TÉCNICO LISBOA

Inteligência Artificial e Sistemas de Decisão (IASD) – Laboratório 1

Ricardo Simas, 72769 Nuno Mendes, 73716 Prof. Luís Custódio

A versão de Python usada foi a 3.4. Testes executados mostraram que o código também funciona na versão 3.5. O código Python encontra-se em vários ficheiros, funcionando o ficheiro istravel.py como main. Para executar o código deve executar-se o programa com os seguintes argumentos:

$ python istravel.py input1.map input2.cli –d algoritmo

-d é um argumento opcional que acciona uma hipótese de debug que imprime no terminal a solução, isto é, o mesmo conteúdo que é impresso no ficheiro input2.sol. algoritmo corresponde ao algoritmo implementado com o qual se pretende resolver o problema. Substituir algoritmo por –dfs leva a que seja usado o Depth-first search. Substituir algoritmo por –bfs leva a que seja usado o Breath-first search. Substituir por –ucs leva a que seja usado o Uniform cost search. E finalmente substituir por –astar leva a que seja usado o A*. Quando se usa –astar a heurística usada é a implementada com recurso ao algoritmo Floyd-Warshall. Caso não seja escrito o nome de um algoritmo como último argumento, o algoritmo usado é o Uniform cost search por pré-definição.

(a) O *problem state* encontra-se representado na classe State. Os seus elementos são inicializados num *method* da classe. Cada estado é representado pelo: nó actual (cidade); transporte usado na deslocação para o nó actual; tempo a partir do qual se pode iniciar uma viagem; e custo monetário da última viagem.

Os operadores correspondem aos elementos que influenciam a viagem de um nó para outro. Os operadores são: o meio de transporte e o nó que se pretende atingir de acordo com as restrições existentes.

O *initial state* corresponde à cidade de partida do cliente; ao transporte usado para alcançar essa cidade que é NA; ao tempo a partir do qual a viagem pode ser iniciada; e ao custo monetário das ligações efectuadas que é nulo.

O *goal state* corresponde à cidade de chegada do cliente (terceiro elemento de cada linha do ficheiro .cli, com excepção da primeira linha), ao transporte usado para alcançar essa cidade, que se trata de um de quatro transportes (autocarro, avião, barco, comboio); ao tempo a partir do qual a viagem pode ser iniciada que é NA; e ao custo monetário acumulado entre as ligações usadas entre a cidade de partida e a de chegada.

O custo monetário é dado pela soma de todos os custos de cada estado. O custo de tempo é dado pela diferença entre o tempo actual e o tempo inicial.

(b) Os algoritmos *uninformed* implementados foram o Depth-first search (DFS), o Breadth-first search (BFS) e o Uniform cost search (UCS). O algoritmo *informed* implementado foi o A*.

O DFS efectua uma pesquisa em profundidade visitando todos os nós existentes em cada ramo antes de efectuar *backtracking*. O algoritmo usa uma *stack* na *fringe* que permite investigar os filhos de cada nó. Este tipo de pesquisa não considera ligações com diferentes custos não sendo por isso o mais adequado ao problema. Neste caso tem uma má performance e não é óptimo. O algoritmo foi implementado para poder ser comparado com os outros em termos de performance.

O BFS efectua uma pesquisa em largura, isto é nível a nível. Explora todos os nós vizinhos do nó actual antes de se deslocar para um desses nós vizinhos repetindo o processo. Para explorar os nós nível a nível usa uma fila de espera (*Queue*) como estrutura da *fringe*. O algoritmo é óptimo se todas as ligações entre nós tiverem o mesmo custo, pois permite encontrar o percurso com o menor número de ligações. Porém, neste problema, o percurso com o menor número de ligações não é necessariamente o melhor percurso, pois as ligações não têm todas um custo idêntico. Por esse motivo, o algoritmo não é ideal para o problema. Foi implementado para ser comparado com os outros algoritmos em termos de performance.

O UCS é uma variante do algoritmo de Djikstra. O tipo de pesquisa é semelhante ao realizado pelo BFS tendo porém em conta os custos das ligações. O UCS pesquisa todos os nós adjacentes ao nó actual, visitando o nó cujas ligações até então

usadas permitem o menor custo possível. Este algoritmo é óptimo e é o algoritmo *uninformed* mais adequado ao problema.

O A* consiste numa variante do UCS. Para além de recorrer à acumulação de custos entre ligações, considera ainda uma função heurística. Se esta função heurística for eficiente - O(1) - e admissível, representa uma grande melhoria na procura no grafo. A computação realizada para encontrar o melhor caminho entre dois nós usa a expressão f=g+h, sendo g a soma de custo entre ligações previamente utilizadas e h o valor da função heurística que se trata da distância (custo) entre o nó actualmente visitado e o nó que se pretende alcançar (*goal*). Este algoritmo é adequado ao problema pois para além de considerar custos, evita a deslocação por ligações cujos nós se apresentam uma elevada distância heurística ao nó *goal*.

(c) A parte de domínio independente encontra-se no ficheiro search.py e é a seguinte:

```
closed = set()
fringe.push([problem.getStartState()])
while True:
   if fringe.isEmpty():
      return None
   v = fringe.pop()
   if problem.isGoalState(v[-1]):
      return v
```

```
if v[-1].arrives() not in closed:
        closed.add(v[-1].arrives())
        # put everything tabbed if we used closed sets
        for child in problem.getSuccessors(v[-1]):
          aux = v + [child]
          if problem.isPlanValid(aux):
            fringe.push(aux)
```

A separação entre a parte independente e a dependente foi executada alterando a estrutura da *fringe* de cada algoritmo, antes de ser realizado o primeiro *push*, ou seja, antes de ser conhecido o estado inicial. A *fringe* do BFS contém *Stack*, a do DFS uma *Queue* e a do UCS, assim como a do A*, uma *PriorityQueue*. Estas estruturas encontram implementadas em classes de forma eficiente no ficheiro util.py.

(d) Como não é conhecida a distância "aérea" entre dois nós do grafo, conhecendo-se apenas a distância por ligações entre os nós, é complicado encontrar uma heurística admissível e que não seja tão dificil de calcular como o problema original. Neste problema encontrar uma boa heurística apresenta um nível de complexidade tão elevado quanto solucionar o próprio problema.

Uma heurística implementada no algoritmo A* consiste em usar um valor nulo como função de heurística. Esta é a pior heurística existente pois transforma o algoritmo A* no UCS.

Outra heurística implementada foi com recurso ao algoritmo Floyd-Warshall. Esta heurística consiste determinar o custo óptimo este dois quaisquer nós do grafo e é admissível. Porém é necessário fazer uma pesquisa por todos os nós o que é pouco eficiente e pode levar a que o algoritmo demore muito tempo a resolver o problema.

(e) A comparação qualitativa entre os algoritmos implementados, para os ficheiros enunciado.map e enunciado.cli, encontra-se na tabela 1.

Verifica-se que o UCS é o melhor algoritmo *uninformed*, como era esperado pois é o único dos três algoritmos que considera a duração/custo de cada ligação ao explorar nós.

Os resultados do A* e do UCS foram idênticos. Ambos os algoritmos são óptimos por isso apresentam o melhor caminho possível em termos de duração/custo.

Comprovou-se experimentalmente que número de nós visitados é superior no DFS do que nos outros algoritmos. Tal deve-se à pesquisa em profundidade do algoritmo.

Client\algorit		DFS	BFS	UCS	A*
1	Duração	3212	600	90	90
	Custo	239	14	21	21
2	Duração	4740	910	1020	1020
	Custo	300	48	47	47
4	Duração	805	5	5	5
	Custo	204	2	2	2
5	Duração	1253	942	853	853
	Custo	104	206	100	100
6	Duração	1860	1750	1860	1860
	Custo	63	64	63	63
8	Duração	600	600	600	600
	Custo	24	14	14	14

Tabela 1