Pontifícia Universidade Católica de
Minas Gerais

PUC Minas

Inteligência Artificial - Trabalho 2

8-Puzzle

Alunos	Gabriel Luciano Gomes
	Geovane Fonseca de Sousa Santos
	Luigi Domenico Cecchini Soares
	Paulo Junio Reis Rodrigues
Professor	Cristiane Neri Nobre

Belo Horizonte, 13 de junho de 2019

1 8-Puzzle

O quebra-cabeça pode ser visualizado como uma matriz de dimensões $3x3$, em que oito das nove células contém um número de um a nove; o quadrado restante se caracteriza por ser vazio. A partir de um estado inicial S_i, o objetivo é que se alcance uma configuração final S_j por meio de ações durante o jogo. Estas ações se referem a trocar de posição alguma célula preenchida (um a nove) com o quadrado vazio. Para tanto, estas devem estar próximas (adjacentes) ao espaço vazio. Um exemplo de configurações inicial e final pode ser visto a seguir:

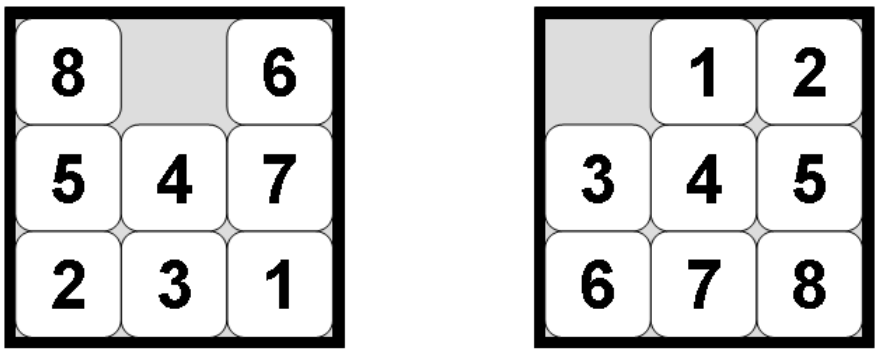

Figura 1: Pior caso inicial (à esquerda) para configuração final (à direita)
Fonte: `http://www.aiai.ed.ac.uk/~gwickler/eightpuzzle-uninf.html`

2 Métodos de busca

2.1 Busca em largura (amplitude)

Formalmente, a busca em largura é um método de busca não-informada que expande e examina os vértices de um grafo. Nesse sentido, diferente das técnicas de busca-informada, esta não utiliza $g(n$ ou $h(n)$; apenas explora os vizinhos de um vértice até que se encontre o nó final.

O caminho realizado pelo algoritmo leva em consideração a visita aos nós vizinhos do atual. Isto é, os vértices $u \in U$ adjacentes a v são inseridos em uma fila (FIFO) para serem explorados nas próximas iterações. Em outras palavras, considerando um grafo do tipo árvore, a busca em largura a percorrerá nível por nível, visitando os vértices de uma altura h_i antes de passar para $h_j, j > i$.

Tem-se como pior caso do algoritmo a exploração de todos os vértices presentes no grafo. Dessa forma, considerando um grafo representado por meio de uma lista de adjacências, a complexidade de tempo é dada por $\mathcal{O}(|E| + |V|)$, sendo E a quantidade de arestas e V e a de vértices. Ademais, a complexidade de espaço é representada por $\mathcal{O}(V)$.

Por fim, a busca em largura pode ser considerada completa e ótima em determinadas situações. Para ser considerado um algoritmo completo, é necessário que a quantidade de nós deve ser finita. Já para ser considerado ótimo, é preciso que os pesos das arestas sejam iguais. Nesse caso, considerando o tema abordado (8-puzzle), tem-se que o peso das arestas é sempre igual (1) e que a quantidade de estados é finita. Portanto, para o problema considerado neste relatório, a busca em largura é tanto completa quanto ótima.

2.2 Busca gulosa

Esta é uma técnica de busca que explora um grafo $G = (V, E)$ a partir da expansão dos vértices mais promissores. Isto é, a ordem de visita dos nós é dada pela minimização da função $f(n) = h(n), n \in V$. Nesse sentido, $h(n)$ se refere a uma heurística que procura predizer o quão perto do objetivo se encontra o estado n atual. Esse método é comumente implementado através de uma fila ordenada pelo menor valor de $h(n)$.

De forma geral, a busca gulosa não costuma expandir vértices que não fazem parte da solução, embora seja um caso possível de ocorrer. Por esse motivo, se mostra um algoritmo bastante eficiente; se comparado a busca em largura e ao A*, implementados neste trabalho como solução para o 8-puzzle, é a técnica de melhor desempenho. Por fim, caracteriza-se como um método de busca completo, tendo em vista que se existe uma solução esta será encontrada. Entretanto, não pode ser considerado como um algoritmo ótimo, posto que a primeira solução encontrada nem sempre será a ótima.

2.3 A*

Também conhecido como busca heurística, o algoritmo A* se caracteriza por ser uma mescla entre as técnicas de busca gulosa e busca de custo uniforme (Dijkstra). Em outras palavras, levando em consideração que o algoritmo guloso procura minimizar $f(n) = h(n)$ e que o método desenvolvido por Dijkstra consiste em minimizar $f(n) = g(n), g(n)$ sendo o somatório dos pesos "reais"das arestas entre o vértice inicial e o nó n, tem-se que o A* apresenta como objetivo a minimização de uma função $f(n) = g(n) + h(n)$.

Ademais, o A*, assim como o método de busca guloso, é caracterizado por ser completo. Por fim, essa técnica pode ser considerada ótima caso uma condição seja satisfeita: $h(n)$ deve ser uma heurística admissível. Dada uma distância real $h^*(n)$ entre o vértice atual n e o final, para que a heurística seja admissível deve-se ter $h(n) \leq h^*(n)$.

3 Resultados e discussões

Para a validação dos algoritmos, foram usados dois exemplos para teste. O primeiro foi um exemplo mais fácil de ser resolvido, e o segundo é mais difícil de se encontrar uma solução. Isso foi feito para verificar como os algoritmos se comportam em duas situações diferentes. Os dois modelos podem ser observados na figura 2.

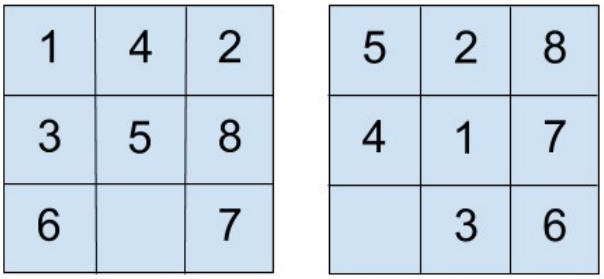

Figura 2: Primeiro teste (à esquerda), segundo teste (à direita)
Fonte: Criação do grupo

3.1 Primeiro teste

Para a primeira sequência foi notado que os algoritmos de busca Gulosa e o A* tiveram características semelhantes, quanto tempo de resposta e custo. Entretanto, a solução encontrada utilizando busca heurística por esta, foi melhor do que a Gulosa. Por outro lado, o algoritmo BFS não apresentou resultados satisfatórios, uma vez que possui um custo elevado e, consequentemente, leva mais tempo para ser processado e encontrar a solução. Porém, esta resposta é ótima, já que nesse caso a distancia de um para outro bloco é sempre a mesma. O teste realizado se encontra na figura 3.

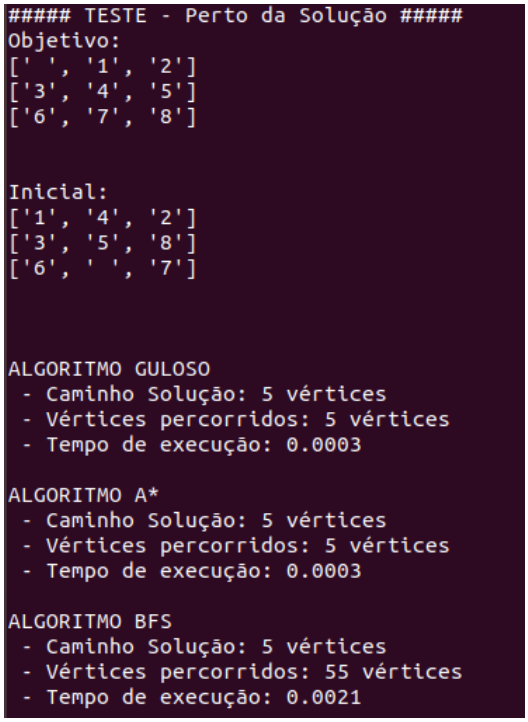

Figura 3: Resultado do primeiro teste
Fonte: Criação do grupo

3.2 Segundo teste

Já para o segundo teste, pode-se perceber a discrepância nos resultados. O algoritmo guloso teve o resultado menos custoso, por explorar apenas a heurística da árvore gerada. Entretanto, esta não é a solução ótima. Por outro lado, o A* explora não só a heurística, como também o valor real do caminho, que possibilita uma busca mais eficiente no problema. Sendo assim, apesar de ser mais custoso, a solução ótima é encontrada. Por fim, o algoritmo BFS, mais uma vez, fica para trás. Isso porque neste não há uma ”lógica”na busca, o que justifica a quantidade de vértices percorridos e o tempo de execução. Contudo, ele encontra ainda a solução ótima. O teste realizado se encontra na figura 4.

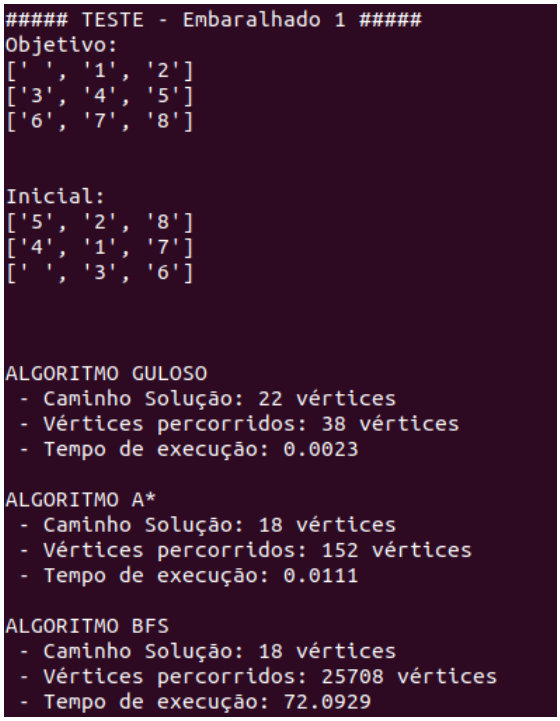

Figura 4: Resultado do segundo teste
Fonte: Criação do grupo

3.3 Uso da memória

Ao analisar os algoritmos em relação do uso de memória, têm-se o custo de criação de novos nós da árvore. Cada um destes é uma possibilidade para o tabuleiro, o algoritmo Guloso pode não encontrar a solução ótima. Contudo é o algoritmo que gasta menos memória, sendo o mais rápido dentre os três listados. Já a busca A* encontra a melhor solução, entretanto há um gasto maior de memória levando em comparação o método Guloso. Por fim, o algoritmo BFS possui alto custo de execução e memória, conforme analisado nos exemplos acima. Sendo assim, este é o pior dentre os três algoritmos implementados.