U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Exactly One Mazes

Inteligência Artificial

Grupo 13_1C:

- André Flores - up201907001
- Diogo Faria - up201907014
- Tiago Rodrigues - up201906807

Especificação

Encontrar um caminho num tabuleiro entre o Start (canto inferior esquerdo) e o Finish (canto superior direito), em que tem que passar por um e apenas um quadrado de cada peça com formato de L.

Consideramos que cada tabuleiro tem exatamente 1 solução, sendo que a solução ótima para este problema de pesquisa será a que apresente o resultado mais rapidamente, tal parâmetro influenciado tanto pelo algoritmo escolhido, como pela heurística, quando aplicável.

Tal implementação foi feita em Python 3, utilizando Visual Studio Code.

A interface é em linha de comandos, é requisitado ao utilizador o método de pesquisa, heurística, caso necessária, e o tabuleiro cuja solução quer encontrar. O utilizador também tem a opção de sair do programa. Para cada solução é apresentada a posição final, o tabuleiro com a solução, e o caminho encontrado.

Heurísticas e Função Objetivo

Foram utilizadas 3 heurísticas:

- A distância Manhattan desde a posição corrente até à posição final (canto superior direito do tabuleiro);
- O número de L's que faltam visitar;
- A subtração da distância Manhattan (1ª heurística) e do número de L's que já se visitaram.

Na função objetivo verificam-se 3 condições:

- Já se visitaram todos os L's:
  - len(state.lVisit) == 0;
- A posição corrente está na posição final:
  - state.pos[0] == 0 and state.pos[1] == (len(state.board) - 1)
- Na posição corrente/final tem um 1:
  - state.board[state.pos[0]][state.pos[1]] == 1

Operadores

Nome	Pré-condições	Pós-Condições	Custo
Down	R < Size-1 && Board[R+1][C] == 0	Pos = (R+1, C) e Board[R+1][C] = 1	1
DownL	R < Size-1 && Board[R+1][C] ∈ LVisit	Pos = (R+1, C), LVisit.remove(Board[R+1][C]) e Board[R+1][C] = 1	1
Up	R > 0 && Board[R-1][C] == 0	Pos = (R-1, C) e Board[R-1][C] = 1	1
UpL	R > 0 && Board[R-1][C] ∈ LVisit	Pos = (R-1, C), LVisit.remove(Board[R-1][C]) e Board[R-1][C] = 1	1
Left	C > 0 && Board[R][C-1] == 0	Pos = (R, C-1) e Board[R][C-1] = 1	1
LeftL	C > 0 && Board[R][C-1] ∈ LVisit	Pos = (R, C-1), LVisit.remove(Board[R][C-1]) e Board[R][C-1] = 1	1
Right	C < Size-1 && Board[R][C+1] == 0	Pos = (R, C+1) e Board[R][C+1] = 1	1
RightL	C < Size-1 && Board[R][C+1] ∈ LVisit	Pos = (R, C+1), LVisit.remove(Board[R][C+1]) e Board[R][C+1] = 1	1

Uninformed Search Methods

Nós implementámos **Breadth-First Search**, **Depth-First Search**, **Iterative Deepening** e **Uniform Cost**.

Em termos de **Breadth-First Search** e **Depth-First Search**, implementámos o algoritmo tal como foi dado na aula, utilizando uma lista com os estados já visitados de forma a eliminar visitas repetidas.

Quanto ao **Iterative Deepening** também se usou uma lista com os estados já visitados, com uma alteração em que se podiam visitar estados já visitados desde que tivessem uma profundidade menor ao igual visitado anteriormente.

O **Uniform Cost** foi implementado para a utilização de uma função de custo constante, visto que todos os operadores têm custo 1, e, por isso, comportar-se como um algoritmo **Breadth-First Search**.

Heuristic Search Methods

Implementámos **Greedy Search** e **A***.

Tanto **Greedy Search** e **A*** foram implementados com a possibilidade de passar as diferentes heurísticas nos argumentos das funções, sendo que na **A*** pode-se também passar diferentes funções de custo constante, visto que todos os operadores têm custo constante de 1.

Resultados

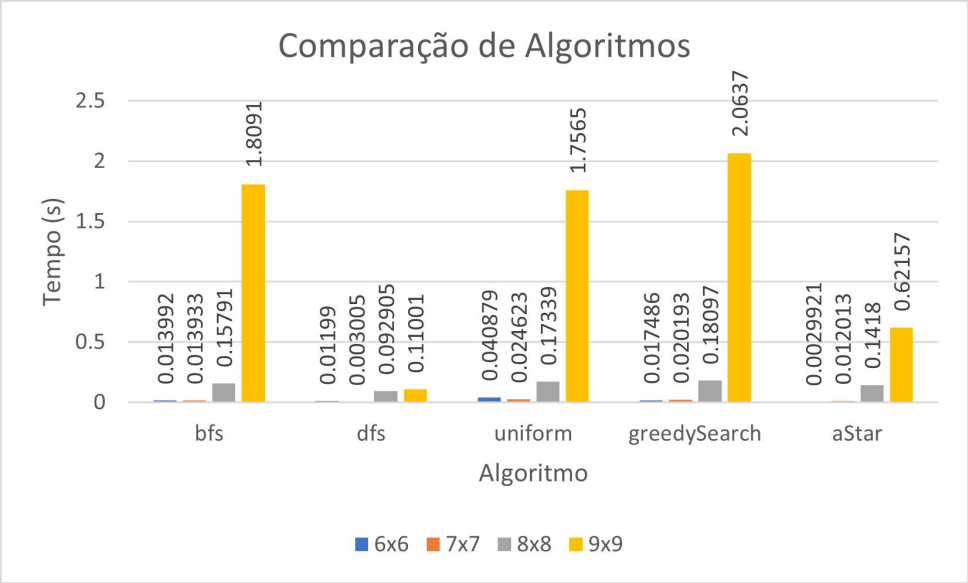

Fig. 1 - Comparação dos tempos de solução para cada algoritmo e para tabelas de tamanhos diferentes. As soluções de Iterative Deepening não se encontram representadas devido a sua relativa lentidão que tornava o gráfico ilegível.

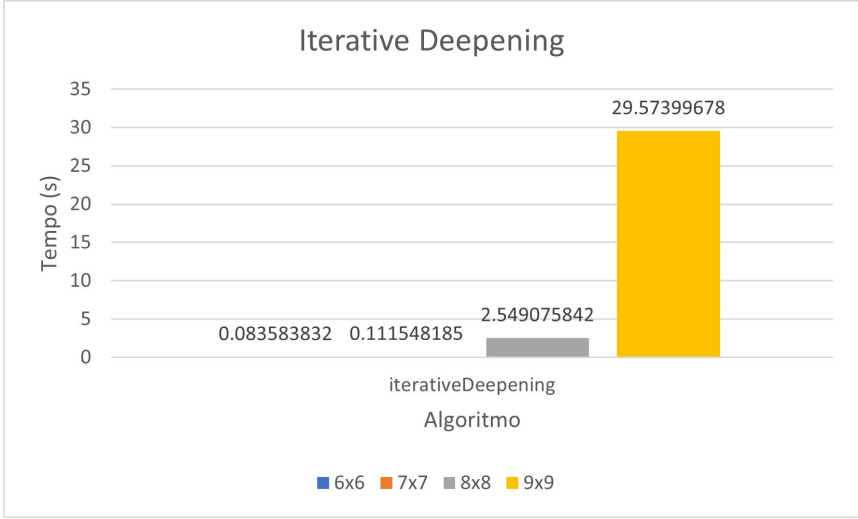

Fig. 2 - Comparação dos tempos de solução do algoritmo Iterative Deepening.

Resultados

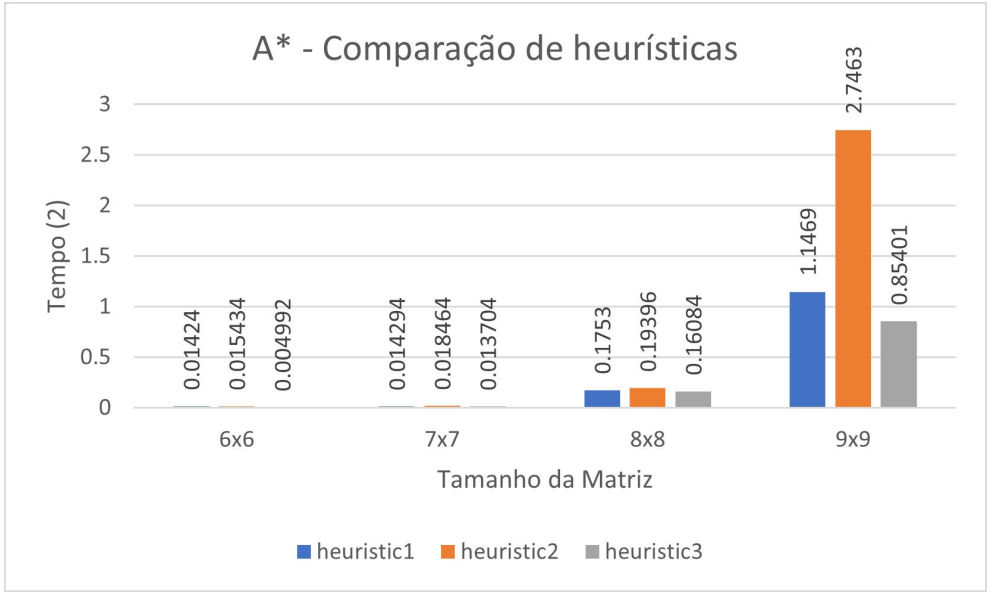

Fig. 3 - Comparação de heurísticas no algoritmo A*, agrupadas pelo tamanho de tabelas.

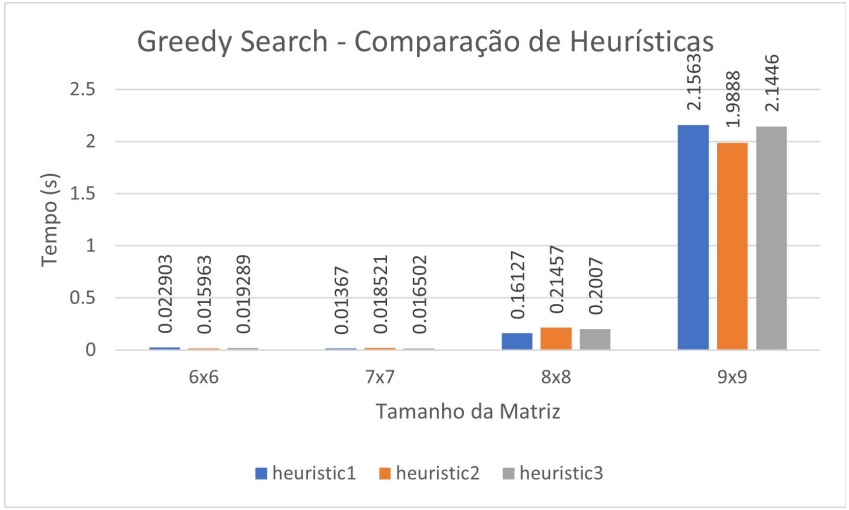

Fig. 4 - Comparação de heurísticas no algoritmo Greedy, agrupadas pelo tamanho de tabelas.

Conclusões

Este projeto permitiu-nos compreender melhor a utilização dos algoritmos de pesquisa implementados, sendo que todos os pedidos foram implementados corretamente, cumprindo os objetivos deste projeto.

Em relação aos algoritmos não informados, verificou-se que o Iterative Deepening foi o mais lento por uma grande margem, sendo que as suas vantagens comparando com o DFS não são visíveis devido à existência de apenas uma solução. O algoritmo BFS que o Uniform Cost, mesmo que se comportem da mesma forma com um tempo acrescido devido à ordenação de possíveis opções em função do custo, que, neste problema, vai ser constante para todas as operações.

Dos métodos heurísticos verificámos a maior eficiência do A* em relação ao Greedy, devido a ter em conta o custo de chegar a um nó da árvore de pesquisa. Quanto à heurística, verificou-se que quando oferecemos mais informação a heurística torna o algoritmo mais eficiente, logo a heurística que utiliza tanto o número de Ls visitados quanto a distância de Manhattan torna-se melhor que as duas separadamente.

Por fim, verificou-se que o algoritmo DFS foi o mais eficiente entre todos, pois, na nossa opinião, como os tabuleiros apresentam apenas uma solução as vantagens dos algoritmos heurísticos (que se manifestam em arranjar um caminho mais próximo do ótimo) tornam-se irrelevantes e o *overhead* adicional de calcular os valores das heurísticas apenas os abranda. No caso do DFS o facto de apenas haver uma solução resulta no rápido corte de quaisquer ramos de árvore inefetivos em que, caso contrário, poderia perder muito tempo.

Referências

- https://erich-friedman.github.io/puzzle/exactly1/
- Material teórico e prático da cadeira