E-fólio B | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Introdução à Inteligência Artificial
CÓDIGO: 21071
DOCENTE: José Coelho
A preencher pelo estudante
NOME: Bruno Ferreira
N.º DE ESTUDANTE: 2201529
CURSO: Engenharia Informática
DATA DE ENTREGA: 16 de mai. de 24

Critérios	Auto-avaliação:
Análise (1 valor)	1
Algoritmos (1 valor)	1
Resultados (2 valores)	1.5

Auto-avaliação de acordo com o avaliador: +0.1 na nota do e-fólio

Critérios de correção no enunciado.
** Procurei melhorar a minha nota relativamente ao e-folio passado. A análise do problema não é complexa. Acredito que tenha implementado de forma correta o algoritmo A* , entre gerações de estados e avaliações, acredito que possa falhar em alguns números e no melhor custo encontrado.

Relatório

Análise do problema: Temos um problema, que tem como objetivo tornar a movimentação de famílias dentro de determinados territórios menos custosa.Assim, a nossa missão é posicionar as estações de forma a conseguirmos que as famílias(população), cheguem a elas com o menor custo de movimentação possível.

Os nossos territórios, são-nos atribuídos por matrizes de várias dimensões, em que famílias encontram-se em zonas divididas por quadrados e o calculo da distância mínima do uma zona para uma dada estação é fundamental, e para este problema especifico, vamos aplicar a distância de Chebyshev.

Para resolver o problema, vamos aplicar algoritmos informados de inteligência artificial, em que a busca vai ter como base um valor retornado por uma função heurística.

Dado a natureza do nosso problema, uma função heurística admissível, seria uma função que penaliza o custo de deslocação de famílias até a estação, subtraindo a esse penalidade, o peso do conjunto de famílias dentro de um raio de distancia até 1, visto que o custo para essas distâncias é zero. Há importância em atribuir um peso ao número de famílias nesta zona, pois podem haver casos em que a densidade populacional nesse raio pode ser bastante alta, contribuindo para implementação do número mínimo de estações.

Após ter testado os algoritmos implementados na unidade curricular, na abordagem a este problema, senti-me mais confortável a trabalhar e explorar o A* visto que no meu caso produziu os melhores resultados e ainda que os melhorativos estejam no código, não estou confiante nos mesmo ainda e prefiro aguardar pelas soluções propostas pelo professor.

Código.

O nosso código está definido por dois ficheiros, um com os mapas, e coordenadas iniciadas de forma estática, [(4, 3), (3, 2), (2, 1), (1, 0)] , onde cada tupla representa uma estacão, caso haja a necessidade de se testar o código em coordenadas especificas e também, uma função geradora de posições

aleatórias tendo em conta o número de estações definidas por nós, no ficheiro main.

Pode ser executado normalmente na linha comandos, estando no diretório certo, com o comando : python3 mainf.py

E após isto, escolhemos o nome do algoritmo exatamente como queremos, e um s para sim e n para não caso queiramos usar coordenadas ou fazer uma geração aleatória.

```
Qual algoritmo quer usar? (A*, Genético, Escalada, Melhor Primeiro): A*
Quer usar coordenadas específicas? (s/n): s

Matriz 1:
0#   7   0   0   4
 0   0   0   4   0
 1   0   0   0   0
 4   4   1   0   0
 6   0   3   4   4
```

O código começa com a importação de bibliotecas especificas para resolver o nosso problema, seguidas da função def chebyshev_distance , que é usada para calcular a distancia mínima, tendo em conta movimentos horizontais, verticais e diagonais.

def travel_cost(distance): Esta função vai retornar o custo de acordo com a distancia especificada

Temos a nossa função heurística, que calcula a penalidade do custo de deslocação menos o peso de famílias num raio de distância 1 da estação atual. O peso atribuído, para evitar números negativos foi de 30%.

Na função def calculate_travel_cost(map, stations):#, calculamos e retornamos o custo médio de deslocação.

def minimize_cost(map, stations): Aqui temos a formula de custo mínimo para o nosso problema.

Depois temos, def a_star_algorithm(map, initial_stations): que trabalha conjuntamente com, def expand_and_generate_states(map, current_stations, visited, frontier, counter):
Para aplicar a lógica de expansão do algoritmo A*. E no final a função para processar matrizes. def processar_matrizes(algoritmo, instancias, num_estacoes_por_matriz, coordenadas_especificas):

Na penultima linha, 357. num_estacoes_por_matriz = [1, 1, 1, 1, 1, 1, 2, 2, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4] #quantidade de estaçoes por matriz, estao ordenadas.
Podemos definir o número de estações por matriz. Os números estão por ordem, equivalendo a primeira posição a primeira matriz.

Tabela de Resultados:

Algoritmo/Configuração	Instância	1	2	3	4	5	6	7	8	9	10
A*	Expansões	1	1	1	1	1	8	6	2	9	7
(suc3)	Gerações	1	1	1	1	1	21	104	25	712	633
	Avaliações	1	1	1	1	1	21	104	25	712	633
	Custo	1264	1134	1114	1301	1236	1237	2267	2273	3274	3293
	Tempo (msec)	0.09	0.05	0.07	0.08	0.1	1.01	10.1	1.21	121.6	62.1
	Melhor resultado	1114			1171		1233				3273

Algoritmo/Configuração	Instância	11	12	13	14	15	16	17	18	19	20
A*	Expansões	6	10	16	11	12	15	29	12	14	17
(suc3)	Gerações	435	904	1376	941	1112	1278	11951	5503	5832	6473
	Avaliações	435	904	1376	941	1112	1278	11951	5503	5832	6473
	Custo	3295	3268	3285	3298	3288	3285	4264	4285	4293	4295
	Tempo (msec)	50.8	91.9	234.71	107.3	198.4	165.68	2902.9	999.2	1604.7	1342.47
	Melhor resultado	3292		3272	3284	3273	3272			4285	