Inteligência Artificial

Luís A. Alexandre

UBI

Ano lectivo 2018-19

Conteúdo

Conteúdo

Pesquisa adversarial
- Jogos
- Decisões ótimas em jogos
- Poda alfa-beta
- Decisões imperfeitas em tempo real
- Jogos que incluem aleatoriedade

Leitura recomendada

Jogos

- O caso do ambiente com **agentes competitivos** leva ao aparecimento de problemas de **pesquisa adversarial** - os **jogos**.
- A teoria dos jogos vê os **ambientes multi-agente** como um jogo desde que o impacto de cada agente sobre os outros seja significativo, independentemente do ambiente ser **cooperativo** ou **competitivo**.
- Na IA os jogos são normalmente dum tipo muito específico: determinísticos, à vez, dois jogadores e soma nula, com informação perfeita.

Jogos

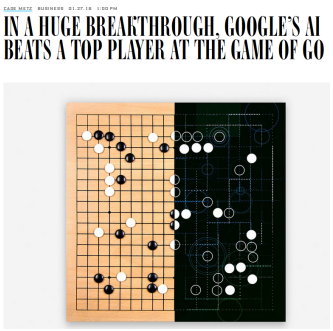
- Isto significa que:
  - os jogos se passam num ambiente determinístico;
  - existem dois agentes cujas ações devem ser executadas de forma alternada;
  - os valores da função de utilidade no fim do jogo são sempre valores iguais e de sinal oposto;
  - o ambiente é totalmente observável.
- P. ex., se um jogador ganha um jogo de xadrez (+1) ou outro perde (-1). É esta oposição entre as funções de utilidade dos jogadores que torna a situação adversarial.

Porquê estudar jogos em IA ?

- São construções abstratas, logo sem os detalhes que surgem nos problemas reais e que complicam as análises.
- O estado de um jogo é fácil de representar.
- Os agentes têm normalmente um número reduzido de ações que podem executar e cujas consequências estão definidas por regras.
- Os jogos foram assunto para a IA praticamente desde o seu começo.
- Atualmente as máquinas superam os humanos nas damas e no othello, já derrotaram campeões do mundo de xadrez e de gamão e são muito competitivas nos restantes jogos.
- Para termos uma noção do grau de dificuldade destes jogos, vejamos o caso do xadrez:
  - fator de ramificação de cerca de 35
  - um jogo chega a levar 50 jogadas por cada jogador
  - árvore de pesquisa com $35^{100} = 10^{154}$ nodos, embora "apenas" 10^{40} sejam distintos

Avanço da IA nos jogos

- Em 2009: "Atualmente as máquinas superam os humanos... A principal exceção é o jogo Go onde aparentemente ainda estão num nível básico."
- 2016:


IN A HUGE BREAKTHROUGH, GOOGLE'S AI BEATS A TOP PLAYER AT THE GAME OF GO


GOOGLE'S AI WINS FIFTH AND FINAL GAME AGAINST GO GENIUS LEE SEDOL

Mais detalhes.

Porquê estudar jogos em IA ?

- Os jogos, como a vida real, exigem que se tomem decisões mesmo sem poder calcular qual é a decisão ótima.
- A **ineficiência** nos jogos é normalmente penalizada de forma severa: uma má implementação do A* pode ter 50% da eficiência duma outra: num jogo como o xadrez com limite de tempo isso pode significar que essa implementação perca sempre.
- Dadas as questões que referimos antes, o estudo dos jogos em IA levou à descoberta de várias ideias de como aproveitar melhor o tempo em termos da resolução de problemas.

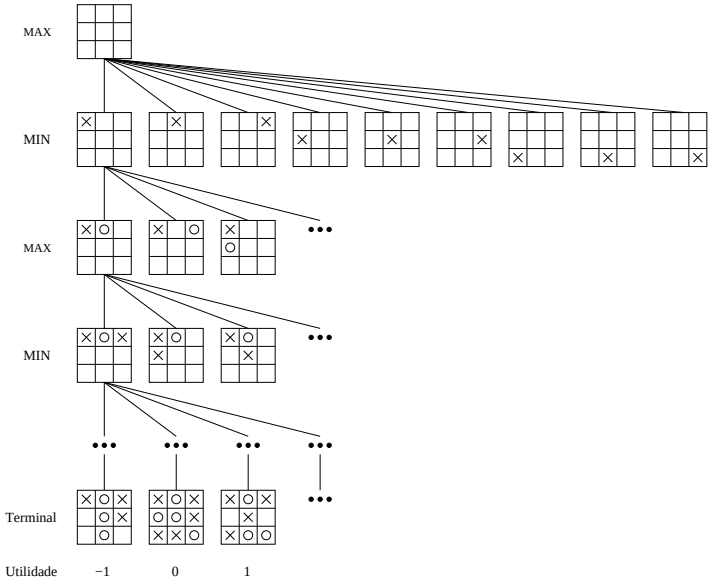
Conteúdo

Decisões ótimas em jogos

- Consideremos jogos com 2 jogadores a que chamaremos MAX e MIN.
- MAX joga primeiro e depois jogam à vez.
- Um jogo pode ser definido como um **problema de pesquisa** com os seguintes componentes:
  - um **estado inicial** que inclui as posições no tabuleiro e identifica o jogador que irá começar;
  - *jogador*(s) indica o jogador que joga no estado s;
  - *acoes*(s) devolve o conjunto de jogadas possíveis no estado s;
  - *resultado*(s, a) devolve o estado resultado de fazer a ação a quando o jogo está no estado s;
  - *estado_terminal*(s) que é verdadeiro quando s é um estado terminal (jogo termina) e falso caso contrário;
  - *utilidade*(s, j) (também chamada função objetivo) dá um valor numérico ao estado terminal s para o jogador j. No xadrez o resultado pode ser vitória, derrota ou empate com valores respetivos de +1, -1, e 0. O gamão por exemplo tem uma gama de valores possíveis maior (de +192 a -192).

Decisões ótimas em jogos

- A figura ao lado mostra parte da **árvore de jogo** do jogo do galo.
- O estado inicial e as jogadas legais definem esta árvore.
- A **árvore de pesquisa** é uma sub-árvore da árvore de jogo com os nodos consultados enquanto se tenta resolver o jogo.



Decisões ótimas em jogos

- Partindo do estado inicial o MAX tem 9 possibilidades colocar o seu marcador (X).
- O MIN coloca o seu (O) alternadamente até chegarmos a um nodo folha que corresponde a um **estado terminal**: um dos jogadores tem 3 em linha ou já não existem quadrados livres no tabuleiro.
- O número em cada folha indica o **valor de utilidade** do estado terminal para o MAX: valores altos são bons para o MAX e maus para o MIN.

Estratégias ótimas

- Num problema normal de pesquisa uma solução ótima seria uma sequência de ações que conduzem a um estado objetivo: um estado terminal que é uma vitória.
- Num jogo, MAX tem sempre de levar em consideração as ações do MIN.
- Assim, chamamos uma estratégia de **ótima** se levar a um resultado de MAX pelo menos tão bom quanto o alcançável se MIN fosse infalível.
- Veremos de seguida como achar a estratégia ótima, embora só seja possível fazê-lo (devido a custos computacionais) para jogos muito simples como o jogo do galo.

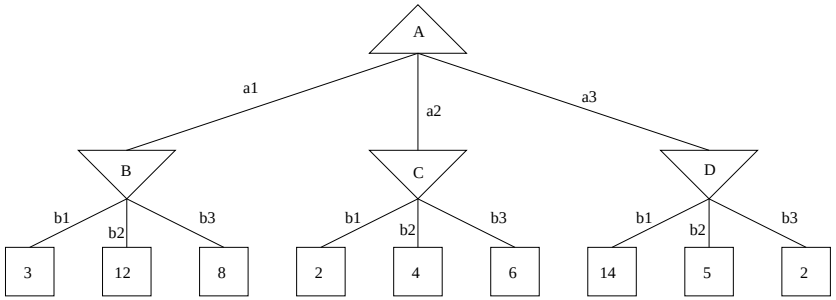
Estratégias ótimas

- Dada a árvore de jogo, a estratégia ótima pode ser obtida examinando o **valor minimax** de cada nodo, a que chamaremos $MINIMAX(s)$.
- Este valor corresponde à utilidade para MAX desse nodo assumindo que ambos os jogadores jogam de forma ótima a partir desse nodo até ao fim do jogo.
- Dada a escolha, MAX prefere um movimento para um nodo com valor minimax máximo, ao passo que MIN prefere um nodo com valor minimax mínimo.
- Assim vem

$$MINIMAX(s) = \begin{cases} utilidade(s, jogador(s)) & \text{se } s \text{ for estado term.} \\ \max_{a \in acoes(s)} MINIMAX(resultado(s,a)) & \text{se jogador}(s) == \text{MAX} \\ \min_{a \in acoes(s)} MINIMAX(resultado(s,a)) & \text{se jogador}(s) == \text{MIN} \end{cases}$$

Estratégias ótimas

- Mesmo para o jogo do galo não conseguimos facilmente desenhar a sua árvore de jogo logo vamos começar por considerar o seguinte jogo trivial: MAX pode fazer uma de 3 jogadas, a_1, a_2, a_3 e MIN pode responder com uma de 3 jogadas também b_1, b_2, b_3.
- A árvore de jogo é a seguinte, onde os triângulos representam nodos em que é a vez de MAX jogar e os triângulos invertidos os estados em que é a vez de MIN jogar. Os quadrados são os nodos terminais e têm o valor de utilidade para MAX.



Estratégias ótimas

- Vejamos como se aplica a definição do $MINIMAX$ no caso do jogo acima.
- Os estados terminais já contêm o seu valor de utilidade.
- O primeiro nodo MIN, chamado B, tem 3 nodos sucessores, com valores 3, 12 e 8, logo o seu valor minimax é ____.
- Os outros 2 nodos minimax (C e D) têm valor minimax ____.
- O nodo raiz é nodo MAX e dados os valores minimax dos seus sucessores, ele tem valor ____.

Estratégias ótimas

- Assim podemos identificar a decisão minimax na raiz: a ação a_1. É ótima pois conduz ao sucessor com maior valor minimax.
- Esta definição de ótimo para MAX assume que o MIN também vai agir de forma ótima: tenta sempre o pior resultado para MAX. E se o MIN não jogar de forma ótima ? Então o MAX vai ter ainda melhores resultados usando a estratégia minimax.

Algoritmo minimax

- O seguinte algoritmo acha a decisão minimax a partir do nodo actual.
- Usa uma implementação recursiva: começa por ir até às folhas da árvore onde acha a decisão usando a função *utilidade*(s, j). Conforme a recursividade se vai desfazendo, os valores de minimax de cada nodo vão sendo preenchidos.
- Este algoritmo usa PPP e se a maior profundidade da árvore for p e existirem b jogadas legais que a complexidade temporal é $O(b^p)$. A complexidade espacial é $O(bp)$ se gerarmos todos os nodos duma vez e $O(p)$ para um algoritmo que gera um sucessor de cada vez.
- Dada a complexidade temporal deste algoritmo, ele não é utilizável em jogos reais, mas vai servir de base para o estudo de outros algoritmos mais eficientes.

Algoritmo minimax

```
function MINIMAX_DECISÃO(estado) RETURNS ação
 return arg max_{a∈acoes(estado)} VALOR_MÍNIMO(resultado(estado,a))

function VALOR_MÁXIMO(estado) RETURNS valor de utilidade
 SE estado_terminal(estado) ENTÃO return utilidade(estado,jogador(estado))
 v← −∞
 PARA a IN acoes(estado)
        v← MAX(v,VALOR_MÍNIMO(resultado(estado,a)))
 return v

function VALOR_MÍNIMO(estado) RETURNS valor de utilidade
 SE estado_terminal(estado) ENTÃO return utilidade(estado,jogador(estado))
 v← +∞
 PARA a IN acoes(estado)
        v← MIN(v,VALOR_MÁXIMO(resultado(estado,a)))
 return v
```


Conteúdo

Poda alfa-beta

- O problema com a abordagem minimax é que o número de estados que devemos examinar cresce exponencialmente com o número de jogadas (vimos atrás que é $O(b^p)$).
- Não é possível eliminar o expoente mas podemos dividi-lo por 2, pois é possível achar a decisão minimax correta sem ser necessário olhar para todos os nodos da árvore.
- Podemos podar a árvore de forma a eliminar vários estados.
- A técnica de poda que iremos estudar chama-se **poda alfa-beta**.
- Esta técnica quando é aplicada à árvore minimax devolve a mesma decisão que o minimax normal, mas **elimina ramos da árvore que não podem influenciar a decisão**.

Poda alfa-beta

- Consideremos de novo o jogo simples que vimos atrás, apenas com uma jogada para cada jogador ($p = 2$).
- Se voltarmos a calcular os valores minimax com atenção podemos verificar que chegamos à decisão minimax sem nunca precisarmos de visitar dois dos nodos folha. Quais?

Poda alfa-beta

- A poda alfa-beta pode ser aplicada a árvores de qualquer profundidade e muitas vezes podam-se sub-árvores completas e não apenas nodos folha.
- O nome desta estratégia vem dos seguintes parâmetros:
  - α é o valor da melhor escolha (valor maior) até um dado momento para MAX
  - β é o valor da melhor escolha (valor menor) até um dado momento para MIN

Algoritmo da pesquisa alfa-beta

- A pesquisa alfa-beta vai atualizando os valores α e β enquanto percorre a árvore e poda os restantes ramos num nodo assim que o valor desse nodo for pior que o valor atual de α ou β, consoante seja a vez de jogar o MAX ou o MIN.

Algoritmo da pesquisa alfa-beta

function PESQUISA_ALFA_BETA(estado) RETURNS ação
 v ← VALOR_MÁXIMO(estado,$-\infty$,∞)
 return a $\in$ acoes(estado) com valor v

function VALOR_MÁXIMO(estado,α,β) RETURNS valor de utilidade
 SE estado_terminal(estado) ENTÃO return utilidade(estado,jogador(estado))
 v← $-\infty$
 PARA a IN acoes(estado)
 v← MAX(v,VALOR_MÍNIMO(s,α,β))
 SE v $\geq \beta$ ENTÃO return v
 $\alpha \leftarrow$ MAX(α,v)
 return v

function VALOR_MÍNIMO(estado,α,β) RETURNS valor de utilidade
 SE estado_terminal(estado) ENTÃO return utilidade(estado,jogador(estado))
 v← $+\infty$
 PARA a IN acoes(estado)
 v← MIN(v,VALOR_MÁXIMO(s,α,β))
 SE v $\leq \alpha$ ENTÃO return v
 $\beta \leftarrow$ MIN(β,v)
 return v

Conteúdo

Decisões imperfeitas em tempo real

- O algoritmo minimax gera todo o espaço de pesquisa, enquanto a poda alfa-beta permite eliminar grande parte desse espaço.
- No entanto, mesmo a poda alfa-beta precisa de atingir o nodo folha para uma parte do espaço de pesquisa.
- Quando as jogadas têm de ser feitas num tempo limitado pode não ser possível chegar até um nodo folha.
- Shannon sugeriu em 1950, num artigo sobre escrita de programas para jogar xadrez, que os programas deviam **terminar a pesquisa antes de chegarem aos nodos folha** e usarem uma função de avaliação heurística para estimar a utilidade do nodo.
- Isto traduz-se em duas alterações aos algoritmos minimax e poda alfa-beta:
  - substituição da função de utilidade por uma função de avaliação heurística, EVAL, que **estima** a utilidade do nodo;
  - substituir o teste de terminação por um **teste de corte** que decide quando se deve usar EVAL.

Heuristica-minimax

- Seguindo as ideias anteriores chegamos à seguinte heurística minimax para o estado s à profundidade d:

$$HMINIMAX(s,d) =$$

$$\begin{cases} EVAL(s) & \text{se } s \text{ for estado corte} \\ \max_{a \in acoes(s)} HMINIMAX(resultado(s,a), d+1) & \text{se jogador}(s) == \text{MAX} \\ \min_{a \in acoes(s)} HMINIMAX(resultado(s,a), d+1) & \text{se jogador}(s) == \text{MIN} \end{cases}$$

Funções de avaliação

- Uma função de avaliação devolve uma **estimativa** para a utilidade do jogo a partir duma dada posição, do mesmo modo que as funções heurísticas que vimos nas pesquisas devolviam uma estimativa da distância até ao objetivo.
- A ideia duma função de avaliação já era usada muito antes da proposta de Shannon por todos os jogadores que tentam estimar o valor da sua posição no jogo. Isto porque os humanos não conseguem fazer muita pesquisa.
- A função de avaliação deve:
  - ordenar os estados terminais como a função de utilidade verdadeira, senão poderia fazer jogadas sub-ótimas mesmo que conseguisse ter acesso a todos os estados do jogo.
  - ser rápida, pelo menos mais rápida que o minimax, senão não vale a pena usá-la.
  - ser fortemente correlacionada com a **probabilidade** de vitória, para estados não terminais.

Funções de avaliação

- O termo **probabilidade** surge aqui mesmo sem haver lançamento de dados porque ao interrompermos a pesquisa em nodos não terminais ficamos numa situação de **incerteza** face ao verdadeiro valor desses estados.
- A maioria das funções de avaliação calcula várias **características** do estado (ex.: no xadrez, uma característica poderia ser o número de peões que cada jogador tem ainda em jogo).
- As características permitem definir **classes de equivalência de estados**: estados que possuem as mesmas características são equivalentes, independentemente dos detalhes (a posição dos peões, p.ex.).
- Cada **classe de equivalência** contém alguns estados que irão dar vitórias, outros derrota e outros empates.

Funções de avaliação

- Não é possível saber à priori quais os estados que vão conduzir a cada desfecho sem se calcular a árvore até aos estados terminais, mas pode-se achar o seu **valor esperado**.
- Por exemplo, se numa dada classe de equivalência existir uma probabilidade de vitória de 72% (valor de utilidade =1), 20% de derrota (-1) e 8% de empate (0), então o valor esperado desta classe é $0.72 \times 1 + 0.20 \times (-1) + 0.08 \times 0 = 0.52$.
- As estimativas das probabilidades são obtidas após análise de um conjunto grande de jogos.

Funções de avaliação

- Exemplo: os livros básicos de xadrez dão alguns valores aproximados de cada peça: peão vale 1, cavalo e o bispo valem 3, torre 5 e rainha 9. Algumas características como a boa colocação dos peões ou a segurança do rei podem valer meio peão.
- O que se faz é adicionar os valores destas características para se ficar com uma estimativa do valor da posição.
- Uma vantagem de cerca de 1 peão dá boas hipóteses de vitória e se for de 3 peões dá vitória quase certa.
- A função de avaliação é deste tipo

$$EVAL(s) = \sum_{i=1}^{n} w_i f_i(s)$$

  onde w_i representa a importância da característica f_i.
- P. ex., ainda no xadrez, f_i poderia ser o número de peças de cada tipo ainda no tabuleiro e os w_i os valores respectivos de cada peça referidos acima.

Interromper a pesquisa

- O passo seguinte será decidir **quando é que se deve interromper a pesquisa**.
- A abordagem mais simples é definir uma profundidade máxima p.
- Esta profundidade é escolhida de forma a que nunca se exceda a quantidade de tempo disponível no jogo.
- Outra hipótese seria aplicar a profundidade iterativa, como estudámos atrás.
- Neste caso, quando o tempo disponível termina, o algoritmo devolve a jogada com maior valor da função de avaliação que tenha conseguido calcular até ao momento.

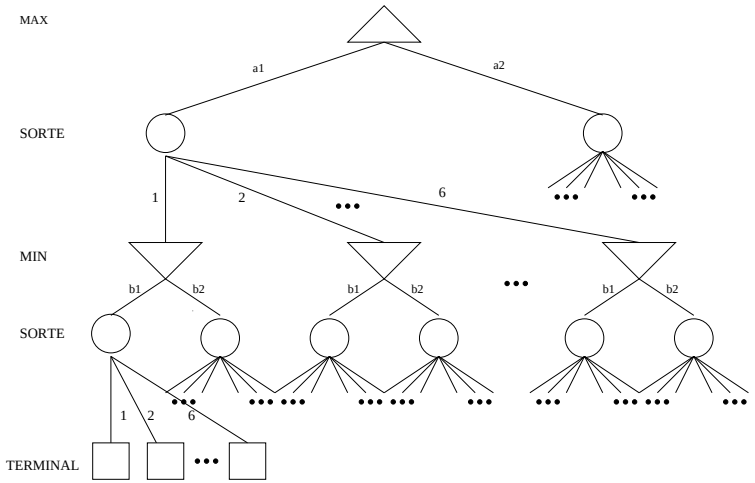
Conteúdo

Jogos que incluem aleatoriedade

- Muitos jogos tentam simular a imprevisibilidade da vida real através da introdução de aleatoriedade, como por exemplo, lançando um dado.
- Para lidarmos com este tipo de jogos, temos de incluir na árvore de pesquisa **nodos aleatórios**, além dos nodos MAX e MIN já vistos.

Jogos que incluem aleatoriedade

- Exemplo para um jogo em que cada jogador pode fazer uma de duas ações e lança um dado de seguida.



Jogos que incluem aleatoriedade

- Agora temos de saber como tomar decisões neste caso: queremos ainda escolher a jogada que leve à melhor posição.
- No entanto, as posições não têm valores minimax definidos: temos de calcular o **valor esperado**.
- A **esperança** é calculada sobre todos os possíveis valores aleatórios (p.ex., no caso de um dado, tomando em consideração que podemos obter valores entre 1 e 6).

Expectiminimax

- Deste modo obtemos uma generalização do valor minimax para jogos com nodos aleatórios, a que chamaremos **expectiminimax** e que definimos como:

$$EXPECTMINIMAX(s) =$$

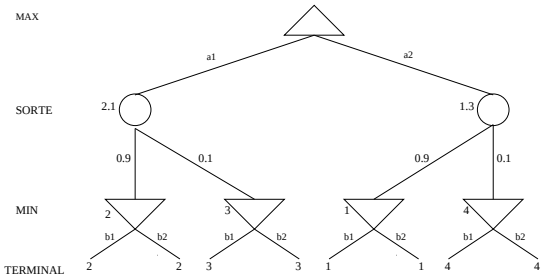
$$\begin{cases} utilidade(s, jogador(s)) & \text{se } s \text{ for estado term.} \\ \max_{a \in acoes(s)} EXPECTIMINIMAX(resultado(s,a)) & \text{se jogador}(s) == \text{MAX} \\ \min_{a \in acoes(s)} EXPECTIMINIMAX(resultado(s,a)) & \text{se jogador}(s) == \text{MIN} \\ \sum_r P(r) EXPECTIMINIMAX(resultado(s,r)) & \text{se jogador}(s) == \text{SORTE} \end{cases}$$

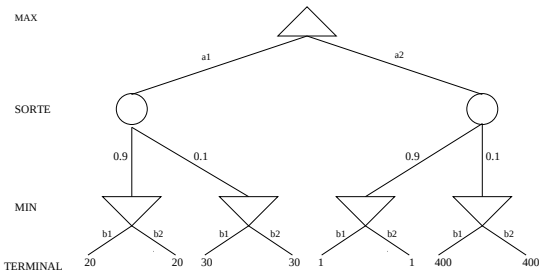
onde r é uma ação aleatória e $P(r)$ é a probabilidade de ocorrência dessa ação.

Avaliação da posição em jogos com nodos aleatórios

- Como fizemos com o minimax, queremos cortar a pesquisa e usar uma função de avaliação.
- O problema é que a função de avaliação para um jogo com aleatoriedade é diferente das dos problemas sem aleatoriedade.
- Exemplo: na figura da direita a melhor jogada para MAX seria a_1.
- Mas se modificarmos a função de avaliação para alterar a escala dos valores mesmo sem alterar a sua ordem, a decisão ótima muda (neste caso a transformação é **não linear**).
- Recalcular os valores expectiminimax para a figura da direita e encontrar a decisão a tomar para os novos valores.

Avaliação da posição em jogos com nodos aleatórios





Avaliação da posição em jogos com nodos aleatórios

- Assim, se queremos uma função de avaliação que não altere os resultados face à pesquisa feita até chegarmos aos nodos, termos de escolher uma **transformação linear positiva** da utilidade.
- Um exemplo de uma função de avaliação não linear para o xadrez?

Leitura recomendada

- Russell e Norvig, cap. 5.