Universidade Federal de Minas Gerais

DCC642 - Introdução à Inteligência Artificial (2025/2)
TP2: Busca Competitiva

Raphael Henrique Braga Leivas - 2020028101

1 Introdução

Neste trabalho, algoritmos de busca competitiva são implementados em Python para competirem em um jogo de Ligue-4. Diferentes algoritmos são implementados e sua performance é comparada experimentalmente com base em diferentes métricas.

2 Objetivos

Os objetivos principais do trabalho são:

- Implementar em Python os algoritmos Minimax, Minimax com poda Alfa-Beta, Iterative Deepening com uma função heurística de avaliação do estado atual do tabuleiro;
- Comparar as performances dos algoritmos com base nas seguintes métricas: taxa de vitória, tempo médio por jogada, média de estados visitados

3 Metodologia

3.1 Heurística de Avaliação

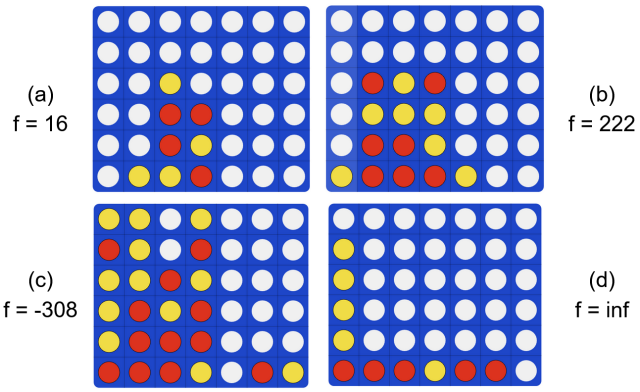
O primeiro passo é definir a heurística de avaliação. Definimos uma heurística da seguinte forma:

- Para cada sequência de 4 posições na horizontal, vertical ou diagonal, adicionamos um valor em uma ordem de grandeza: sequência de duas peças soma-se 10, sequência de três peças soma-se 100, quatro, 1000.
- O valor atribuído varia no sentido positivo indicando que o estado é favorável ao jogar 2, e no sentido negativo para o jogador 1. Assim, o jogador P1 assume o papel de minimizar e o P2 de maximizar.
- As células do centro são mais importantes no jogo, e portanto são multiplicadas por um fator 6.

Dessa forma, a heurística avalia os tabuleiros como mostra a Figura 1.

Na Figura 1 (a), o jogo parece bem empatado, com ambos jogadores sem ameaças iminentes e com controle central disputado. Assim, a avaliação é próxima de zero: $f(n) = 16$. Em (b), o jogador amarelo tem sequências de 3 peças sem bloqueio e bom controle central, de modo que $f(n) = 222$, um valor positivo elevado. Em (c) temos o contrário de (b), logo a heurística é um valor elevado negativo: $f(n) = -308$. Por fim, no caso de vitória de um jogador como ocorre em (d), a função retorna $\pm\infty$.

Figura 1: Exemplos de avaliação do estado atual do jogo com a heurística definida.



3.2 Agente Minimax

O primeiro agente implementado é o Minimax. O jogador vermelho assume o papel de minimizar e o amarelo de maximizar. O Algoritmo 1 mostra o pseudocódigo para o Minimax usado no projeto. Os experimentos comparam a performance do minimax para diferentes valores de profundidade $depth$, bem como o tempo de execução.

Algorithm 1: Minimax.

```
1:  function MINIMAX(state, depth, maxPlayer)
2:      if depth = 0 or ISTERMINAL(state) then
3:          return EVALUATE(state)
4:      end if
5:      if maxPlayer then
6:          maxEval ← −∞
7:          for all child in SUCCESSORS(state) do
8:              eval ← MINIMAX(child, depth − 1, false)
9:              maxEval ← max(maxEval, eval)
10:         end for
11:         return maxEval
12:     else
13:         minEval ← +∞
14:         for all child in SUCCESSORS(state) do
15:             eval ← MINIMAX(child, depth − 1, true)
16:             minEval ← min(minEval, eval)
17:         end for
18:         return minEval
19:     end if
20: end function
```

3.3 Agente Minimax com Poda Alfa Beta

Para adicionar a Poda Alfa Beta no Algoritmo 1, basta adicionar a seguinte condicional no algoritmo dentro do loop dos sucessores:

Algorithm 2: Minimax com Poda Alfa Beta.

```
1: if α ≥ β then
2:     break
3: end if
```

Note que α e β agora são argumentos passados para a função Minimax. O número de nós expandidos com a poda é comparado com o Minimax sem pode, de modo a verificar experimentalmente o impacto da poda na execução do algoritmo.

3.4 Iterative Deepening

COMPLETAR DEPOIS

3.5 Experimentos

Os seguintes experimentos serão realizados:

- Minimax vs Aleatório
- Alfa-Beta vs Minimax (sem poda)
- Iterative Deepening vs Alfa-Beta
- IA do Aluno vs Jogador Humano

Os experimentos são realizados com o seguinte procedimento:

1. Realiza 3 jogos entre o Minimax e o oponente para cada nível de profundidade;
2. Salva os tempos por jogada e número de nós expandidos em cada jogada em um csv;
3. Analisa os dados a posteriori com `matplotlib` e extrai as conclusões.

4 Resultados

4.1 Minimax vs Aleatório

A Figura 2 mostra os tempos por lance para diferentes profundidades do Minimax, e A Figura 3 número de nós expandidos com diferentes profundidades configuradas. Como esperado, o algoritmo gasta mais tempo e expande mais nós para profundidades maiores.

Figura 2: Histograma de tempos gasto por lance para o Minimax para diferentes profundidades.

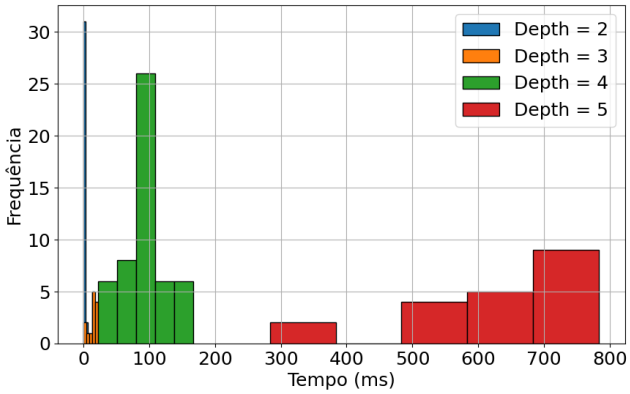
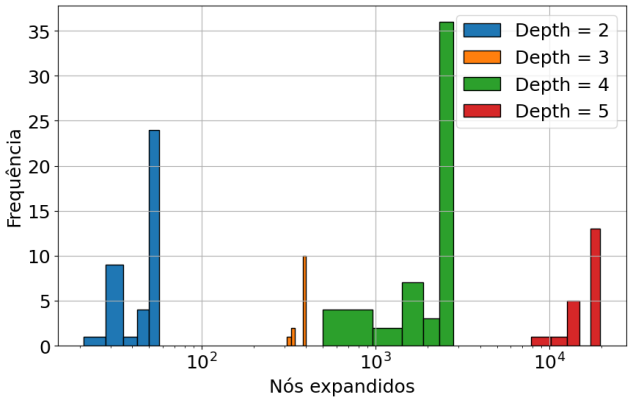


Figura 3: Histograma de nós expandidos por lance para o Minimax para diferentes profundidades.



Podemos tomar a média e o desvio padrão para cada uma das medições acima, e junto com a taxa de vitórias para cada profundidade, obtemos os resultados da Tabela 1.

4.2 Alfa-beta vs Minimax

5 Conclusão

COMPLETAR DEPOIS

Referências

Tabela 1: Resultados finais do experimento Minimax vs Aleatório.

Depth	Vitórias (%)	Tempo (ms)	Nós
2	0	3.6 ± 4.9	48.2 ± 12
3	100	14.1 ± 7.4	384 ± 30
4	100	92.5 ± 33.2	2357 ± 695
5	100	618 ± 130	16769 ± 2354