Inteligência Artificial – Jogo dos 15

Trabalho realizado por: Bruno Silva (up201503818) e Diogo Lourenço (up201508214)

1. Introdução

Este relatório explica alguns métodos utilizados para resolver o problema do Jogo dos 15. O jogo é constituído por um tabuleiro de 4x4 numerado de 1 a 15 em que o objetivo é chegar de uma configuração inicial aleatória a uma configuração final. Para o poder fazer move-se uma peça em branco, que computacionalmente é representada por zero, até chegar à configuração final.

Um problema de busca/procura é caracterizado por ter um estado inicial e um estado final ou objetivo. Cada estado é representado num nó e cada um é representado numa árvore de procura em que a sua estrutura é composta por pelo menos 5 campos: estado, nó pai, jogada/regra aplicada para a geração do nó, o número de nós gerados até ao nó de momento, que representa a profundidade desse nó na árvore, e o custo do caminho desde o nó raiz.

Para a resolução deste jogo os métodos utilizados dividem-se em 2 grupos. Um deles sendo, busca não guiada ou cega que utiliza os algoritmos como o BFS(*Breadth-first Search*), DFS(*Depth-first Search*) e IDFS(*Iterative Depth-first Search*). A estratégia usada por estes algoritmos é fazer uma pesquisa exaustiva, gera sucessores a partir de uma configuração inicial e a sua paragem é feita a partir da comparação da configuração final com a que estamos a tratar. O outro é busca guiada que utiliza os algoritmos como o Guloso e o A*. Estes têm em consideração a informação disponível na configuração inicial e também utiliza o conhecimento específico do problema, chamado de função heurística, para assim poder escolher a qualquer altura a solução mais eficiente.

2. Estratégias de Procura

a) Procura não guiada

Esta estratégia de procura é aplicada quando não se tem informação para tomar uma decisão em cada passo do algoritmo, tendo apenas informação sobre a configuração inicial e final. De entre estes algoritmos podemos escolher o mais indicado em termos de memória ou tempo para a resolução do nosso problema.

- **DFS**(*Depth-first Search):*

O algoritmo DFS é implementado recursivamente expandindo o nó com maior profundidade. Para a implementação utiliza-se uma pilha baseada no princípo *Last In First Out* (*LIFO*), ou seja, o último que entra na pilha é o primeiro que sai. Neste problema esta busca pode entrar em ciclo (fig.1) em que visitamos os nós por esta ordem: A-B-D-F-E-A-B-D-F-E-A-… para resolver esse problema deve-se utilizar um método de deteção de ciclos em que guarde todas as configurações já criadas para não haver repetição. No jogo dos 15 utilizou-se uma “TreeSet” para guardar todas as configurações e verificar se é ou não uma configuração repetida para depois adicionar à pilha.

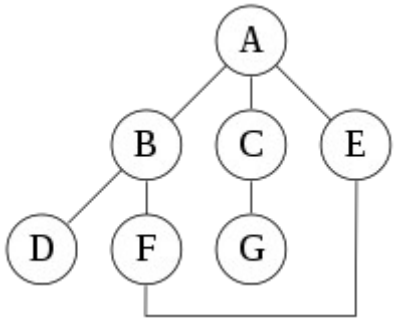

Figura 1 - Busca em profundidade

Este algoritmo não retorna sempre solução ótima por ele expandir sempre o nó com maior profundidade. A **complexidade temporal** é de $O(b^m)$ e **complexidade espacial** é de O(b*m), sendo o b o número de filhos gerados por cada nó e m a profundidade.

- **BFS**(*Breadth-first Search*):

Este algoritmo também é implementado recursivamente e explora todos os nós do mesmo nivel e só depois avança para o próximo (fig.2). Assim garante que encontra sempre solução e no nivel mais baixo. Na implementação utiliza-se uma lista em que o nó a ser explorado encontra-se no início e os descendentes são adicionados no fim. Em comparação com a DFS este algoritmo garante sempre solução ótima sem ser necessário aplicar um detetor de ciclos, pois expande todos os nós antes de passar para o nível seguinte. Mas em termos de **complexidade temporal** e **espacial** são de $O(b^d)$, sendo b o número de sucessores e d a profundidade da árvore.

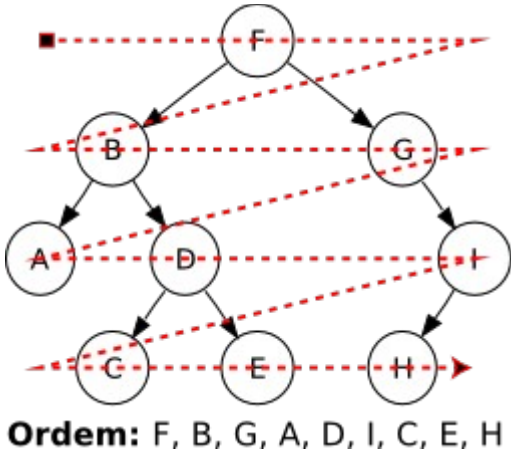

Figura 2 - Busca em largura

- **IDFS**(*Iterative Depth-first Search*):

A busca iterativa em profundidade é um caso particular da DFS, onde a profundidade vai sendo acrescentando gradualmente. Podesse ver este algoritmo como a junção das melhores características do DFS e do BFS, em que são a necessidade de pouca memória e a capacidade de examinar todo o espaço de resultados obtendo solução ótima. Isto quer dizer que só avança para o nível dois quando o nível um estiver todo explorado. A **complexidade espacial** é de O(b*d) e a **complexidade temporal** é de $O(b^d)$, sendo b o número de sucessores e d a profundidade da árvore.

b) Procura guiada

A procura guiada é uma estratégia que pode encontrar uma solução com maior eficiência do que a não guiada, pois tem em consideração o melhor nó a expandir e não se limita a expandir um nó qualquer. Para além de ter atenção à configuração inicial e final, esta tem o custo de cada nó para assim poder tomar uma melhor decisão. Esse custo é feito a partir de uma função heurística.

- **Guloso**(Greedy):

Este algoritmo expande o nó que está mais próximo da solução, com o objetivo de chegar lá mais rápido. Utiliza um função heurística que se calcula da seguinte maneira, f(n)=h(n).

Contudo tem as suas desvantagens porque só quer chegar à solução o mais rápido possível escolhendo sempre o nó com menor custo podendo muitas vezes passar ao lado da solução ótima porque esta tinha um custo maior.

A **complexidade temporal** e **espacial** são de $O(b^m)$.

- **A***:

O algoritmo A* ou A-star, é parecido com o Greedy pois tende a expandir o nó que mais perto está da solução mas com uma diferença na forma como calcula a função heurística f(n)=h(n) +g(n).

Neste algoritmo utilizou-se o mesmo método que o Greedy mas com a atenção de em cada nó ter atenção ao seu custo, número de peças fora do lugar, e também o custo do caminho que o levou até chegar a ele. Por isso o problema do melhor caminho começar por um custo mais alto não interfere.

Para ser admissível é necessário que o custo de cada nó aumente consoante o nivel em que se encontre e não diminui em relação ao nó pai.

Na resolução do Jogo dos 15 escolheu-se a heurística do número de peças fora do lugar pois garante que não aumente muito a complexidade do algoritmo e no fim está tudo no lugar pretendido.

3. Descrição da Implementação

Para a resolução desde trabalho optou-se pela linguagem Java porque dispõem de bibliotecas que facilitam a utilização de algumas estruturas de dados.

No desenvolvimento do trabalho usamos três estruturas de dados sendo elas as *Linked-Lists, Stacks* e *TreeSet.*

As *Linked-Lists* foram utilizadas nos algoritmos BFS porque permitem adicionar um elemento no fim da lista e remover o do início.

As *Stacks* foram usadas nos algoritmos em que é necessário que o último elemento a entrar seja o primeiro a sair, como é o caso dos DFS e IDFS.

Por fim, a *TreeSet* foi escolhida para a deteção de ciclos, onde guarda as configurações já visitadas e facilita a procura das mesmas com complexidade O(log n) para que os algoritmos DFS e Greedy consigam chegar a uma solução completa.

4. Resultados

Configurações inicial e final, respetivamente:

1	2	3	4
5	6	8	12
13	9	0	7
14	11	10	15

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	0

Resultados obtidos:

Estratégia	Tempo(segundos)	Espaço(nº nós gerados)	Encontrou solução?	Profundidade/Custo
DFS	Memória Esgotada			
BFS	~0,139	47032	sim	12
IDFS	~0,181	67887	sim	12
Guloso	~0,081	953	sim	12
A*	~0,061	999	sim	12

5. Comentários Finais e Conclusões

No decorrer do trabalho implementou-se vários algoritmos de procura, uns mais eficientes do que outros mas todos têm vantangens e desvantagens dependendo do problema a ser resolvido. Na nossa opinião o algoritmo DFS não é a melhor solução para este tipo de problema, tanto é que para alguns casos não conseguimos obter resposta num tempo útil. Como esta expande em profundidade se a solução estiver no último descendente do nó raiz terá de percorrer a árvore toda até achar a solução, demorando muito tempo ou esgotando a memória da máquina.

Concluímos que a melhor estratégia para o Jogo dos 15 seria o A* porque garante uma solução completa e ótima com uma função heurística admissível.

6. Referências Bibliograficas

Figura 1 - https://upload.wikimedia.org/wikipedia/commons/6/61/Graph.traversal.example.svg – retirada no dia 3 março de 2018

Figura 2 - https://pythonhelp.files.wordpress.com/2015/01/image00.png – retirada no dia 3 março de 2018

https://pythonhelp.files.wordpress.com/2015/01/image00.png

http://www.dcc.fc.up.pt/~ines/aulas/1718/IA/buscas_informadas.pdf

http://jeiks.net/wp-content/uploads/2013/05/Metodos_de_Busca.pdf – acedido a 3 março de 2018

Russell, Stuart & Norving, Peter. Artificial Intelligence, A Modern Approach. Thrid Edition.