

Engenharia de Computação

7⁰ Período

Inteligência Artificial

Projeto Tema 1 – Agente para solução do quebra-cabeça de 8 peças

Professor Responsável

Prof. Antônio J. de Lima Batista

Autores

Joaquim Augusto Pomarico

Marcos Vinicius Moreira

Otávio Messias Palma

Yuri Fernandes Oliveira

Poços de Caldas, 15 de abril de 2019

# Introdução

O quebra-cabeça de 8 peças é um jogo de peças deslizantes que consiste num tabuleiro de quadrados numerados posicionados de forma aleatória com um dos quadrados faltando (Figura 1). O quebra-cabeça é encontrado em outros tamanhos como o de 15 peças (original). O objetivo do jogo é posicionar os quadrados em ordem deslizando os quadrados através do espaço vazio a fim de se obter o estado objetivo da Figura 2.

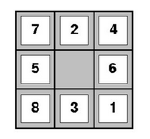


Figura - Exemplo do quebra-cabeça de 8 peças

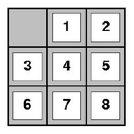


Figura - Estado objetivo do quebra-cabeça de 8 peças

O quebra-cabeça de “n” peças é um problema clássico para modelar algoritmos envolvendo heurísticas, isso ocorre por conta do alto custo para se achar a solução ótima utilizando algoritmos de busca sem informação, as heurísticas mais comuns para este problema são a contagem de peças fora do lugar e a soma das distancias de Manhattan de cada peça, sendo estas duas admissíveis, ou seja, nunca ultrapassam o custo real para se chegar ao objetivo.

Dependendo da situação é possível fazer quatro, três ou duas ações. A primeira opção ocorre quando a peça faltante se encontra no meio; a segunda quando, a peça faltante se encontra nos lados do tabuleiro e a terceira, quando a peça faltante se encontra nos cantos do tabuleiro.

O custo médio da solução para o problema é de aproximadamente 22 passos. Assim, uma busca em profundidade com fator de ramificação de 3 (média entre quatro, três e dois movimentos possíveis) geraria 322 estados. Já uma busca em grafos reduziria para 9! / 2 estados, uma vez que metade dos estados não são alcançáveis. Apesar de ser um número gerenciável, é necessário se utilizar de alguma heurística para resolver o problema em tempo hábil e com solução ótima.

# Algoritmo

O algoritmo implementado baseia-se em duas buscas: busca gulosa e busca A\* (A estrela). Ambas as buscas são informadas, isto é, se utilizam de uma heurística a fim de achar a solução ótima de forma ágil e em tempo hábil.

É importante notar o fato de que cerca de metade dos estados inicias do quebra-cabeça de 8 peças não possuem solução. Assim, foi necessário verificar se o estado inicial gerado de forma aleatória possuía solução antes de passar o caso para as buscas.

## Verificação

Seguindo os dois exemplos dados nas figuras 4 e 5, temos um estado solucionável e um não solucionável. Para verificar se o estado é solucionável, devemos seguir uma simples regra: não é possível solucionar, por uma série de movimentos de peças, um estado do quebra-cabeça de 8 peças que possua um número de inversões ímpar.

Como podemos ver, na figura 4, o número de inversões é 10. Logo, por ser um número par, podemos afirmar que existe uma solução para esse estado inicial.

Já na figura 5, entretanto, o número de inversões é 11. Sendo 11 um número ímpar, concluímos que o estado representado na figura 5 não possui solução.

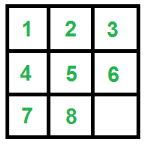


Figura - Estado objetivo

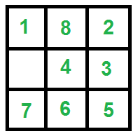


Figura - Exemplo 1 (10 inversões)

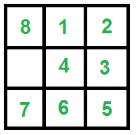


Figura - Exemplo 2 (11 inversões)

A “inversão” constitui-se de um quadrado que precede outro quadrado com valor inferior ao primeiro. Exemplo: na Figura 4, o quadrado 8 precede os quadrados 2, 3, 4, 5, 6 e 7. Logo, temos seis inversões de valores. Já o quadrado 4 precede o quadrado 3, gerando uma inversão. O quadrado 7 precede os quadrados 5 e 6, gerando duas inversões. E, por último, o quadrado 6 precede o quadrado 5, gerando uma inversão. Somando o número de inversões, obtemos dez inversões no total, sendo um número par e, portanto, o estado é solucionável.

Na Figura 5, temos que o quadrado 8 precede todos os quadrados posteriores a ele, totalizando sete inversões; o quadrado 4 precede o 3, totalizando uma inversão; o quadrado 7 precede os quadrados 6 e 5, totalizando duas inversões e, por fim, o quadrado 6 precede o quadrado 5, gerando uma inversão e totalizando 11 inversões que, sendo um número ímpar, leva a conclusão que este é um estado não solucionável.

Após tal verificação, caso haja solução para o estado gerado, o algoritmo irá rodar a Busca Gulosa ou A\* de acordo com a escolha do usuário.

## Heurística

A heurística utilizada no problema do quebra-cabeça de oito peças foi a soma das distancias de Manhattan. Em tal algoritmo, faz-se a distância de Manhattan para cada peça e soma-se todos os valores a fim de obter um valor total. A distância de Manhattan utiliza o sistema de coordenadas para calcular a real distancia de uma origem ao seu objetivo.

Na Figura 6, por exemplo, a peça 7 se encontra na coordenada (0,0). No entanto, seu objetivo se encontra na coordenada (2,1). Assim, para calcularmos a distância de Manhattan fazemos | x1 – x2 | + | y1 – y2 |. No caso da peça 7, será feito | 0 - 2 | + | 0 – 1 |, o que nos dá uma distância de Manhattan de 3. Faz-se para todas as peças essa operação, obtendo, da soma dos resultados, o valor de 18. Considerando que a solução verdadeira leva 26 movimentos para ser solucionada, a heurística da distância de Manhattan nesse caso pode ser considerada admissível, uma vez que o número obtido nunca será maior que o custo real, que é 26. Note que, para o caso da figura 6, temos 16 inversões, ou seja, o estado é um caso possível de solução.

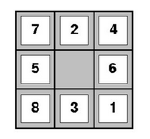


Figura 6 - Exemplo 1

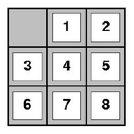


Figura 7 - Estado objetivo

## Busca Gulosa

A busca gulosa tenta expandir o nó que está mais próximo do objetivo, com fundamento de que isso pode conduzir a uma solução rapidamente. Sendo assim, no caso do algoritmo implementado para o quebra-cabeça de 8 peças a soma das distancias de Manhattan, ela avalia os nós usando apenas a função heurística.

Na busca gulosa, o nó raiz é expandido e gera seus filhos. No caso da Figura 6, teremos 4 filhos, uma vez que a peça faltante se encontra no meio, e 4 movimentos possíveis. Teremos, portanto, os seguintes estados:

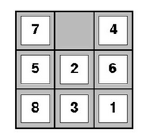


Figura 8 - Estado 1

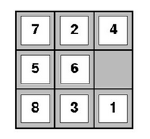


Figura 9 - Estado 2

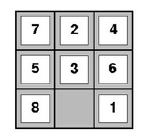


Figura 10 – Estado 3

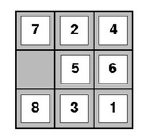


Figura 11 – Estado 4

Para o Estado 1, a distância de Manhattan total será de 19. Já para o Estado 2, a distância será de 17. Para o Estado 3, 17, e para o Estado 4, a distância será de 17. Dessa forma, a busca gulosa irá expandir os filhos do Estado 2, uma vez que a heurística do mesmo apresenta um valor menor que a do Estado 1. A forma como o algoritmo foi implementado faz com que caso haja valores iguais, o filho mais à esquerda é o escolhido para expansão. Assim, o Estado 2 irá gerar mais 3 filhos, sendo que 1 deles é o nó raiz que será desconsiderado para impedir que entremos em um loop infinito. Essa iteração continua até que o estado objetivo seja encontrado.

Caso um estado possua uma distância de Manhattan de, por exemplo, 15 no nível 3, e outro estado possua o mesmo valor 15 no nível 2, a busca gulosa irá expandir o nó que está mais acima da árvore, independente do posicionamento do nó na árvore de estados.

É importante notar que a busca gulosa não consegue sempre achar a solução ótima uma vez que ela só se importa com o valor da heurística, expandido os valores menores até chegar no objetivo. Sendo assim, o usuário que jogar contra a busca gulosa poderá obter uma resolução melhor ou igual a busca gulosa.

## Busca A\* (A estrela)

A forma de solução mais amplamente conhecida de busca de melhor escolha é a chamada busca A\*. Essa solução avalia os nós através da combinação do custo para alcançar o nó (heurística) e o custo para ir do nó ao objetivo (nível da árvore) e, caso nossa função heurística seja admissível, isto é, a soma do custo para ir do nó ao objetivo mais o custo para alcançar o nó nunca superestima o custo real de atingir o objetivo, a busca A\* será completa e ótima.

Utilizando A\* no caso da Figura 6 teremos 4 filhos estados, uma vez que a peça faltante se encontra no meio, e temos 4 movimentos possíveis.

Note que, neste caso, os estados gerados são semelhantes aos da busca gulosa, no entanto, a busca A\* considera além da heurística (19, 17, 17 e 17 para cada estado respectivamente), o custo para ir do nó raiz ao objetivo. Como nos encontramos no nível 1 da árvore de estados, teremos valores de 20, 18, 18 e 18 respectivamente. Na expansão do Estado 2, seus filhos válidos (isto é, aqueles que não são iguais a um estado já expandido) terão valores 18 e 18 (16 de heurística + 2 de custo da raiz a objetivo). Sendo assim, como os nós do nível 1 possuem os mesmos valores, mas estão em um nível mais acima, eles serão expandidos previamente.

# Ferramentas de Implementação

Foram utilizados os softwares NetBeans IDE 8.2 para implementação do código e GitKraken 5.0.4 para versionamento do código no GitHub.

# Resultados



Figura 12 - Exemplo

No resultado da busca gulosa para o exemplo da Figura 12, teremos uma lista com 162 nós e o total de passos para solucionar o problema foi de 67.

Já no resultado da busca A\* para o exemplo da Figura 12, teremos uma lista com 92 nós e o total de passos para solucionar o problema de 17, valor igual a solução real.



Figura 13 - Exemplo

No exemplo da Figura 13, para a busca gulosa, temos 111 nós expandidos e um total de 78 passos para resolver o problema. Para a busca A\*, temos 4072 nós expandidos, apesar do alto número de nós, chegamos à solução ótima de 26 passos. Esse número alto ocorre por conta do número excessivo de nós expandidos em níveis mais baixos (próximos ao 20). Sabe-se que a solução ótima deste problema leva 26 passos para se solucionar, assim, a gulosa está longe de ser perfeita. Todavia, ela encontra um resultado com muito menos uso de memória.

# Conclusão

Durante o desenvolvimento do projeto, foi observado que a busca gulosa, na maior parte dos estados iniciais válidos, não encontrava uma solução em tempo hábil. Com a implementação da estratégia de backtracking, foi possível encontrar solução em tempo hábil para uma grande maioria de estados. Todavia, utilizando A\*, grande parte das vezes é possível encontrar uma solução e, em sua grande maioria, são soluções com menor caminho para serem atingidas. Conclui-se, portanto, que para encontrar a melhor solução, é mais indicado o uso do algoritmo A\* em detrimento da busca gulosa.

# Referências

### PENMAN, Tristan. **N-Puzzle**. 2019. Disponível em: http://tristanpenman.com/demos/n-puzzle. Acesso em: 14 abr. 2019.

### KUMAR, Chandan. **How to check if an instance of 8 puzzle is solvable?**. 2019. Disponível em: https://www.geeksforgeeks.org/check-instance-8-puzzle-solvable. Acesso em: 14 abr. 2019.

### JUNIOR, Nelson F. GUIMARÃES Frederico G. **Problema 8-Puzzle: Análise da solução usando Backtracking e Algoritmos Genéticos**. Ouro Preto, 2019. Disponível em: http://www.decom.ufop.br/menotti/paa111/files/PCC104-111-ars-11.1-NelsonFlorencioJunior.pdf. Acesso em: 14 abr. 2019.

### MATHEMATICS. **How to check if a 8-puzzle is solvable?**. 2019. Disponível em: https://math.stackexchange.com/questions/293527/how-to-check-if-a-8-puzzle-is-solvable. Acesso em: 14 abr. 2019.

### NORVIG, P.; RUSSEL, S. **Inteligência Artificial**. 3ª ed. Rio de Janeiro: Campus. 2013.