Inteligência Artificial (CC2006) 2021/2022

Relatório de Inteligência Artificial

Beatriz Sameiro da Rocha – 202006133

Beatriz Soares Pereira - 202007190

João Fernandes Azevedo - 202008367

1. Introdução

1.1. Problemas de busca/procura

Um problema de busca/procura consiste em encontrar uma solução sem propriamente o uso de um algoritmo, mas uma especificação do que uma solução parece. Este é formalmente definido por:

- Conjunto de estados

- Estado inicial: o estado no qual o problema começa

- Estado final e teste objetivo: determina se um dado estado é o estado final. Por vezes pode existir um específico conjunto de possíveis estados finais, o teste simplesmente verifica se o estado dado é um deles. Às vezes, o objetivo é especificado por uma propriedade abstrata em vez de um conjunto de estados explicitamente enumerados.

- Função sucessor: mapeia um estado num conjunto de novos estados

- Ações: descrição das possíveis ações disponíveis ao problema.

- Transição de modelo: descrição de o que cada ação faz, ou seja, retorna o estado resultante depois de lhe ser fornecido um estado e ações especificas. Também se pode usar o termo sucessor para se referir a qualquer estado acessível de qualquer estado dado por uma única ação.

- Espaço de estado: o estado inicial, as ações e a transição de modelo implicitamente definem o estado de espaço do problema o conjunto de todos os estados acessíveis do estado inicial por qualquer sequência de ações. O estado de espaço forma um grafo no qual os nós são estados e as ligações entre os nós são ações. Um caminho no estado de espaço é uma sequência de estados conectados por uma sequência de ações.

- *Path cost*: uma função de *path cost* (custo de caminho) atribui um custo numérico a cada caminho. O problema de busca escolhe uma função custo que reflete a sua própria medida de performance. Assume-se que o custo do caminho pode ser descrito como a soma do custo de cada ação individual ao longo do caminho. Assume-se que os custos de passos são não negativos. A qualidade da solução é medida pela função de custo de caminho, e uma solução ótima tem o menor custo de caminho entre todas as soluções.

1.2. Métodos utilizados para resolver problemas de procura:

Métodos não informados de busca (busca cega):

- largura (BL)
- custo uniforme (BCU)
- profundidade (BP)
- limitada em profundidade (BLP)
- profundidade iterativa (BPI)
- bidirecional (BB)

Métodos informados: um algoritmo geral de busca que somente permite introduzir conhecimento na função de enfileiramento e utilizam conhecimento específico do problema para encontrar a solução. Nestes métodos, normalmente utiliza-se uma função de avaliação que descreve a prioridade com que um nó deve ser expandido

- Algoritmos best-first search: o "melhor" nó deve ser expandido primeiro

2. Descrição do jogo dos 15

O jogo dos 15 é um famoso quebra-cabeças composto por uma caixa de 4x4 com 15 espaços cobertos por quinze quadrados deslocáveis numerados habitualmente com números de 1 a 15, também podendo dispor de letras ou figuras, e um espaço vazio para que estas se possam movimentar. Analisando o puzzle, matemáticos chegaram à seguinte conclusão: são quase vinte e um trilhões de configurações possíveis.

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

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

Imagem 1 – Exemplo de um estado final (esquerda) e estado Inicial (direita) do jogo dos 15

Objetivo

O jogo tem a finalidade de arranjar as peças de modo a obter a disposição final pretendida. O estado final mais frequentemente usado é ordená-las de modo crescente, da esquerda para a direita e de cima para baixo.

Regras

As peças numeradas podem ser deslocadas para o espaço vazio, fazendo quantos movimentos necessários sem retirá-las. Visto que só existe um espaço vazio, apenas se pode mover uma peça à vez. Na realidade existem dois tipos de configurações iniciais possíveis: um deles permite a resolubilidade do puzzle, enquanto o outro provoca a sua insolubilidade. A configuração resolúvel pode ser obtida deslocando as peças desde a posição pretendida e efetuando passos para trás de forma aleatória fazendo sempre deslizar as peças adjacentes ao local vazio para o local vazio. A configuração insolúvel é obtida acrescentando aos deslocamentos a troca física de duas peças de posição.

Formulação do problema:

- Conjunto de estados: descreve a localização das 15 peças e do espaço vazio
- Estado inicial: qualquer estado pode ser considerado como um estado inicial
- Ações: as ações são definidas como os movimentos feitos para o espaço vazio. Dependendo de onde este se encontra, é possível fazer subconjuntos com os movimentos: esquerda, direita, cima e baixo.
- Transição de modelo: retorna o estado resultante de acordo com o estado e as ações fornecidos
- Path cost: o custo do caminho (path cost) é o número de passos no caminho, sabendo que cada passo custa 1 (step cost)

3. Estratégias de Procura

Procura não guiada (*blind* – "cega"):

A procura não guiada é tipo de algoritmo de pesquisa caracterizado por não usar conhecimento prévio para a guiar a procura da solução, ou seja, um algoritmo de força bruta. Este tipo de procura é mais lenta e têm um custo maior de memória. Existem 6 tipos de procura não guiada. Neste trabalho usamos 3 dessas pesquisas.

1. DFS – *Depth-First Search* (Profundidade):

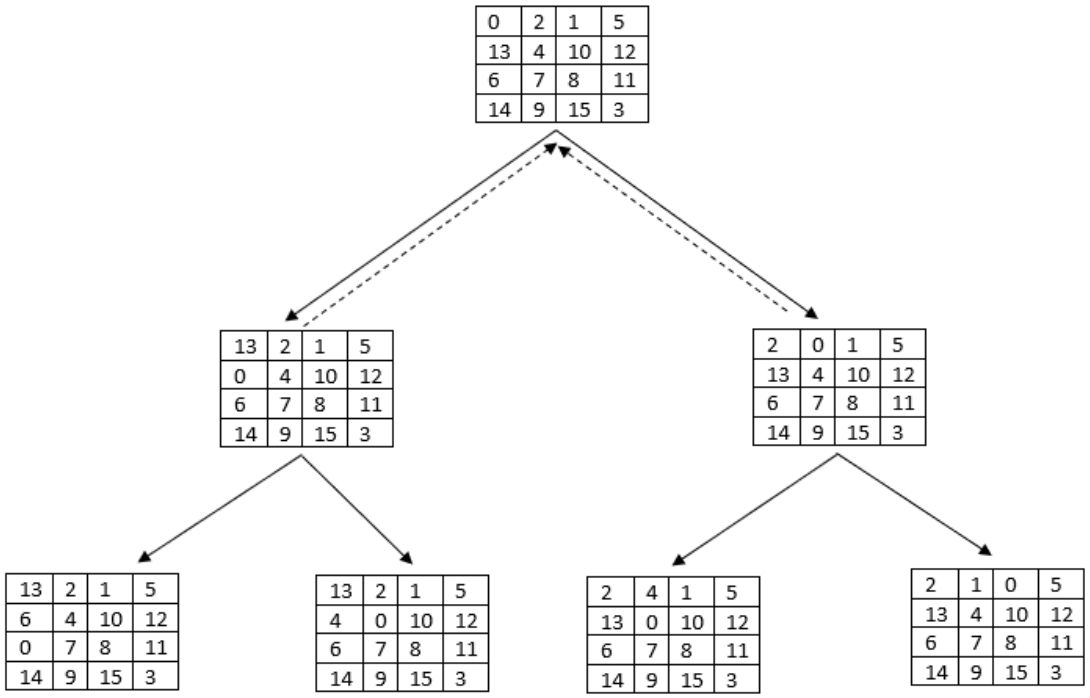
As buscas em profundidade têm complexidade temporal de O (4 ^ n) e espacial de O (4 * n). O DFS começa do estado inicial e explora até ao máximo da profundidade antes de ir para o próximo ramo.

A DFS segue um padrão:

Nó de raiz --> Nó à esquerda --> Nó à direita

A DFS é usada, normalmente, quando é preciso pesquisar uma árvore inteira.

Imaginemos que queremos ir do tabuleiro [0,2,1,5,13,4,10,12,6,7,8,11,14,9,15,3] para o [2,1,0,5,13,4,10,12,6,7,8,11,14,9,15,3] teríamos uma árvore assim:



Esquema 1

O DFS percorreria esta arvore de seguinte forma:

N1 -> N2 -> N4 -> N5 -> N3 -> N6 -> N7

2. BFS – *Breadth-First Search* (Largura):

As buscas em largura têm complexidade temporal de O (4 ^ n) e espacial de O (4 ^ n) para chegar a resposta no caminho mais curto. Assim esta começa no estado inicial e vai do próximo ao próximo até encontrar o estado do objetivo. O BFS encontra sempre a melhor solução, mas ocupa muito espaço pois têm de guardar todo o que já viu.

O BFS é usado quando é só preciso arranjar uma solução, ou o caminho mais curto para a solução.

Usando o exemplo do esquema 1 o BFS ia percorrer a árvore da seguinte forma:

N1 -> N2 -> N3 -> N4 -> N5 -> N6 -> N7

3. Busca Iterativa Limita em Profundidade:

Também conhecida como IDDFS (*Iterative deepening depth-first search*) esta busca têm complexidade temporal de O (4 ^ n) e espacial de O (4 * n). É considerada uma mistura das duas de cima, isto é, usa visitar todos os nós da mesma profundidade como o DFS, mas com um limite de profundidade que vai aumentando até chegar ao objetivo como no BFS.

O IDDFS é usado quando não se sabe a profundidade a que a solução se encontra.

Para o exemplo do esquema 1, o IDDFS começa por avaliar o N1 vê que não é solução e aumenta o nível de profundidade avaliando N2 e N3 vê que não é solução e aumenta um nível de profundidade e encontra a solução em N7.

Procura guiada:

Também denominada de pesquisa heurística, esta procura a solução com conhecimento prévio, ou pistas na forma de um heurístico. Vamos falar de 2 exemplos aqui.

1. Gulosa

Mais conhecida com *greedy* ou *best first search* deriva do BFS e do DFS usando prioridades de ambos. A pesquisa tenta procurar a arvore através do nó que ao momento parece mais próximo de chegar a da solução, o melhor nó (nó com o menor custo) a seguir é avaliado pela função heurística (f(n)=h(n)). O greedy é utilizado normalmente para otimização. O *greedy* não é a melhor forma de procura, pois é incompleta e pode levar a loops infinitas.

Com o exemplo do esquema 1 o *greedy* começaria pelo nó inicial N1 usaria a função heurística para perceber que o N3 estaria mais próximo da solução e assim chegaria a solução N7 dando nos um caminho assim: N1 -> N3 -> N7

Usamos dois tipos de heurística: somatório das peças fora do lugar e somatório das distâncias de cada peça ao seu lugar na configuração final. Estas heurísticas são consideradas heurísticas admissíveis, ou seja, nunca superestimam o custo de chegar ao objetivo, fazendo com que resulte sempre uma solução ótima (6).

2. A*:

A* junta a procura de custo uniforme (um tipo de procura não guiada, não mencionada neste trabalho) e o *greedy*. Vamos perceber como a procura de custo uniforme (UCS) funciona para depois perceber melhor a A*.

A UCS utiliza um grafo com pesos, e procura o grafo dando prioridade aos caminhos com menor valor cumulativo (1).

Assim A* junta *greedy* com UCS, utilizando os valores de custo para os nós utilizados no *greedy* e soma os valores dos grafos do UCS (que neste caso são o valor estimado até a solução) dando nos uma função heurística caracterizada por se f(n)=g(n)+h(n) sendo g(n) a função usada para calcular o custo da raiz ao nível n e h(n) a função que estima o menor custo para chegar a solução. Assim f(n) da nos uma estimativa do menor custo para chegar a solução através de n, escolhendo expandir através o nó com menor custo em f(n), se dois nós tiverem o mesmo valor de f(n) é escolhido o com o menor valore de h(n) (2).

Por isso usamos dois tipos de heurística: somatório das peças fora do lugar e somatório das distâncias de cada peça ao seu lugar na configuração final, como no *greedy*, mas este complementa o com o custo de movimento, estas heurísticas são usadas pelas razões dadas na explicação do *greedy*.

4. Descrição da implementação

Decidimos usar C++ como a nossa linguagem por as seguintes razões principais: a primeira por todo o nosso grupo já estar bem familiarizado com C, e a sintaxe de C++ ser muito parecida com C; por C++ ter várias estruturas de dados pré-programadas e prontas a utilizar; e por C++ ter programação orientada a objetos.

O nosso código foi dividido em 7 ficheiros:

Um para uma base de todos os algoritmos. É neste ficheiro que a classe principal *board* está definida, bem como os seus métodos e funções que ajudem a sua implementação.

De seguida temos 5 ficheiros que são a implementação dos algoritmos de pesquisa (cada ficheiro como o seu próprio algoritmo). Todos estes ficheiros (exceto o A*) seguem uma mesma estrutura:

```
algoritmo:
                    iniciação da(s) estrutura(s) de dados (ED);
                    introduzir o tabuleiro dado como input na ED;
                    Enquanto ED não esta vazia:
                        tirar 1 tabuleiro da lista
                        se o tabuleiro não é valido -> passa a frente
                        se o tabuleiro já foi visitado -> passa a frente
                        se o tabuleiro é o final -> quebra o ciclo
                        desenvolve no nas quatro direções
                    Escrever o caminho encontrado (desde o tabuleiro inicial ate ao
tabuleiro final)
Escrever "stats" do algoritmo (no de boards avaliados, tempo, em segundos, ate encontrar
solução e no de jogadas feitas)
```

A estrutura de dados escolhida para cada algoritmo foi baseada no quão útil é para este. Assim:

Para DFS e IDDFS é necessária uma estrutura de *First in Last Out* então usamos uma *stack*.

Para BDS precisamos uma estrutura de *First in First Out* então usamos uma *queue*.

Para as pesquisas guiadas utilizamos uma *priority queue*, dada uma função de comparação de 2 tabuleiros, nos possamos determinar o melhor tabuleiro e fazer com que ele aparece no início da estrutura.

Para além dessas estruturas, usamos também mapas para BFS e pesquisas guiadas para que possamos facilmente verificar que tabuleiros já foram vistos.

A implementação do A* é diferente de todos os outros algoritmos, a diferença esta no facto de que ao serem gerados novos tabuleiros a partir do tabuleiro pai esses novos tabuleiros vão ser logo verificados para saber se são validos, se já foram vistos ou se são a solução. Para alem disso os tabuleiros que já foram vistos, se for encontrado um novo caminho mais curto que o encontrado previamente, são retirados do mapa e adicionados a pilha.

5. Resultados

Input #1:

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

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

Algoritmos	Tempo	Tabuleiros Desenvolvidos	Solução
DFS	0.0042	125	55
IDDFS	0.0028	9	1
BFS	0.0029	2	1
Greedy: missplaced	0.0027	2	1
Greedy: manhatan	0.0028	2	1
A*: missplaced	0.0028	2	1
A*: manhatan	0.0029	2	1

Input #2:

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

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

Algoritmos	Tempo	Tabuleiros Desenvolvidos	Solução
DFS	0.3601	287689	19
IDDFS	0.0023	347	5
BFS	0.0060	150	5
Greedy: missplaced	0.0040	6	5
Greedy: manhatan	0.0037	12	5
A*: missplaced	0.0032	6	5
A*: manhatan	0.0027	9	5

Input #3:

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

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

Algoritmos	Tempo	Tabuleiros Desenvolvidos	Solução
DFS	8.9990	7278762	20
IDDFS	0.1591	152629	12
BFS	0.0608	34151	12
Greedy: missplaced	0.0081	1139	32
Greedy: manhatan	0.0030	55	12
A*: missplaced	0.0039	55	12
A*: manhatan	0.0085	215	12

Input #4:

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

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

Algoritmos	Tempo	Tabuleiros Desenvolvidos	Solução
DFS	7.9505	6480302	17
IDDFS	0.0079	2888	7
BFS	0.0042	451	7
Greedy: missplaced	0.0031	8	7
Greedy: manhatan	0.0033	18	7
A*: missplaced	0.0029	8	7
A*: manhatan	0.0032	19	7

Input #5:

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

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

Algoritmos	Tempo	Tabuleiros Desenvolvidos	Solução
DFS	14.9681	13573584	20
IDDFS	0.1651	130751	12
BFS	0.0475	30080	12
Greedy: missplaced	0.0035	34	12
Greedy: manhatan	0.0052	318	12
A*: missplaced	0.0029	22	12
A*: manhatan	0.0069	150	12

Input #6:

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

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

Algoritmos	Tempo	Tabuleiros Desenvolvidos	Solução
DFS	22.4556	22946207	20
IDDFS	3.5784	2998468	16
BFS	0.6537	414139	16
Greedy: missplaced	0.0035	22	16
Greedy: manhatan	0.0035	67	16
A*: missplaced	0.0056	105	16
A*: manhatan	0.0215	840	16

6. Comentários Finais e Conclusões

Depois de comparar os resultados obtidos a partir dos testes de input é possível perceber que as pesquisas guiadas são mais eficazes e têm um melhor desempenho que as pesquisa não guiada.

Dentro das 3 pesquisas cega a que teve pior desempenho, foi a DFS, esta nunca chegou a resposta ótima, ocupando sempre mais memoria e demorando muito mais tempo que o resto dos algoritmos, isto pode ser explicado pelo facto da DFS pesquisar todas as possibilidades sem ter alguma direção para chegar a resposta pesquisando assim todas as arvores possíveis até chegar a resposta.

As outras duas pesquisas não guiadas são bastante mais eficientes, destas duas a IDDFS é a que tem um pior desempenho, isto deve se ao facto de este avaliar todas as possibilidades com a mesma profundidade até chegar a resposta.

O BFS é a pesquisa não guiada com melhores resultados, usa menos memoria e demora menos tempo, podemos atribuir isto, ao facto de no BFS a pesquisa ser feita na direção que é mais semelhante ao objetivo.

Sobre as pesquisas guiadas podemos concluir que são claramente mais eficientes do que os algoritmos de pesquisa cego. Isto deve se ao facto de apenas desenvolverem tabuleiros que consideram bons. Porem dentro das pesquisas guiadas podemos ver que a pesquisa greedy é mais eficiente tanto em memoria como tempo do que o algoritmo A*, mas este chega sempre há solução ótima enquanto o greedy pode não chegar a essa solução, como é podemos ver no exemplo do input 3.

Para além disso conseguimos também perceber que as duas heurísticas utilizadas produzem resultados muito diferentes e que missplaced tende a ser melhor do que manhatan distance. Contudo há casos em que a heurística missplaced pode falhar, enquanto a heurística de manhatan distance tem resultados mais constantes.

7. Referências

(1) https://www.analyticsvidhya.com/blog/2021/10/an-introduction-to-problem-solving-using-search-algorithms-for-beginners/

(2) Chicago. Russell, Stuart J. (Stuart Jonathan). Artificial Intelligence: A Modern Approach.

(3) https://pt.wikipedia.org/wiki/Problema_de_busca

(4) https://pt.wikipedia.org/wiki/Algoritmo_de_busca

(5) https://pt.gadget-info.com/difference-between-informed

(6) https://www.engati.com/glossary/admissible-heuristic