Resolução da Atividade 2a - Resolução de Problemas de Pesquisa

Nuno Lopes (201605337)
Faculdade de Engenharia da
Universidade do Porto
Porto, Portugal
up201605337@fe.up.pt

***Resumo* - Esta resolução foi realizada no âmbito de uma atividade avaliada da cadeira IART e teve como objetivo a aplicação de vários métodos de pesquisa na resolução de vários puzzles do problema N-puzzle. O código fonte desta resolução foi escrito na linguagem python.**

Keywords: Inteligência Artificial, Pesquisa, Algoritmo A*, N-puzzle, Pathfinding, Procura em Largura, Pesquisa Gulosa

I. Formulação do Problema

Representação do Estado: Matriz NxN em que cada elemento pertence a {0, ..., N-1}, sendo que não existem elementos iguais dentro da mesma matriz.

Estado Inicial: O estado inicial depende do puzzle em questão. Consiste numa distribuição de diferentes números sem que estes se encontrem ordenados.

Estado Objetivo: Os números têm de estar ordenados de forma crescente entre 1 e (NxN)-1, sendo que o número 0 tem de ficar na última posição da matriz.

```
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15  0
```

Ilustração 1 - Estado objetivo do 16-puzzle

Operadores:

Os operadores que estão aqui apresentados são os movimentos para o número 0.

- ***left(pos)***:

  Pré-Condição: tem de haver pelo menos um número à esquerda do número 0.

  Efeito: o número 0 troca a sua posição com o número que está imediatamente à sua esquerda.

  Custo: 1.

- ***right(pos)***:

  Pré-Condição: tem de haver pelo menos um número à direita do número 0.

  Efeito: o número 0 troca a sua posição com o número que está imediatamente à sua direita.

  Custo: 1.

- ***up(pos)***:

  Pré-Condição: tem de haver pelo menos um número por cima do número 0.

  Efeito: o número 0 troca a sua posição com o número que está imediatamente por cima dele.

  Custo: 1.

- ***down(pos)***:

  Pré-Condição: tem de haver pelo menos um número por baixo do número 0.

  Efeito: o número 0 troca a sua posição com o número que está imediatamente por baixo dele.

  Custo: 1.

Custo da solução: Cada movimento do número 0 custa 1, logo o custo da solução é o número total de movimentos do número 0.

II. Algoritmos de Pesquisa

Nesta resolução foram implementados os algoritmos de pesquisa pedidos, sendo eles: pesquisa em largura; pesquisa gulosa; Algoritmo A*.

A implementação dos algoritmos foi muito semelhante à implementação realizada no primeiro trabalho prático.

III. Experiências e Resultados

Foram realizados vários testes de forma a recolher informação sobre o comportamento dos métodos de pesquisa utilizados, que podem ser consultados no ficheiro n-puzzle.py. Estes foram realizados para 4 puzzles sugeridos no guião, alterando entre duas heurísticas (também sugeridas) se fosse o caso.

Para correr estas experiências basta fazer ***pip3 install numpy*** e ***python3 n-puzzle.py*** num terminal, dentro da pasta **src**. Para visualizar as jogadas visualmente com a representação do puzzle basta descomentar a função ***print_puzzle_moves*** para cada teste.

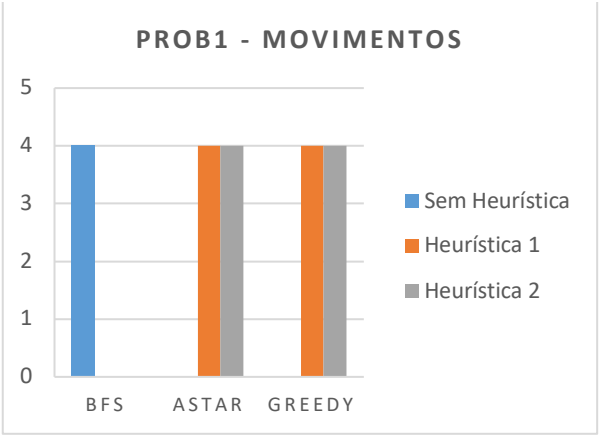
Para além das tabelas com os dados recolhidos também foram criados gráficos para ilustrar melhor as diferenças entre os algoritmos.

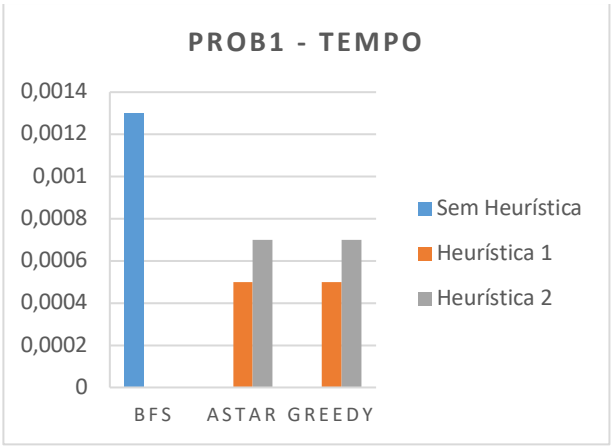
No caso do prob1 os resultados encontram-se na tabela e nos gráficos seguintes:

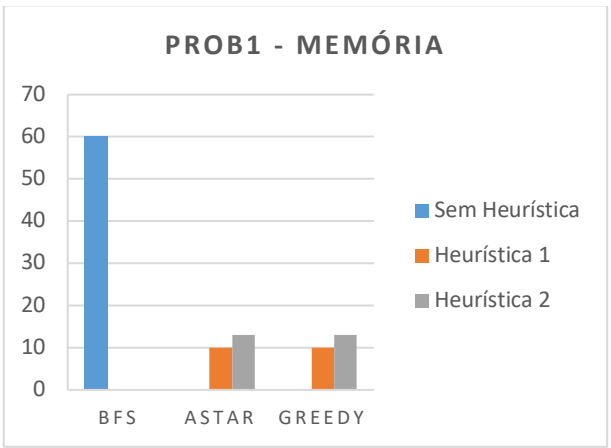
	Prob1	
Pesquisa em Largura	*Movimentos: 4* *Tempo: 0,0013 seg* *Memória: 60*	
Algoritmo A*	**Heurística 1**	**Heurística 2**
	Movimentos: 4 *Tempo*: 0.0005 seg *Memória*: 10	*Movimentos*: 4 *Tempo*: 0.0007 seg *Memória*: 13
Pesquisa Gulosa	**Heurística 1**	**Heurística 2**
	Movimentos: 4 *Tempo*: 0.0005 seg *Memória*: 10	*Movimentos*: 4 *Tempo*: 0.0007 seg *Memória*: 13

Podemos verificar que ambos os algoritmos obtiveram a solução ótima, porém já se notam algumas diferenças em termos de performance entre o algoritmo de pesquisa em largura e os outros dois.

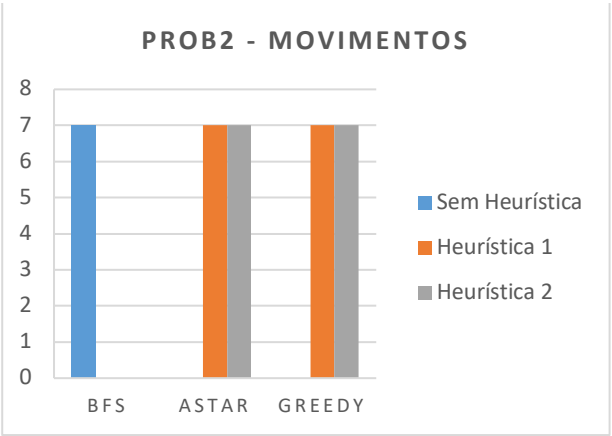
A análise do prob2 encontra-se na tabela e nos gráficos seguintes:

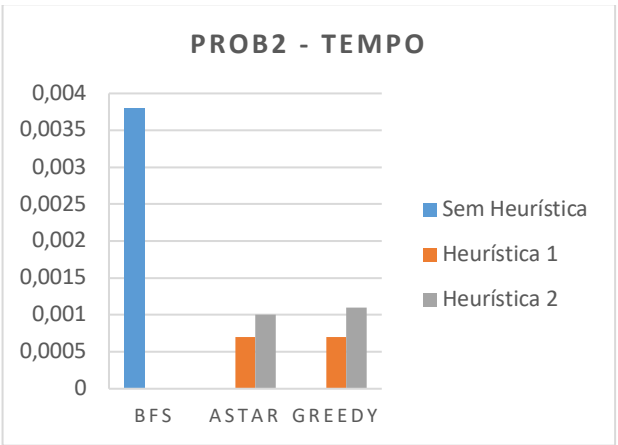


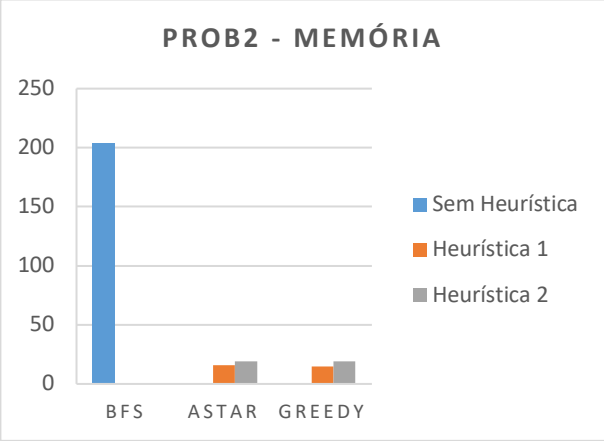


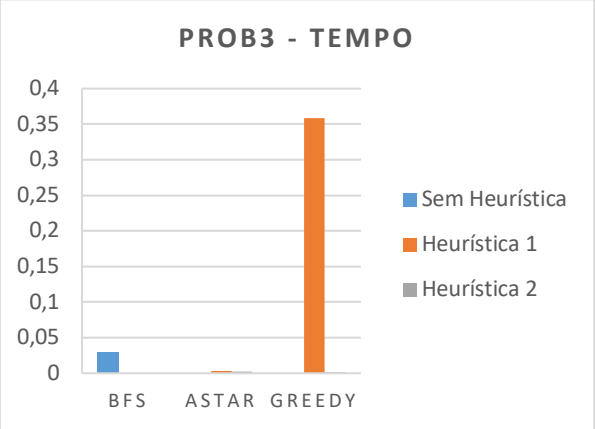


	Prob2	
Pesquisa em Largura	*Movimentos: 7* *Tempo: 0,0038 seg* *Memória: 204*	
Algoritmo A*	**Heurística 1**	**Heurística 2**
	Movimentos: 7 *Tempo*: 0.0007 seg *Memória*: 16	*Movimentos*: 7 *Tempo*: 0.0010 seg *Memória*: 19
Pesquisa Gulosa	**Heurística 1**	**Heurística 2**
	Movimentos: 7 *Tempo*: 0.0007 seg *Memória*: 15	*Movimentos*: 7 *Tempo*: 0.0011 seg *Memória*: 19





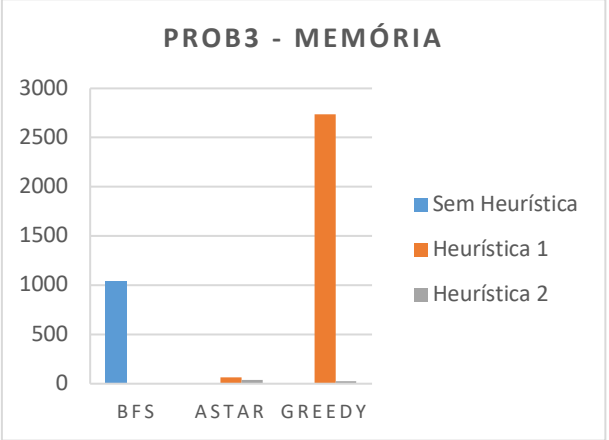




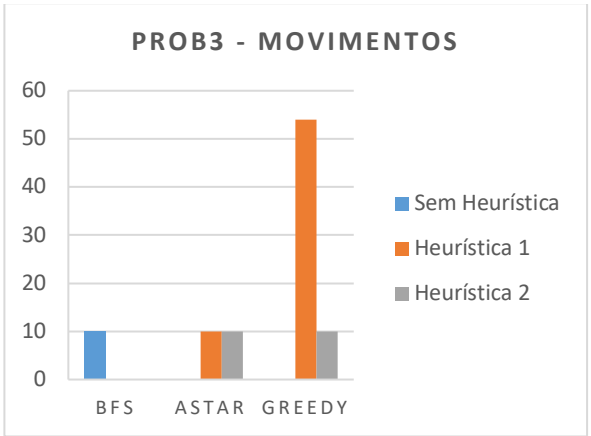
Mais uma vez os algoritmos obtiveram a solução ótima para o problema apesar de se notar uma diferença mais acentuada no tempo de execução e número de nós explorados por parte do algoritmo de pesquisa em largura.

No caso do prob3 a tabela de análise e os gráficos encontram-se de seguida:

	Prob3	
Pesquisa em Largura	*Movimentos: 10* *Tempo: 0,0292 seg* *Memória: 1042*	
Algoritmo A*	**Heurística 1**	**Heurística 2**
	Movimentos: 10 *Tempo*: 0.0029 seg *Memória*: 64	*Movimentos*: 10 *Tempo*: 0.0021 seg *Memória*: 36
Pesquisa Gulosa	**Heurística 1**	**Heurística 2**
	Movimentos: 54 *Tempo*: 0.3585 seg *Memória*: 2734	*Movimentos*: 10 *Tempo*: 0.0016 seg *Memória*: 28

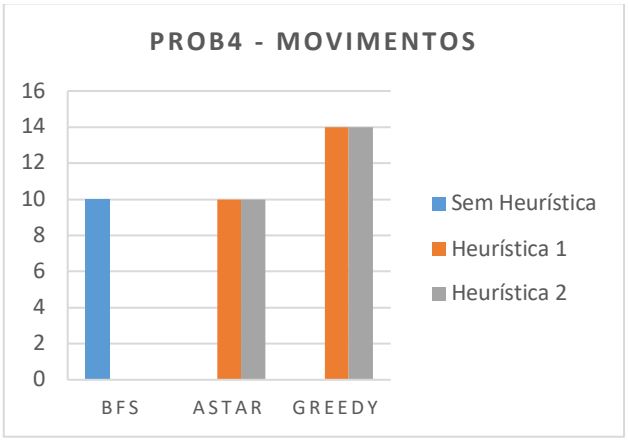


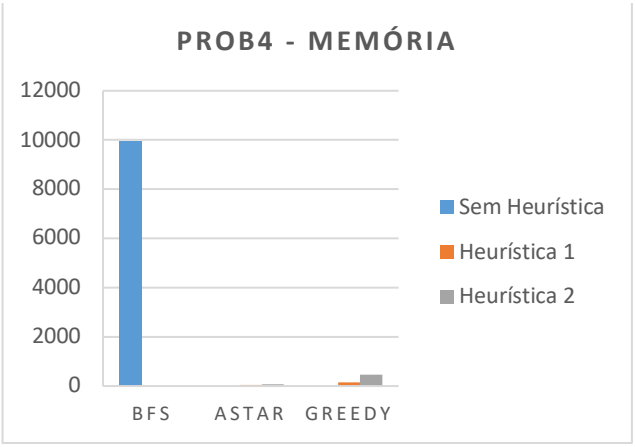
Nestes dados obtidos podemos verificar o porquê do algoritmo de pesquisa gulosa nem sempre pode ser fiável. Para a heurística 1 o algoritmo de pesquisa gulosa obtém um resultado muito pior em relação aos outros.

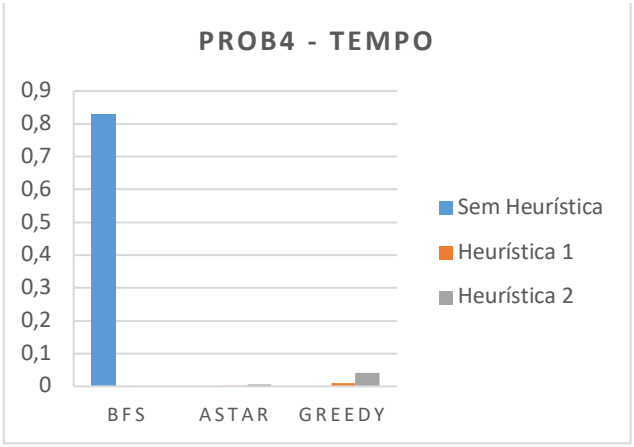


Os dados analisados do prob4 encontram-se de seguida:

	Prob4	
Pesquisa em Largura	*Movimentos: 10* *Tempo: 0,8284 seg* *Memória: 9934*	
Algoritmo A*	**Heurística 1**	**Heurística 2**
	Movimentos: 10 *Tempo*: 0.0029 seg *Memória*: 46	*Movimentos*: 10 *Tempo*: 0.0056 seg *Memória*: 75
Pesquisa Gulosa	**Heurística 1**	**Heurística 2**
	Movimentos: 14 *Tempo*: 0.0094 seg *Memória*: 156	*Movimentos*: 14 *Tempo*: 0.0419 seg *Memória*: 455







Com estes dados todos podemos verificar que o algoritmo A* é o que obtém melhores resultados em termos de custo, tempo de execução e nós explorados. Conclui-se então que o algoritmo A* é o mais eficiente.