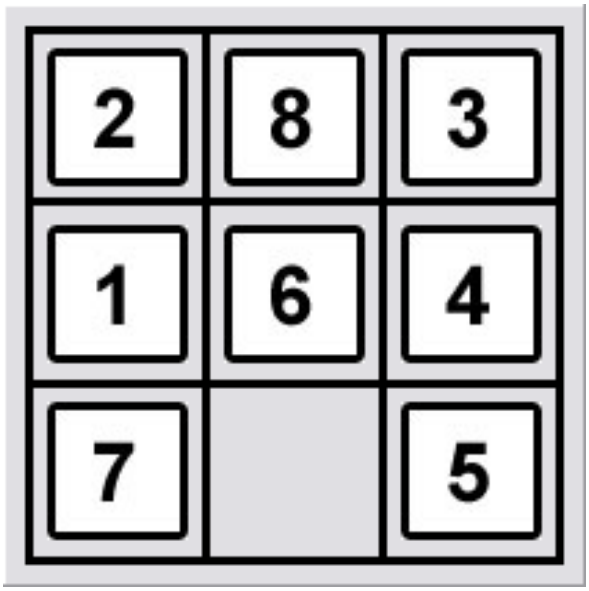
U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO



Inteligência Artificial
Março 2016

PROBLEMA DE PROCURA

"JOGO DOS 8"

Ricardo Dias Azevedo | up201403133
Rodrigo Flamínio Ribeiro | up201407863

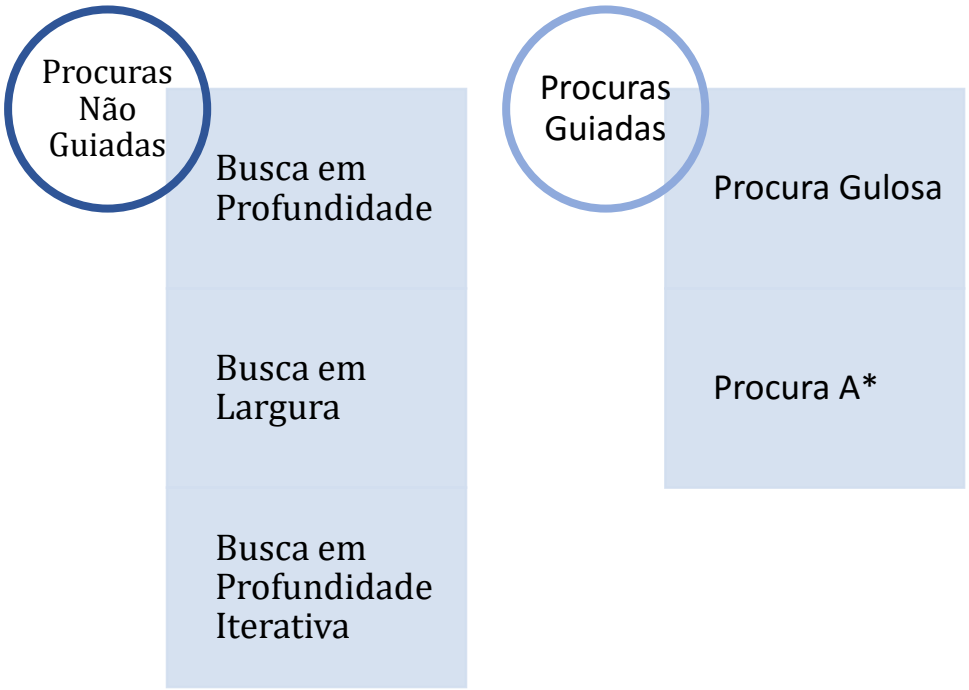
Índice

Introdução

No presente trabalho foi-nos pedido que, através de uma linguagem de programação, implementássemos várias estratégias de pesquisa na tentativa de compreender como cada uma se comporta no sentido da resolução de todas as possíveis configurações do *Jogo dos 8*. O referido jogo é definido por uma matriz de 3x3 e composto por oito células numeradas e uma em branco, sendo que o seu objetivo consiste em, partindo de uma configuração de células baralhadas, alcançar um padrão final determinado por uma sequência específica de algarismos.

Uma vez introduzido o problema que nos foi atribuído, podemos admitir que, de uma forma geral, um problema de procura pode ser definido como um conjunto de estados, entre os quais se destacam um estado inicial e outro final e uma função sucessor (capaz de gerar as possíveis ações a tomar – subestados – em prol do objetivo). No seu conjunto, o estado inicial e a função sucessor constituem a representação do espaço de estados, ou seja, o conjunto de todos os subestados que podem ser alcançados, partindo da posição inicial. O problema compreende ainda o "teste de objetivo", que nos permite determinar se o subestado adquirido é o objetivo, e o "custo do caminho", função que atribui um custo numérico a cada caminho [1] e que é utilizada em algoritmos mais eficientes, tais como o da procura guiada.

No sentido de desenvolver uma resolução para os diversos problemas de procura, deparamo-nos com vários métodos:



Estratégias de Procura

Como referido anteriormente, são diversas as estratégias de procura para a resolução de problemas.

I. Procura Não Guiada

Também definida como procura "cega", remete para um método em que não existe qualquer informação adicional sobre os estados da procura. Deste modo, esta estratégia apenas permite distinguir um estado que cumpre o objetivo, daquele que se desvia do estado final desejado [1]. Dentro deste tipo de procura encontramos vários modelos de busca:

i. Busca em Profundidade (DFS – *Depth-First Search*) – Nesta forma de busca verificamos que há progressão ao longo da expansão que se inicia no primeiro nó filho da árvore de busca. Assim, esta explora de forma cada vez mais profunda a "descendência" desse nó, podendo deparar-se com duas possibilidades: ou encontra o estado final objetivado, ou um nó que não apresenta sucessão. Quando tal acontece, a busca retrocede, passando ao segundo nó filho do nó raiz que ainda não foi explorado (figura 1). Para a realização de todo este procedimento, este método recorre ao princípio LIFO, que utiliza uma pilha de informação, em que o topo da pilha é, inicialmente, ocupado pelo nó raiz, sendo que este vai sendo sucessivamente substituído pelos nós filhos mais profundos e ainda não expandidos [1]. Quanto às complexidades temporal e espacial verificamos que a primeira é definida pela expressão $\mathbf{O(b^m)}$, onde **m** constitui a profundidade máxima explorada e **b** o fator de ramificação, e representa o tempo de execução necessário dada a dimensão da árvores de busca. Já a complexidade espacial remete para a memória requerida para a execução da procura, dado o número de nós a explorar. Este tipo de complexidade permite armazenar o caminho entre um nó raiz e o seu nó filho, bem como todos os nós ainda não expandidos [1]. Assim, a complexidade espacial pode ser definida por **O(bxn)**, onde **b** é o fator de ramificação e **n** o número de nós não explorados. Este tipo de busca é preferencialmente utilizado quando estamos perante problemas que têm várias soluções, por ser mais rápido (quando comparado com a Busca em Largura) e ter a capacidade de detetar a solução após explorar um pequeno espaço de busca. Porém, esta estratégia não é ótima nem completa, uma vez que pode criar caminhos redundantes que não atingem o objetivo final. Tal torna-se evidente em árvores de grande profundidade ou profundidade infinita, pelo que a sua utilização não está indicada nestas situações [1].

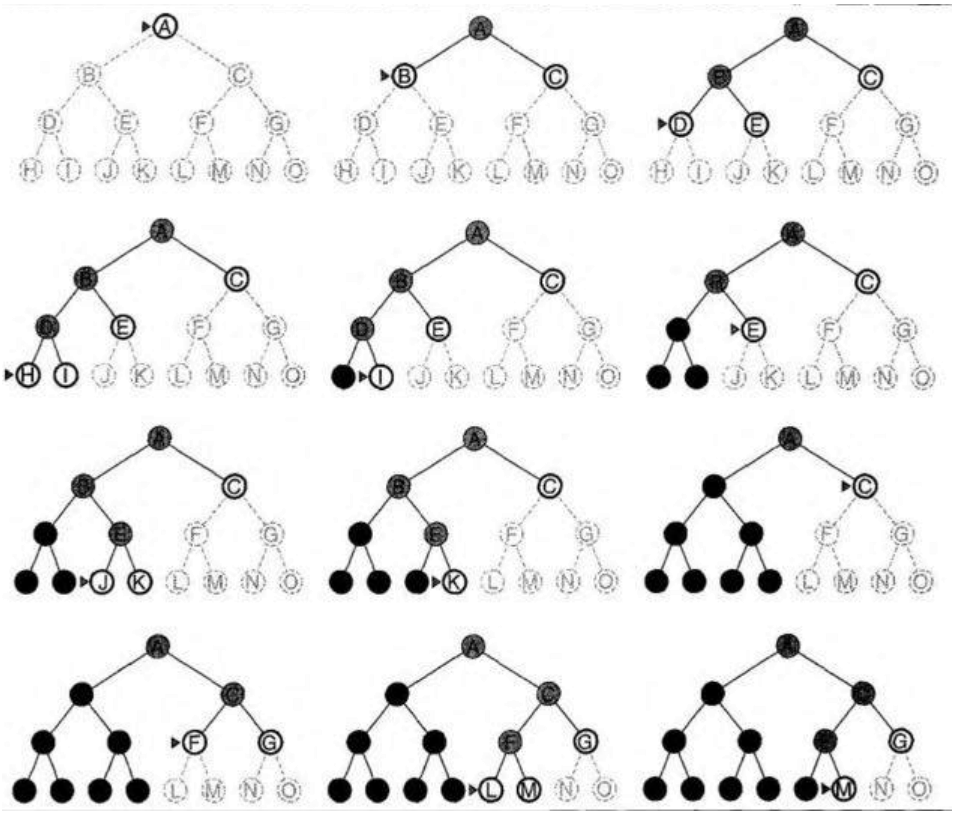


Fig.1 – Modelo de Busca em Profundidade [1]

ii. Busca em Largura (BFS – *Breadth-First Search*) – Neste método, a pesquisa é feita em amplitude, ou seja, primeiramente avalia-se o nó raiz, sendo que seguidamente há a expansão de todos os seus nós filho, passando depois a explorar os nós sucessores dos nós filho [1]. Este processo repete-se até se alcançar o objetivo – solução mais próxima da raiz (figura 2). Deste modo, só após explorar todos os nós de um dado nível de profundidade, é que se avança para o nível seguinte da árvore de busca. Para realizar esta pesquisa utiliza-se o principio FIFO, no qual se hierarquiza a expansão dos nós, dando prioridade aos mais "velhos" (nós pais e que, por isso, se encontram em menor profundidade) [1]. De notar que esta forma de procura retém una característica importante: é completa, uma vez que se o nó objetivo estiver a um dado nível finito de profundidade, irá ser encontrado após se gerarem todos os outros desse mesmo nível que não superaram o "teste do objetivo" [1]. Por outro lado, se existir mais do que uma solução, é certo que esta irá encontrar a mais rasa em primeiro lugar [2]. Porém, só se constitui como ótima se o custo do caminho for uma função não decrescente da profundidade do nó [1],[2]. Quanto às suas complexidades, verificamos que tanto a temporal como a espacial se definem igualmente: $\mathbf{O(b^m)}$. Não é aconselhada a sua utilização quando o fator de ramificação da árvore de busca é muito elevado, ou quando a solução se situa a grande profundidade (rápido esgotamento da memória). Assim, pode aplicar-se em problemas cuja solução se encontra num nível baixo de profundidade.

Fig.2 – Modelo de Busca em Largura [1]

iii. Busca em Profundidade Iterativa – Este tipo de procura possui muitas semelhanças com a busca em profundidade. No entanto, a árvore de busca possui um limite máximo de profundidade em cada iteração, ou seja, a procura de solução é feita incrementando gradualmente o limite da árvore. Assim, quando se inicia a procura, a árvore começa com o nó raiz, que traduz o nível 0, avançando seguidamente para o nível 1 (onde se encontram os nós filho). Caso estes não correspondam à solução, a busca retorna ao início, construindo novamente a árvore e adicionando mais um nível à sua procura. Este mecanismo repete-se até que se encontre o estado final desejado – solução ótima (figura 3) [1]. Deste modo, podemos assumir que a busca iterativa compreende, não só características da busca em profundidade, como também da busca em largura, permitindo percorrer todo o espaço de estados sem que haja um elevado gasto de memória, uma vez que, apesar de parecer um "trabalho desnecessário", a geração repetida dos mesmos nós não aparenta ter um custo elevado [1]. Quanto às complexidades, a temporal pode ser definida por $\mathbf{O(b^m)}$ [2], enquanto a espacial se apresenta como **O(bxm)**. Recorre-se a este método de busca quando as árvores de procura são bastante extensas e a profundidade não é conhecida [2].

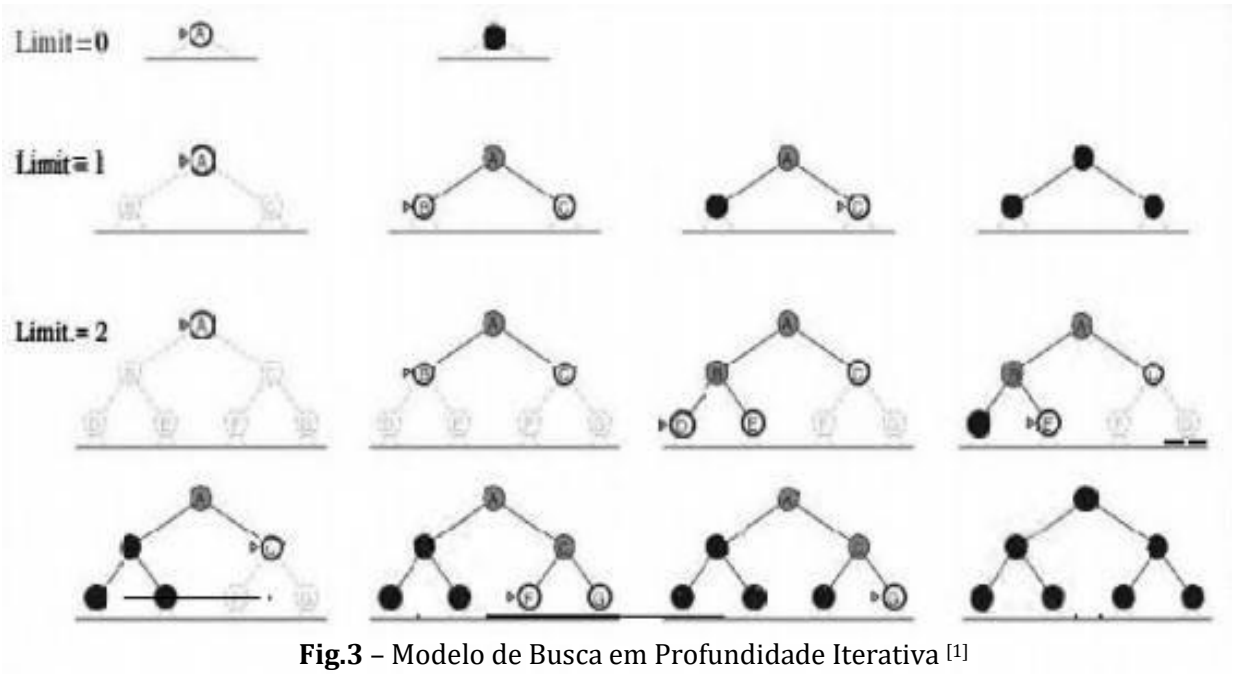


Fig.3 – Modelo de Busca em Profundidade Iterativa [1]

II. Procura Guiada/Informada

Contrariamente às estratégias anteriores, este método de procura utiliza conhecimentos específicos acerca da definição do problema em si mesmo. Deste modo, tem a capacidade de encontrar soluções de forma mais eficiente (poupança de tempo e memória) quando comparado com as buscas não informadas [1].

Estes tipos de procura recorrem a uma aproximação designada por *best-first search*, na qual um nó é selecionado para expansão tendo por base uma função avaliadora (heurística) – h(n) – capaz de nos fornecer uma estimativa de custo entre um dado nó e o estado final, fazendo com que aquele que possui a avaliação de custo mais baixa seja expandido primeiro [1]. Assim, segundo este algoritmo, o melhor nó será aquele que se encontra mais próximo do objetivo, ou que está no caminho de melhor custo [3].

i. Pesquisa Gulosa (*Greedy best-first search*) – Toma esta designação por expandir, em primeiro lugar, o nó que, aparentemente, se encontra mais próximo do objetivo, considerando que tal jogada culminará numa solução mais rapidamente [1]. Uma vez que não expande nós que não se encontrem no caminho do estado final, o seu custo é minimizado [1],[3]. De notar que este método revela algumas similaridades à Busca em Profundida, na medida em que percorre um caminho da árvore de busca sempre na mesma direção, com o objetivo de chegar à solução [3]. Porém, tem sempre a possibilidade de mudar de trajeto, quando avaliado o custo dos nós ainda não expandidos. Apesar de ser rápido e consumir menos memória, é incompleto e não ótimo, pois ao encontrar uma *dead-end* pode acabar por escolher expandir um nó com maior custo [3]. No que respeita as suas complexidades, tanto a complexidade temporal como a espacial podem ser definidas por $\mathbf{O(b^m)}$ [3], onde **b** representa o fator de ramificação e **m** a profundidade máxima explorada. De notar que, com uma boa função heurística, a complexidade pode ser significativamente reduzida, sendo esta diminuição depende do problema particular e da qualidade da heurística [1]. Esta busca é, geralmente, aplicada para solucionar problemas em que existem vários caminhos possíveis e pretendemos uma solução rápida que não necessita de ser ótima.

ii. Pesquisa A* (*A-star search*) – Tal como a estratégia anteriormente apresentada, esta recorre igualmente à função heurística – h(n). Porém, a avaliação dos nós não se baseia somente nessa função, mas também em g(n), responsável por estimar o custo do caminho desde o nó raiz até ao nó a considerar [1]. Assim, os nós são definidos por um f(n) igual a h(n)+g(n), o que nos permite obter o custo estimado da melhor solução dentro de n [1]. Este método de procura é, não só ótimo, como também completo, pelo que sempre que existir uma solução, ele irá encontrá-la. O que permite torná-la uma busca ótima é o facto de, se h(n) for uma heurística admissível, ou seja, não sobrestimar o custo para obtenção do objetivo, então também f(n) será admissível. Assim, esta heurística apresenta uma "visão positiva" pelo simples facto de considerar que o custo de solucionar o problema é mais baixo do que na realidade é [1]. Este tipo de busca tem a potencialidade de ser aplicado a qualquer problema que requer uma solução ótima e rápida.

Para a resolução do "Jogo dos 8" recorremos a duas heurísticas:

Primeira Heurística

- Indica-nos o número de peças que se encontram na posição errada. Neste caso, varia entre 0 e 8.

Segunda Heurística

- Indica-nos a soma das distâncias das peças às suas posições originais (Distância de Manhattan). Varia entre 0 e 24.

Descrição da Implementação

Para o desenvolvimento deste trabalho optamos pela linguagem de programação C++. A sua escolha passou pelo facto de, em primeiro lugar, estarmos mais familiarizados com a sua forma de funcionamento, seguindo-se o facto de esta ocupar menos memória do que a linguagem JAVA (esta requer máquina virtual), ser orientada para objetos e de o código resultante de uma compilação ser muito eficiente, aspeto esse que se deve a uma dualidade de linguagem de alto e baixo nível, assim como a um tamanho reduzido da linguagem em si mesma.

No decorrer da implementação, recorremos a uma estrutura de dados baseada numa *struct* (la) que funciona como um nó da árvore que vai sendo criada. Esta estrutura "la" contém uma matriz 3x3 com a configuração do estado do "mapa" a cada momento, um apontador para o nó pai – fundamental para o desenvolvimento sucessivo da árvore por permitir, de forma simples, a ligação entre um nó filho e o pai –, um carater (*char*) indicativo da jogada realizada para atingir a posição em que se encontra, um inteiro que nos informa sobre a profundidade a que esse nó se encontra e ainda um outro inteiro utilizado para guardar um custo, ou seja, para a heurística dos algoritmos informados.

Introduzimos também uma *binary heap* criada por uma *priority queue* responsável por organizar os apontadores dos nós (os *pla's*), para que estes se encontrem ordenados de acordo com o custo (o mais baixo na primeira posição), permitindo selecionar, de forma fácil e rápida, aquele que se pretende analisar em primeiro lugar (corresponderá ao que apresenta menor custo).

Assim, ao adotarmos o uso de apontadores e *binary heaps*, construímos um programa que utiliza menos memória, tornando-se, por consequência, mais rápido e eficiente na execução dos algoritmos.

Resultados

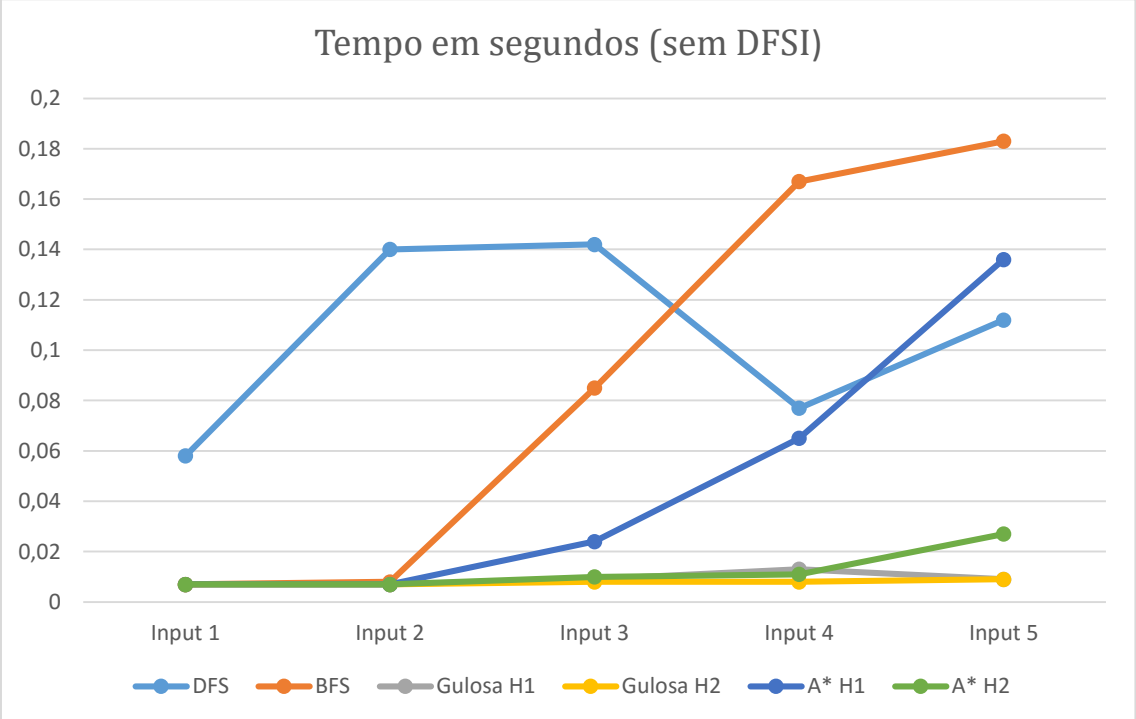
Nas tabelas a seguir apresentadas estão indicados os resultados obtidos nos vários parâmetros estudados.

Tempo(s)	**DFS**	**BFS**	**DFSI**	**Gulosa H1**	**Gulosa H2**	**A* H1**	**A* H2**
Input 1	0,058	0,007	0,551	0,007	0,007	0,007	0,007
Input 2	0,140	0,008	0,776	0,007	0,007	0,007	0,007
Input 3	0,142	0,085	1,358	0,009	0,008	0,024	0,010
Input 4	0,077	0,167	2,152	0,013	0,008	0,065	0,011
Input 5	0,112	0,183	8,144	0,009	0,009	0,136	0,027

Espaço(nós)	**DFS**	**BFS**	**DFSI**	**Gulosa H1**	**Gulosa H2**	**A* H1**	**A* H2**
Input 1	24360	54	52	9	9	9	9
Input 2	90677	449	89	31	21	48	21
Input 3	99516	56189	46856	486	281	5844	761
Input 4	31883	109633	77984	1599	305	20763	1063
Input 5	47415	175679	163205	748	506	66071	6491

Solução	DFS	BFS	DFSI	Gulosa H1	Gulosa H2	A* H1	A* H2
Input 1	23753	5	5	5	5	5	5
Input 2	85754	10	10	14	10	10	10
Input 3	93074	20	20	58	50	20	20
Input 4	31075	23	23	127	55	23	23
Input 5	45947	27	27	79	67	27	27

Nota: A cor verde encontram-se destacadas as soluções ótimas.



Gráficos 1 e 2 – Representativos da primeira tabela apresentada.

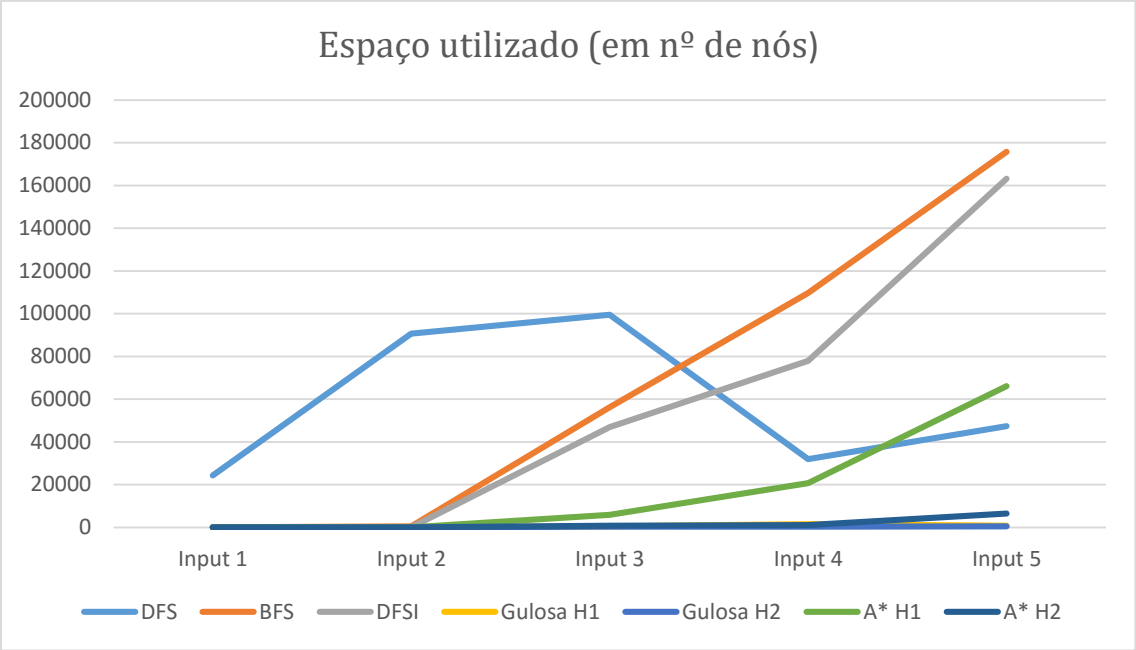


Gráfico 3 – Representativo da segunda tabela apresentada.

Comentários Finais e Conclusões

Após uma análise dos resultados, comprovamos que, apesar de algumas diferenças em termos de tempo e consumo de memória, todos os nossos algoritmos foram capazes de alcançar uma solução em cada um dos cinco *inputs*, sendo que os algoritmos informados se revelaram, como esperado, bastante mais eficientes do que os não informados.

No que respeita as procuras informadas, foi notório que o algoritmo com melhor desempenho em termos de tempo (entre 0.007s e 0.009s) e com menor consumo de memória (entre 9 e 1599 nós) foi o correspondente à Busca Gulosa. No entanto, este nem sempre foi capaz de alcançar uma solução ótima (tal só se verificou para *inputs* pequenos). Como segundo classificado encontramos o modelo de procura A* que, apesar de não tão eficaz, encontrou sempre a solução ótima.

Com o recurso a duas heurísticas de dimensões distintas, foi possível concluir que estas desempenham um papel vital na eficácia do algoritmo, dado que para o mesmo input, a heurística de menor dimensão tem um consumo de tempo e memória muito mais acentuado do que a heurística maior. Tomemos como exemplos desta situação os valores obtidos no *input* 5 para o tempo e espaço da procura A* e o input 2 da busca Gulosa (com a heurística dois esta torna-se capaz de atingir a solução ótima, ao contrário do que ocorre com a heurística um).

Quanto aos algoritmos não informados, o que contempla o melhor desempenho temporal é o método de procura em profundidade. Apesar de rápido para *inputs* onde a solução se encontra a algumas jogadas de distância, este algoritmo é pouco eficiente pois se, por exemplo, o 0 se encontrar no centro e a solução passar por deslocá-lo para baixo e para a direita, mas se, por oposição, a implementação do algoritmo ordenar uma primeira procura para cima e para a esquerda, o tempo necessário irá aumentar.

Já entre a procura em largura e a procura em profundidade iterativa apuramos que ambos chegam à solução ótima, sendo que a busca iterativa requer mais tempo para o fazer, apesar da particularidade de despender menos memória.

Em suma, foi-nos possível concluir que a escolha do algoritmo está dependente de alguns fatores, entre os quais se destacam a procura de uma solução ótima, a quantidade de memória a despender e o tempo disponível.

Bibliografia

[1] Russell, Stuarrt J.; Norvig, Peter, 2009. *Artificial Intelligence a Modern Aproach*. 3ª edição, Pearson Education, Inc., 13-978-0-13-604259-4

[2] Aula Teórica *Estratégias não informadas de Busca*, Prof. Inês Dutra

[3] Aula Teórica *Estratégias informadas de Busca*, Prof. Inês Dutra