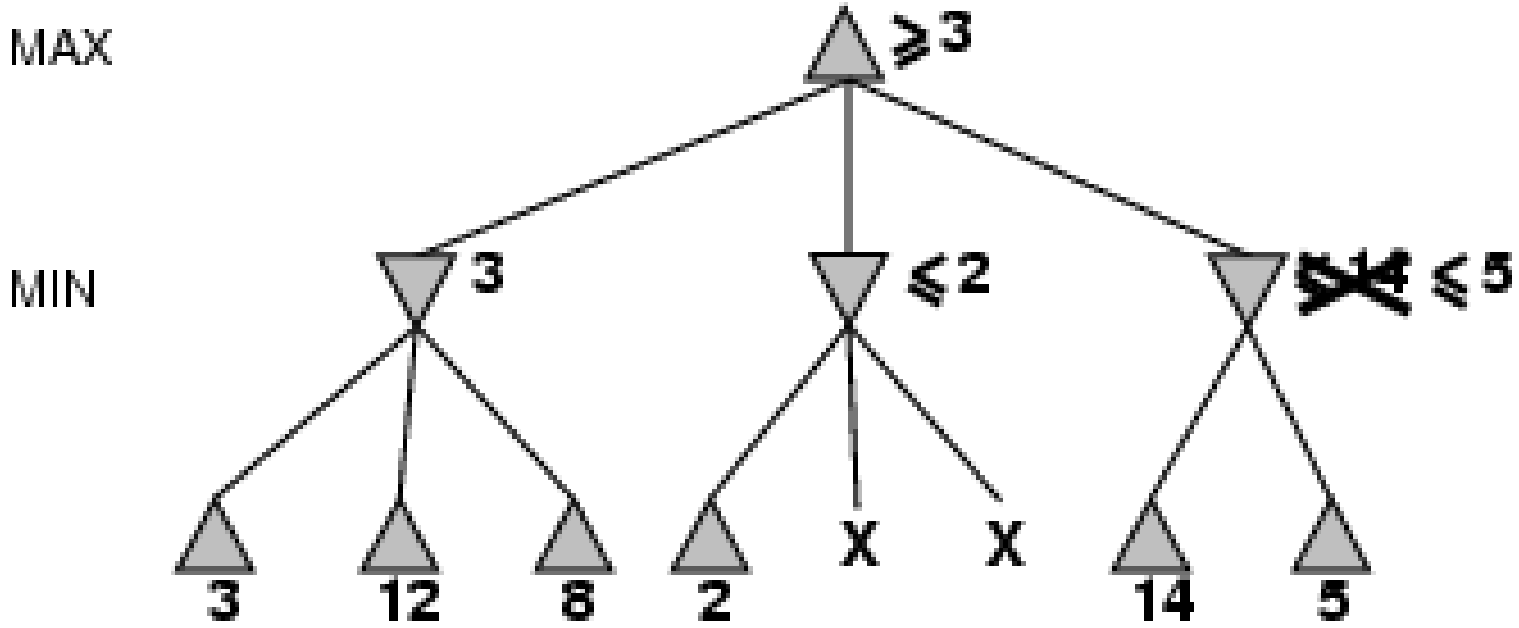
Poda α-β



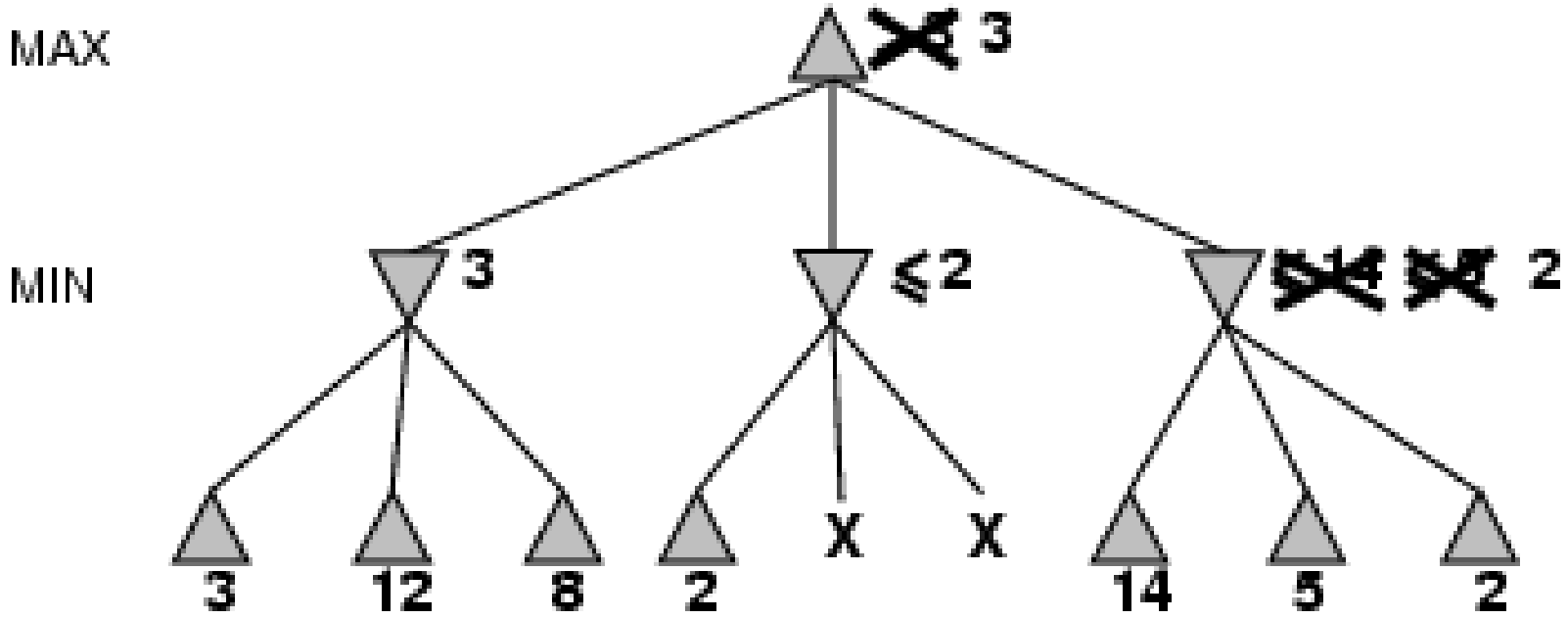
Poda α-β



Poda α-β



Poda α-β



Poda α-β

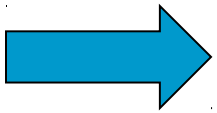
- A efetividade da poda α-β depende da ordem em que os sucessores são examinados

Por que “α-β” ?

- α é o valor da melhor escolha (valor mais alto) encontrado até então para qqr ponto de escolha de MAX;
- Se *v* é pior do que α, MAX não percorrerá este caminho (irá podar este ramo de busca)
- β é definido de maneira análoga.

Decisões imperfeitas em tempo real

- Minimax gera o espaço de busca todo;
- Poda α-β ainda tem que chegar até os estados terminais



São ineficientes para jogos que possuam muitos passos para os estados terminais... I.e., quase todos os jogos interessantes!

Decisões imperfeitas em tempo real

- Sugestão (Shannon, 50):
  - substituir a função utilidade por uma ***função de avaliação heurística*** e substituir o teste de término por um teste de *corte*;
  - Função de avaliação retorna uma *estimativa* da utilidade esperada do jogo a partir de uma dada posição
  - I.e., nós não terminais se transformam em nós terminais para minimax ou corte α-β.

Decisões imperfeitas em tempo real

- ***função de avaliação heurística***
  - Deve ordenar nós terminais da mesma forma que a função utilidade;
  - A computação deve ser rápida;
  - Em estados não terminais a função de avaliação deve prover as chances reais de vitória;
    - o algoritmo será necessariamente incerto com relação aos resultados finais pois a busca será cortada!

Decisões imperfeitas em tempo real

- Definição de função de avaliação heurística: ***características de estado***
  - em conjunto definem *categorias* ou classes de *equivalência de estados* (ex. número de peões tomados);
    - Estados de cada categoria têm os mesmos valores para cada característica;
  - calcula contribuições numéricas separadas de cada característica e as combina para gerar um resultado final...

Decisões imperfeitas em tempo real

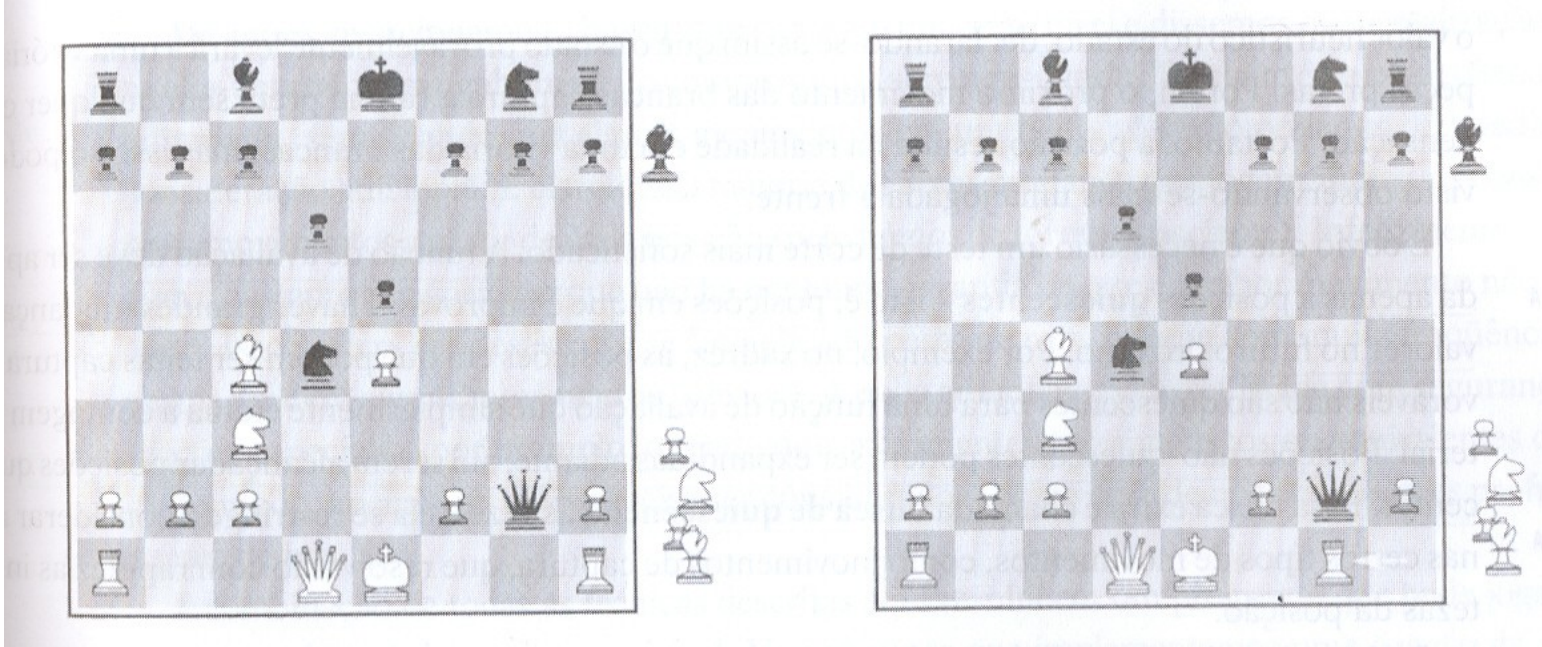
- Definição de função de avaliação heurística**: exemplo xadrez**:
  - valor material de cada peça: peão=1, cavalo ou bispo=3, torre=5, rainha=9
  - boa estrutura de peões, segurança do rei = 1/2 peão
- função de avaliação: f. linear ponderada
  - AVAL(s) = w1f1(s)+ w2f2(s)+... wnfn(s)

Decisões imperfeitas em tempo real

- função de avaliação: f. linear ponderada
  - AVAL(s) = w1f1(s)+ w2f2(s)+... wnfn(s)
- Em somar os valores de características assumimos que as contribuições de cada característica são independentes das outras.
  - *Ex. ignora o fato de um bispo ser mais valioso no fim do jogo e um cavalo no início*
- ***É possível usar combinações não lineares...***

- Características vem da experiência em um determinado domínio...
- Se esta experiência não existe, os pesos da função de avaliação podem ser estimados por técnicas de aprendizado de máquina.

Decisões imperfeitas podem levar a erros



- Suponha que a busca parou na profundidade em que a as pretas tem vantagem de um cavalo e dois peões
- No próx. mov. As brancas capturam a rainha e ganham o jogo.

Decisões imperfeitas podem levar a erros

- Solução: um corte mais sofisticado:
  - busca quiescente: aplicar a função de avaliação somente em posições em que é improvável haver grandes mudanças de valores em estados futuros (posições quiescentes)

Decisões imperfeitas: efeito de horizonte

- Surge em movimentos inevitáveis do oponente que causam sérios danos;
  - busca com profundidade fixa protela estes movimentos “para além do horizonte da busca”

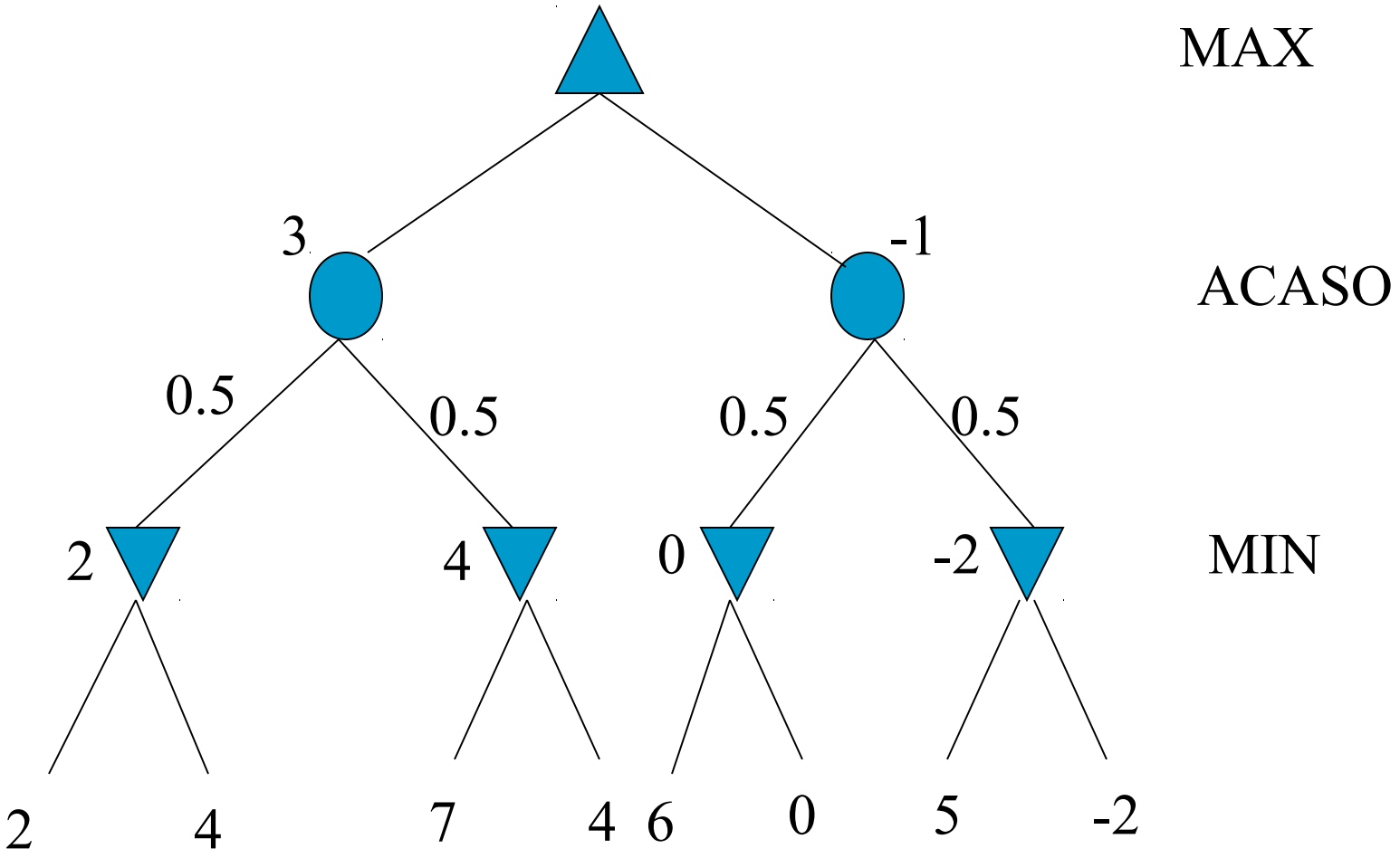
Jogos não determinísticos

- Elemento aleatório proveniente de jogo de dados, sorteio de cartas, etc.
- Não determinismo é inerente em domínios reais;
- o estudo de algoritmos para jogos com elemento aleatório é um passo em direção a métodos aplicados no mundo real.

Jogos não determinísticos

- Uma árvore de um jogo não determinístico deve incluir ***nós de acaso*** além de nós minimax
- Ramificações que levam a cada nó de acaso denotam “jogadas de dados possíveis” (a probabilidade de cada mudança de estado não determinística).

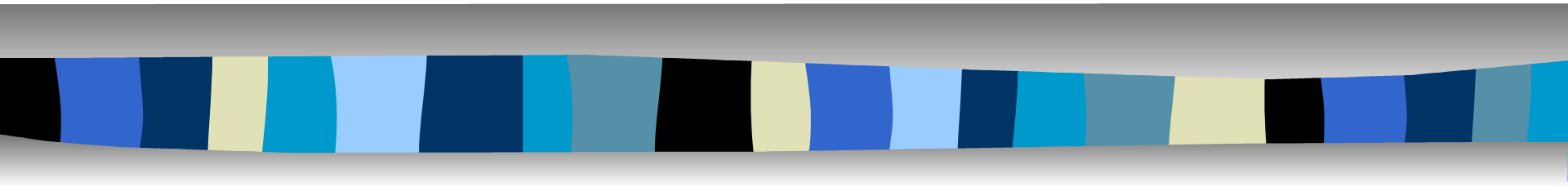
Jogos não determinísticos



Conclusão

- “games are to AI as grand prix racing is to automobile design”

Exercícios sobre busca

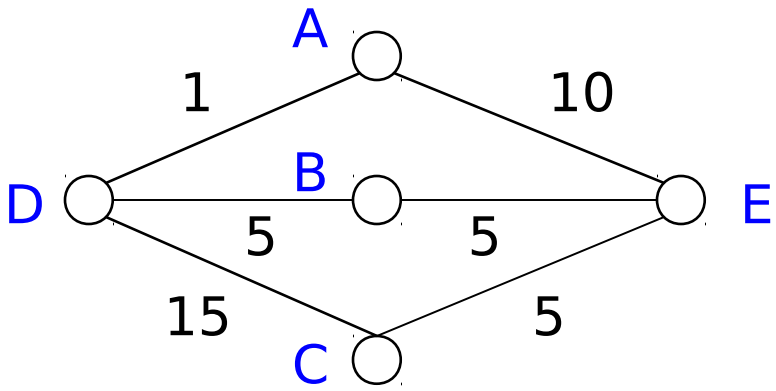


1. Busca Cega

- (Jarros) Dados uma bica d`agua, um jarro de capacidade 3 litros e um jarro de capacidade 4 litros (ambos vazios). Como obter 2 litros no jarro de 4?
  - a. Formalizar o problema e apresentar espaço de estados;
  - c. Desenhar busca em largura;
  - d. Desenhar busca em profundidade;
  - e. Desenhar busca com aprofundamento iterativo;
  - f. Como a busca bidirecional funcionaria?

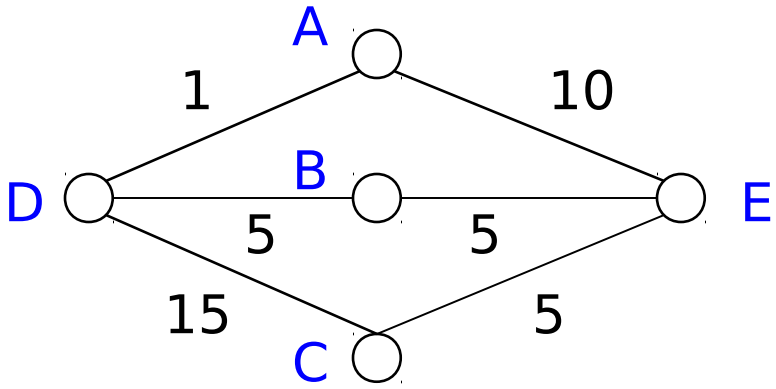
2. Busca Heurística

- “Uma caixeiro viajante possui uma lista de cidades, as quais ele deve vizitar exatamente uma vez. Há estradas diretas entre cada par de cidades da lista. O problema é encontrar o circuito mais curto que tem início e término em qualquer uma das cidades.”



2. Busca Heurística

- a. Defina uma heurística admissível para este problema;
- b. mostre como o algoritmo guloso funciona neste problema;
- c. mostre como A* funciona;



3. Busca competitiva

- (2,2-nim) Inicialmente, há 2 conjuntos de 2 fósforos. Em cada jogada um jogador remove qualquer númer de fósforos de exatamente uma pilha. O vencedor é aquele que remove o último fósforo do jogo.
  - Desenhe a árvore minimax deste jogo.
  - Há como executar poda α-β para 2,5-nim?