Jogo dos 15

Implementação de Métodos de Busca

Trabalho realizado por:
Miguel Filipe Miranda dos Santos (202105289)
Miguel Proença Fernandes Ramalho Amaro (202106985)
Nuno dos Santos Moreira (202104873)

1) INTRODUÇÃO

A - O que é um problema de busca/procura?

Em contexto computacional, um problema de busca é um tipo de problema representado por uma relação binária, que consiste em encontrar uma estrutura Y num objeto X.

De acordo com o livro "*Artificial Intelligence: A Modern Approach*", um problema de busca pode ser definido formalmente através dos seguintes conceitos:

- Conjunto de estados, nos quais o ambiente pode estar, ao qual se chama espaço de estados (normalmente representado por um grafo onde os vértices são estados e os vértices são ações);
- Estado Inicial, no qual o agente começa;
- Um ou mais estado(s)-objetivo;
- Ações disponíveis ao agente;
- Modelo de transição, que descreve o efeito de cada ação;
- Função de custo de ação, que denuncia o custo da aplicação de cada ação a cada estado;

Tendo definido estes conceitos, define-se caminho como uma sequência de ações (e consequentemente estados), e solução como um caminho desde o estado inicial até ao estado final. O custo de um caminho é calculado através da soma dos custos de cada ação tomada,

B – Quais são os métodos utilizados para resolver problemas de busca/procura?

Um problema de busca é resolvido através de um algoritmo, se existe pelo menos uma estrutura e uma instância desta estrutura é tida como output. Existem 2 categorias de métodos utilizados para a resolução de problemas de busca:

- Busca Não Informada – algoritmos que resolvem o problema através de "força bruta", isto é, sem qualquer informação relativa à distância e/ou custo dos nós ao objetivo (pelo que também se pode chamar "Pesquisa Cega");

- Busca Informada – algoritmo que utiliza informação relativa à localização do(s) objetivo(s), sob forma de função heurística (denotada por h(n)), para decidir que nó deve expandir. É bastante útil para problemas com espaços de pesquisa muito grandes.

Como visto, a ordem de expansão dos nós é definida pelo método de busca escolhido. De um modo geral, usa-se o seguinte código:

Algorithm 1 Algoritmo Geral de Busca

```
1: GeneralSearchAlgorithm(QueueingFunction,configInicial,configFinal)
2: if thereIsNoSolution(configInicial,configFinal) then
3:     return "It is impossible to reach a solution"
4: end if
5: queue = configInicial
6: while queue not empty do
7:     node = removeFrontNodeFrom(queue)
8:     if node is solution then
9:         return Path to solution
10:    end if
11:    descendantList = MakeDescendants(node)
12:    insert(descendantList,queue,QueueingFunction)
13: end while
14: return "solution not found"
```

fonte: ficha relativo ao relatório final

O processo de resolução passa, respetivamente, pelos seguintes passos: definição do problema; análise do problema; identificação de possíveis soluções; escolha da solução ótima; implementação. De modo a avaliar a eficiência de diferentes algoritmos, tornando possível a sua comparação, utilizamos as seguintes características:

- Completude – um algoritmo diz-se completo quando fornece uma solução para qualquer input, quando esta existe;
- Otimalidade – um algoritmo diz-se ótimo quando a solução encontrada é a melhor solução possível (a que tem menor custo no seu caminho);
- Complexidade Temporal – o tempo que um algoritmo consome durante a realização de uma tarefa é chamado de complexidade temporal. Quanto menor for a complexidade temporal, mais eficiente é o programa;

- Complexidade Espacial – quantidade máxima de memória utilizada pelo algoritmo a qualquer dado momento da execução da tarefa. Quanto menor for a complexidade espacial, mais eficiente é o programa.

(2) DESCRIÇÃO DO PROBLEMA DE PROCURA ESTUDADO, JOGO DOS 15

O problema abordado neste trabalho será o "jogo dos 15, uma variante do "n-puzzle", jogo este que existe há já centenas de anos. Embora a história da sua origem não seja totalmente clara, a sua criação é comumente atribuída a Noyes Chapman, tendo havido uma explosão na popularidade do puzzle quando Sam Loyd prometeu oferecer 1000$ a quem conseguisse resolver uma configuração do jogo (que mais tarde revelou ser impossível).

O tabuleiro deste jogo consiste numa matriz de dimensão 4x4, onde 15 das 16 células estão numeradas com números de 1 a 15, e a célula que sobra encontra-se vazia. O objetivo é, partindo de uma dada disposição das peças, chegar a uma disposição final, podendo apenas mover as peças para o lugar vazio na horizontal (direita e esquerda) e na vertical (cima e baixo).

De qualquer modo, e como referido no primeiro parágrafo, nem todas as disposições têm uma solução. Assim, para aferir a solucionabilidade recorremos a uma método que, resumidamente, passa pelos seguintes pontos:

- tem-se uma "inversão de ordem n" (denotado por n_i) se no tabuleiro um número i é sucedido de um número menor que i (sendo o tabuleiro interpretado da esquerda para a direita, de cima para para baixo);
- se o tabuleiro tem uma largura ímpar, é solucionável se o número total de inversões é par;
- se o tabuleiro tem uma largura ímpar, é solucionável se
    - o espaço vazio está numa fila par (a contar a partir da última fila, isto é, a última fila é 1, a penúltima fila é 2,…) e o número de inversões é ímpar, ou
    - o espaço vazio está numa fila ímpar (a contar a partir da última fila, isto é, a última fila é 1, a penúltima fila é 2,…) e o número de inversões é par.

Temos o seguinte exemplo:

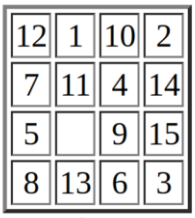

fig 4

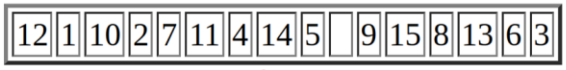

fig 5:
Tiles written in a row

https://s3.eu-central-1.amazonaws.com/unitbv.horatiuvlad.com/facultate/inteligenta_artificiala/2015/TilesSolvability.html

Neste exemplo, temos um total de 49 inversões (o 12 tem 11 inversões, o 1 tem 0 inversões, o 10 tem 8 inversões,…). Uma vez que o tabuleiro tem dimensão de 4x4 (largura par), o número total de inversões é ímpar, e o espaço vazio encontra-se numa fila par a contar da última fila, então este exemplo tem solução.

(3) ESTRATÉGIAS DE PROCURA

A – Pesquisa Não Informada

A1 – Busca Em Profundidade (Depth-First Search)

Esta estratégia de busca consiste na expansão de um caminho até à máxima profundidade possível. Deste modo, esta estratégia não é recomendada para problemas de profundidade elevada ou infinita. De qualquer modo, apenas são armazenados os nós do caminho atual e os nós ainda não expandidos. A busca em profundidade tem complexidade temporal $O(b^m)$ e complexidade espacial O(bm), sendo b o fator de ramificação e m a profundidade máxima. Não é nem completa, nem ótima.

A2 – Busca Em Largura (Breadth-First Search)

Esta estratégia de busca consiste na expansão dos nós todos de um nível, antes de expandir qualquer nó do nível seguinte. Deste modo, esta estratégia encontra a solução mais "rasa" primeiro. A busca em largura tem $O(b^d)$ como complexidade temporal e complexidade espacial, onde b é o fator de ramificação e d a profundidade da solução. É completa e ótima.

A3 – Busca Iterativa Limitada em Profundidade (Iterative Deepening Search)

Esta estratégia de busca combina as vantagens da busca em largura com a busca em profundidade, expandindo os nós em profundidade, mas apenas até um certo nível. Deste modo, esta estratégia é utilizada quando a árvore é muito grande e a profundidade desconhecida. A busca em profundidade iterativa tem complexidade temporal $O(b^d)$ e complexidade espacial O(bd), onde b é o fator de ramificação e d a profundidade da solução. É completa e ótima.

B – Busca Informada

B1 – Busca Gulosa (Greedy Search)

Esta estratégia de busca tenta minimizar o custo estimado para chegar à solução, expandindo o nó que aparenta estar mais próximo da solução. A aproximação será tanto mais acertada quanto melhor for a função heurística, h(n). A busca gulosa tem $O(b^m)$ como complexidade temporal e complexidade espacial, onde b é o fator de ramificação e m é a profundidade máxima. Não é nem completa, nem ótima.

B2 – Busca A*

Esta estratégia de busca combina a função h(n) da busca gulosa com a função g(n) da busca de custo uniforme, de modo a criar f(n)=h(n)+g(n). Na busca A*, a complexidade temporal varia de acordo com a escolha da heurística, enquanto que a complexidade espacial é semelhante aos outros métodos de pesquisa. De qualquer modo, nenhum outro algoritmo garante expandir menos nós que o A* search, sendo este completo e ótimo.

(4) DESCRIÇÃO DA IMPLEMENTAÇÃO

Para a implementação deste código foi utilizada a linguagem de programação Python, escolha que se deve à familiaridade e experiência que temos com esta linguagem. A utilização desta linguagem permite, também, o acesso a diversas bibliotecas disponibilizadas, que facilitam a criação do código.

No que toca ao tipo de estruturas utilizadas, escolhemos operar com arrays, listas e sets. Os arrays foram utilizados para representar os tabuleiros do jogo (uma vez que facilitam a visualização do estado do jogo e, graças à biblioteca NumPy, são mais rapidamente manipuláveis). Por sua vez, as listas e os sets foram utilizados nos algoritmos de busca, sendo as listas usadas para guardar os próximos estados do jogo a serem explorados (pois guardam esta informação mais facilmente) , e os sets usados para guardar os estados que já foram explorados (pois apresenta uma menor complexidade temporal para verificar se um dado elemento se encontra presente).

Falando, agora, da organização do código, este é constituído por: uma classe Board (que tem como atributos um estado de jogo, o número e o tipo de jogadas já realizadas, e a localização da célula vazia); uma função que trata do movimento da célula vazia para cada uma das direções possíveis; um conjunto de funções que permitem, através do estado inicial e do estado objetivo do tabuleiro, saber se existe ou não uma solução; e contém uma função para cada algoritmo de busca (IDFS, BFS, DFS, GREEDY, A*).

(5) RESULTADOS

Estratégia	Tempo (s)	Espaço (nós guardados)	Encontrou solução?	Custo (profundidade)
DFS	null	null	não	null
BFS	18.6444253921508 8	37386	sim	12
IDFS	59.0654878616333	34	sim	12
Greedy, misplaced	0.43274354934692 383	504	sim	50
Greedy, Manhattan	0.03758692741394 043	24	sim	12
A*, misplaced	0.01461625099182 1289	329	sim	12
A*, Manhattan	0.01374125480651 8555	105	sim	12

(6) CONCLUSÃO

Com os resultados demonstrados anteriormente, conseguimos observar que, pelo menos para a instância do jogo testada, quase todos os algoritmos conseguiram encontrar a solução para o puzzle, à exceção do algoritmo DFS.

É, também, possível observar que entre todos os algoritmos que chegaram à solução, apenas a greedy search com a heurística de peças fora do lugar não chegou à solução ótima (custo 12).

Quanto ao espaço e tempo, é possível concluir que existem grandes discrepâncias entre os algoritmos. Temos, por exemplo, o algoritmo BFS que ocupa mais de 30000 nós, sendo que a maior parte dos restantes não passa dos 500; e o algoritmo IDFS que demora quase 1 minuto a resolver o puzzle, enquanto que o A* demora menos de 1 segundo.

Com todas estas informações, podemos concluir que a melhor estratégia para resolver este problema é o A*, principalmente com a heurística da distância de Manhattan (dado que é a que demora menos tempo e, mesmo que não seja a que ocupe menos espaço, consegue sempre chegar a uma solução ótima).

(7) REFERÊNCIAS BIBLIOGRÁFICAS

- https://moodle.up.pt/pluginfile.php/194243/mod_resource/content/1/ebin.pub_artificial-intelligence-a-modern-approach-global-edition-4nbsped-9780134610993-1292401133-9781292401133-9781292401171.pdf
- https://en.wikipedia.org/wiki/Search_problem
- https://people.cs.pitt.edu/~milos/courses/cs1571/Lectures/Class3.pdf
- https://www.javatpoint.com/search-algorithms-in-ai
- https://www.analyticsvidhya.com/blog/2021/10/an-introduction-to-problem-solving-using-search-algorithms-for-beginners/
- https://mathworld.wolfram.com/15Puzzle.html
- https://personal.math.ubc.ca/~cass/courses/m308-02b/projects/grant/fifteen.html
- https://s3.eu-central-1.amazonaws.com/unitbv.horatiuvlad.com/facultate/inteligenta_artificiala/2015/TilesSolvability.html
- http://kociemba.org/themen/fifteen/fifteensolver.html
- http://puzzles.nigelcoldwell.co.uk/sixteen.htm
- https://www.javatpoint.com/ai-uninformed-search-algorithms
- https://www.javatpoint.com/ai-informed-search-algorithmsa
- https://www.analyticsvidhya.com/blog/2021/02/uninformed-search-algorithms-in-ai/
- https://www.geeksforgeeks.org/difference-between-informed-and-uninformed-search-in-ai/
- https://en.wikipedia.org/wiki/Depth-first_search