Solução do 8-*Puzzle* por meio do algoritmo A*

Marco Cezar Moreira de Mattos[1], **Rômulo Manciola Meloca**[1]

[1]DACOM – Universidade Tecnológica Federal do Paraná (UTFPR)
Caixa Postal 271 – 87301-899 – Campo Mourão – PR – Brazil

{marco.cmm,rmeloca}@gmail.com

Resumo. *Relata o procedimento tomado para construir um algoritmo A* que soluciona o problema 8-Puzzle e os testes feitos sobre ele.*

1. O Problema

O jogo 8-*Puzzle*, aos olhos humanos possui uma solução, que embora não seja trivial, bastante intuitiva, dado seu objetivo. Consiste em um tabuleiro 3x3 sobre o qual deslizam oito peças enumeradas, onde os únicos movimentos possíveis para se atingir o objetivo são aqueles permitidos pelo buraco deixado pela nona peça. O objetivo do jogo é ordenar o tabuleiro.

O problema ocorre quando não há um agente dotado de intelecto para resolver o problema, não pela complexidade das verificações feitas para atingir-se o objetivo, mas sobre quais decisões devem ser tomadas em cada estado do problema, para atingir-se a solução do problema.

O espaço dos estados do 8-*Puzzle* é 9! e a solução ótima tem classe NP-Completo, portanto, sortear o próximo estado ou expandir todas as possíveis soluções jamais poderia obter a solução em tempo plausível, o primeiro porque a aleatoriedade possui a mesma probabilidade de caminhar rumo a solução quanto de caminhar no sentido oposto, o segundo porque demandaria processamento e memória difíceis de serem obtidos.

Enfim, problemas cujos espaço dos estados fogem da possibilidade viável de computação dado a complexidade do algoritmo, são resolvíveis por meio do uso de inteligência artificial, que, muito embora não forneça a melhor solução, fornece uma solução muito boa em tempo muito bom (é claro que alguns tipos de problemas são melhores resolvidos com determinados tipos de algoritmos observando-se os determinados parâmetros que o fazem comportar-se bem).

Para o 8-*Puzzle* é possível lançar-se mão desta categoria de algoritmos, contudo neste trabalho, utilizou-se o algoritmo A* que não é capaz de aprender (uma vez que armazenar resultados anteriores e observar se já foram visitados não é aprendizagem de máquina), mas que retorna um resultado muito bom em tempo viável dado sua capacidade de ignorar estados que afastam-se do objetivo e caminhar sempre rumo a ele.

2. O algoritmo

Para a solução do problema, utilizou-se o algoritmo A*, um algoritmo de busca heurística (e portanto gulosa), que diferencia-se dos demais por aplicar a heurística em duas etapas: A distância entre o estado inicial para o estado do vizinho somado com a distância entre o estado vizinho e o objetivo.

Data: Instância de um Puzzle a ser resolvida.
Result: Lista do caminho percorrido para solucionar o puzzle.
Insere a primeira instância na lista do caminho percorrido;
while *Puzzle não está resolvido* **do**
    Obtém o última instância do caminho;
    Obtém os possíveis movimentos da instância;
    Calcula a heurística para cada possível movimento;
    Escolhe a instância que possui melhor heurística;
    Adiciona a instância ao caminho percorrido;
end

Algorithm 1: Busca A* para resolver 8-Puzzle

Para calcular a heurística, conforme supramencionado, utilizou-se a distância de *Manhattan* e o número de peças trocadas multiplicados por escalares, uma adaptação do trabalho Júnior e Guimarães. A distância de *Manhattan* é calculada entre duas intâncias do puzzle fazendo-se o somatório dos módulos das distâncias entre cada elemento da matriz. O número de peças que não encontram-se na mesma posição nas duas instâncias. Define-se matematicamente a heurística pela soma das fórmulas que seguem:

Sejam A e B matrizes quaisquer e a_{ij} e b_{ij} respectivamente seus elementos,

$$\sum_{i=1}^{3}[\sum_{j=1}^{3}(|i-k|+|j-l|)] \mid a_{ij} = b_{kl} \wedge i,j,k,l \in \{\mathbb{N}^* < 3\}$$

Quantidade de elementos no conjunto $\{x \mid a_{ij} \neq b_{ij}\}$

Deste modo, para tomada de decisão sobre qual melhor vizinho a ser expandido, assume-se a seguinte heurística:

$$18[manhattanDistance(initial, son) + manhattanDistance(son, solution)] + 36[quantidadePecasTrocadas(initial, son) + quantidadePecasTrocadas(son, solution)]$$

Onde *manhattanDistance* é a distância de *manhattan*, isto é, somatório da distância (diferença de linhas + diferença de colunas) entre todos os elementos dos argumentos dados. *numeroPecasTrocadas* corresponde a quantidade de peças trocadas entre os dois argumentos dados. *initial* é a primeira instância do jogo. *son* corresponde ao nó que poderá ser expandido (dado a natureza do jogo, haverão apenas 3 ou 4 possíveis valores para son. Poderão haver valores para son já visitados que serão descartados).

Implementou-se a solução utilizando a linguagem de programação Java, contando com um objeto Puzzle e as devidas e necessárias abstrações para os movimentos possíveis, bem como a utilização de funções fornecidas pela linguagem.

3. Resultados

Executou-se os seguintes testes:

1. $18[manhattanDistance(initial, son) + manhattanDistance(son, solution)] + 36[numeroPecasTrocadas(initial, son) + numeroPecasTrocadas(son, solution)]$

2. $18[manhattanDistance(initial, son) + manhattanDistance(son, solution)] + 36[numeroPecasTrocadas(son, solution)]$
3. $manhattanDistance(initial, son) + manhattanDistance(son, solution) + numeroPecasTrocadas(initial, son) + numeroPecasTrocadas(son, solution)$
4. $manhattanDistance(initial, son) + manhattanDistance(son, solution)$
5. $18[manhattanDistance(initial, son) + manhattanDistance(son, solution)]$
6. $18[manhattanDistance(initial, son) + manhattanDistance(son, solution)] + numeroPecasTrocadas(initial, son) + numeroPecasTrocadas(son, solution)$
7. $18[manhattanDistance(initial, son) + manhattanDistance(son, solution)] + numeroPecasTrocadas(initial, son)$

Utilizou-se o conjunto de valores $\{2, 3, 1, 9, 5, 6, 4, 7, 8\}$ para o tabuleiro da instância *initial*, no qual 9 corresponde ao buraco.

Os resultados observados são os que seguem:

1. Insolúvel (56 passos).
2. 78 passos.
3. Insolúvel (200 passos).
4. Insolúvel (72 passos).
5. Insolúvel (72 passos).
6. Insolúvel (200 passos).
7. Insolúvel (164 passos).
8. Insolúvel (82 passos).

Executaram-se, ainda, testes para tomar a média de passos necessários para a solução de uma instância aleatória do Puzzle. Coletou-se 100 amostras utilizando a única heurística que retornou resultado solúvel para o puzzle. A média de passos foi de 472, com taxas elevadas de variância, de 50 à 1600. Das 100 instâncias aleatórias, não mais que 12 foram resolvidas.

4. Considerações Finais

Considera-se, deste modo, que embora 78 passos seja um número alto de movimentos para solucionar o puzzle, sabe-se que o espaço de soluções possíveis é 9!, isto é, 362880, de modo que obter a melhor solução custam recursos indisponíveis. Assim sendo, considera-se como aceitável que um algoritmo (ainda que não inteligente) resolva um problema dessa magnitude nessa quantidade de passos e com a bela velocidade de execução observada.

Sobretudo, ressalta-se que para os mais variados casos, como revelou o teste adicional de instâncias aleatórias, as heurísticas combinadas não puderam cobrir mais de 15% dos casos, número bastante ruim, mas que justificado devido ao fato de tratar-se de um algoritmo de busca, que, portanto, é incapaz de aprender.

5. Referências

JÚNIOR, Nelson F., GUIMARÃES Frederico G. **Problema 8-Puzzle:** Análise da solução usando Backtracking e Algoritmos Genéticos.