Inteligência Artificial 2023/24

Trabalho Prático

Resolução de problemas – Algoritmos de Procura

Trabalho realizado por

João Pedro Mota Baptista - a100705

João Pedro da Rocha Rodrigues - a100896

João Manuel Machado Lopes - a100594

Tiago Nuno Magalhães Teixeira - a100665

Avaliação pelos Pares

João Pedro Mota Baptista (a100705) -> -0,2

João Pedro da Rocha Rodrigues (a100896) -> -0,2

João Manuel Machado Lopes (a100594) -> +0,6

Tiago Nuno Magalhães Teixeira (a100665) -> -0,2

Índice

Introdução

A Health Planet é uma empresa de distribuição que procura utilizar os meios de entrega mais sustentáveis e ecológicos para o planeta.

Neste relatório vamos abordar a resolução de problemas através da formalização e implementação de algoritmos de procura (tanto de procura informada como não informada) para tornar esse processo de entrega mais eficiente e sustentável.

A solução terá como base um grafo que representará um conjunto de ruas de comprimentos diversos, o que será um fator responsável na procura da solução ideal.

Optamos por escolher um mapa baseado na zona do centro de Barcelos, e, para tornar o exemplo ainda mais realista, utilizamos as coordenas reais, de cada ponto, posteriormente traduzido num node, provando assim que a aplicação estaria facilmente preparada para ser usada no contexto do mundo real.

Em termos de organização de código, e para o tornar o mais modular e genérico posível, de modos que seja intuitivo para qualquer pessoa fazer alterações no mesmo no futuro, criamos as classes “Entrega”, “Estafeta”, “FuncoesAuxiliares”, “Graph”, “HealthPlanet”, “Node”, “MeioTransporte”, e a classe “Main”, responsável pela orquestração do programa.

Todas as funções do programa se encontram devidamente documentadas, para fácil análise e compreensão das mesmas.

O programa cumpre todos os requisitos estabelecidos pelo enunciado do programa, e disponibiliza ainda uma interface orientada ao utilizador, para que o mesmo se consiga servir de todos os métodos disponibilizados.

Formulação do Problema

Relativamente à formulação do problema, este poderá tratar-se de um problema de estado único caso determinemos executar o programa sem permitir ao utilizador introduzir qualquer alteração no mapa ou de contingência (não determinístico, mas totalmente acessível) caso determinemos autorizar o utilizador a fazer alterações no mapa durante a execução.

Em ambos os casos, o estado inicial corresponde ao estado em que o estafeta se encontra no armazém da empresa com as encomendas que tem de entregar, localizando-se este armazém em "Barcelos, Rua Doutor Francisco Torres".

Quanto ao estado final, este corresponde ao estado em que o estafeta realizou todas as entregas ao seu cargo e regressou ao armazém da empresa, podendo a sua travessia ter sido executada num ambiente de estado único ou de contingência.

Em relação aos operadores, o estafeta pode entregar encomendas (pré-condição: ter encomendas atribuídas ao mesmo) e deslocar-se entre ruas adjacentes no mapa (pré-condição: as ruas são adjacentes), sendo que no caso do ambiente determinístico, este tem ainda a capacidade de analisar o mapa a cada rua que avança. Importante salientar que é ainda possível atribuir veículos aos estafetas (bicicleta, mota, carro) e ainda classificações aos estafetas.

Por fim, o custo da solução não se restringe a um único valor, mas antes a um conjunto de três valores (distância total, emissões de CO2 e atraso nas entregas), a que nós olhamos para determinar qual é a melhor solução. No contexto do problema, concluímos que o objetivo da empresa é ser o mais sustentável possível, determinando então que a solução ótima é aquela em que minimizamos a soma dos atrasos das entregas, a emissão de CO2 e a distância percorrida, sendo que priorizamos atrasos menores a emissões de CO2 menores, e emissões de CO2 menores a distâncias menores. Atribuímos estas prioridades aos fatores relevantes para as soluções, pois apesar do objetivo ser determinar as travessias com menos emissões de CO2 (mais sustentáveis), como se trata de uma empresa de entregas é essencial que as encomendas cheguem a tempo.

Grafo (representação do mapa)

De seguida, apresentamos um mapa representativo do grafo introduzido de forma hardcoded no programa, com o objetivo de facilitar a visualização do mesmo e, como suporte para verificar a veracidade dos resultados devolvidos pelos diversos métodos de procura utilizados.

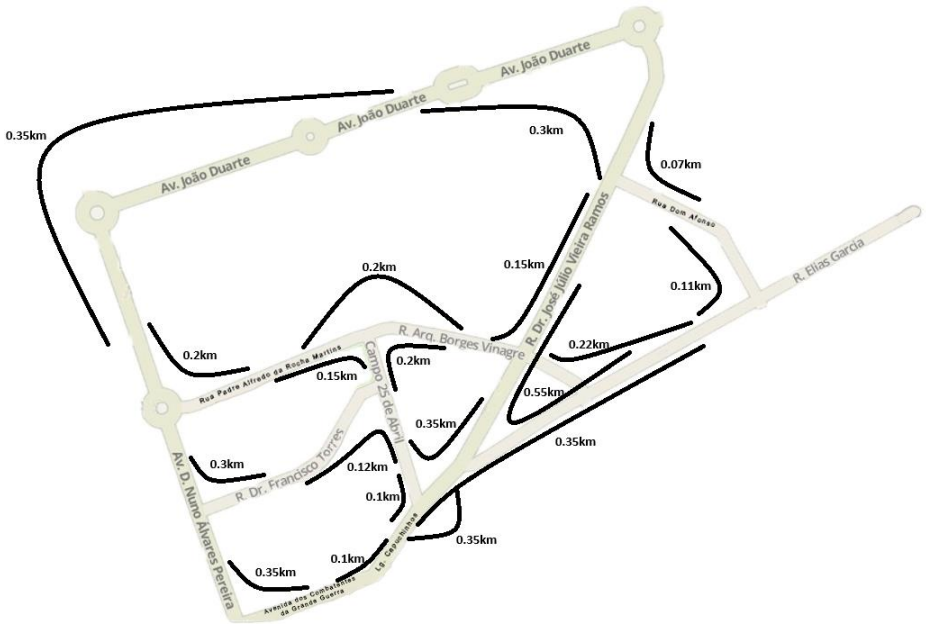

Fig. 1 – Representação do grafo usado

É importante salientar que todas as ruas observadas no mapa bem como as interseções entre as mesmas se encontram no código. Este mapa pode ainda ser expandido, reduzido ou substituído sem daí resultar qualquer erro ou mal funcionamento no programa.

Execução do Programa

Relativamente à execução da parte principal do nosso programa, sendo esta determinar a melhor rota possível para entregar um conjunto de encomendas, este divide-se em 2 partes, uma que consiste numa estratégia para quando o problema é de estado único e outra para quando o problema é de contingência, sendo que a segunda estratégia tira partido da primeira.

Referente à primeira parte, o programa procura encontrar a melhor rota possível para o estafeta realizar todas as entregas tendo em consideração todos os meios de transporte disponibilizados pela empresa, todas a formas possíveis de organizar as encomendas a entregar em viagens diferentes (por exemplo: numa viagem apenas entregar a encomenda 1 e noutra a encomenda 2, ou então entregar as duas numa viagem só) e ainda a ordem pela qual as encomendas são entregues (por exemplo: a encomenda 1 antes da 2, ou vice-versa). Para isso, o nosso programa testa todas as combinações possíveis para os fatores acima referidos (meios de transporte usados, organização de encomendas e ordem de entrega das encomendas) e vai comparando os valores das soluções e guardando sempre a melhor solução encontrada até ao momento, sendo que a melhor solução é sempre aquela que tem menor soma de atrasos das entregas das encomendas, seguida dos menor valores de CO2 emitidos e, por fim, seguida da menor distância (por esta ordem de prioridades).

Importante referir que este algoritmo por si só não calcula qualquer custo, utilizando este, portanto, algoritmos de procura em seu auxílio para calcular o custo entre pontos de passagem obrigatória, sendo estes a sede da empresa e os locais de entrega das encomendas. A partir deste custo o algoritmo principal consegue calcular os restantes valores, emissões de CO2 e atrasos, e determinar qual a melhor solução. Desta forma, a escolha do método de procura a utilizar influencia a solução devolvida pelo algoritmo principal, sendo que se os algoritmos de procura devolverem sempre a solução ótima como é o caso da procura A*, o algoritmo principal irá também devolver a solução ótima.

Por fim, relativamente à execução do programa em ambientes não determinísticos, como é o caso quando permitimos ao utilizador alterar o mapa a meio da realização das entregas, o que o agente (estafeta) faz é alternar procura e execução, ou seja, a cada rua que avança verifica se consegue entregar alguma encomenda na rua em que se encontra e volta a calcular qual o melhor trajeto para concluir a sua travessia, só que em vez de iniciar esse cálculo na sede da empresa, inicia-o a partir da sua localização atual. Para fazer esta procura em cada rua que avança, o estafeta volta a executar o algoritmo mencionado em cima, da parte 1, só que para a posição em que se encontra, assegurando a solução ótima (se o algoritmo de procura devolver a solução ótima) para cada estado do grafo.

Consideramos pertinente referir que o nosso programa é escalável em relação às estratégias de procura utilizadas, sendo muito simples introduzir e utilizar uma nova estratégia, visto que estas não são o core do nosso programa, mas sim apenas funções utlizadas pelo algoritmo principal.

De seguida, apresentamos as várias estratégias de procura implementadas no nosso trabalho, sendo que posteriormente apresentaremos resultados de cada uma face a um mesmo input no nosso programa, como termo de comparação.

Pesquisas não informadas

DFS

A pesquisa em profundidade (DFS) é um algoritmo de busca não informada que explora um grafo seguindo o máximo possível ao longo de um ramo antes de retroceder. Começa num nó inicial e visita todos os seus vizinhos antes de avançar para os vizinhos destes. O algoritmo continua a explorar até atingir o nó de destino ou esgotar todas as opções. Utiliza uma abordagem de pilha, empilhando os nós a serem explorados, o que permite que o algoritmo retroceda quando necessário.

IDDFS

O Algoritmo de Pesquisa em Profundidade Iterativa com Limite (IDDFS) é uma versão melhorada da Pesquisa em Profundidade (DFS) que realiza buscas em profundidade com limites de profundidade crescentes até encontrar o objetivo. Começa com uma profundidade limite mínima (1) e aumenta progressivamente até atingir o limite máximo especificado. Isso permite uma busca em profundidade eficiente, evitando a expansão desnecessária de ramos do grafo. O algoritmo utiliza uma abordagem recursiva e mantém a iformação relativa aos nós visitados para evitar ciclos. Retorna um caminho e o custo total se encontrar um caminho dentro do limite de profundidade, caso contrário, retorna None.

BFS

O Algoritmo de Pesquisa em Largura (BFS) é uma técnica que percorre ou pesquisa estruturas de dados de árvores ou grafos. Começa no nó raiz e explora todos os nós vizinhos na profundidade atual antes de passar para os nós no próximo nível de profundidade. A implementação utiliza uma fila para armazenar os nós a serem visitados. Inicialmente, o nó de partida é colocado na fila e marcado como visitado. O algoritmo continua até a fila estar vazia ou até encontrar o nó de destino. Para cada nó atual, seus vizinhos não visitados são postos na fila e marcados como visitados. O caminho é reconstruído retrocedendo do nó de destino até o nó de partida usando um dicionário de pais. Retorna um tuplo contendo o caminho e o custo total se um caminho for encontrado, ou None se nenhum caminho existir.

UCS

O Algoritmo de Busca de Custo Uniforme (UCS) explora o grafo de maneira em largura, mas considerando o menor custo de cada caminho. Ele mantém uma lista aberta de nós a serem explorados, ordenados pelo custo cumulativo. O algoritmo continua até que a lista aberta esteja vazia ou o nó alvo seja alcançado. Durante a exploração, o algoritmo mantém o estado dos pais de cada nó para reconstruir o caminho posteriormente. A implementação utiliza uma lista aberta, uma lista fechada e um dicionário de pais. Retorna um tuplo contendo o caminho mais curto de start a end e o seu custo, ou None se nenhum caminho existir.

Dijsktra

O algoritmo de Dijkstra inicia a busca a partir do nó de origem, explorando os vizinhos de forma ordenada pelo custo acumulado. O algoritmo mantém um registo dos custos acumulados e dos pais de cada nó no caminho mais curto. Esse processo continua até que o nó de destino seja alcançado ou todos os nós sejam explorados. O caminho mais curto é então reconstruído a partir dos pais guardados. A utilização de uma fila de prioridade contribui para uma busca eficiente, considerando os menores custos primeiro. Tal como os outros métodos, este retorna o caminho que encontrou, e o seu custo total, ou None, caso não tenha encontrado nenhum caminho.

Floyd Warshall

O algoritmo de Floyd-Warshall utiliza uma abordagem dinâmica. Inicializa uma matriz de distâncias com os pesos das arestas existentes e atualiza-a iterativamente para considerar caminhos intermediários. O caminho e o custo total entre os nós de partida e destino são recuperados a partir das matrizes geradas durante o processo. O resultado é um tuplo contendo o caminho e o custo total se uma solução for encontrada, ou None caso contrário.

Bellman – Ford

O algoritmo de Bellman-Ford foi feito para funcionar, mesmo quando existem arestas com pesos negativos. Ele emprega a técnica de relaxamento, iterativamente percorrendo todas as arestas para atualizar os custos mínimos. O algoritmo também verifica a existência de ciclos negativos no grafo. Após as iterações, reconstrói o caminho mais curto a partir dos nós pai, fornecendo o caminho e seu custo total.

Random Walk

O algoritmo de Passeio Aleatório escolhe aleatoriamente vizinhos não visitados até alcançar o nó de destino ou não haver mais opções. Retorna o caminho e o custo total arredondado se encontrar uma solução, ou None caso contrário. Este algoritmo, pode, naturalmente, não devolver uma solução ótima.

Pesquisas informadas

A estrela (A*)

O algoritmo A* utiliza listas abertas e fechadas para otimizar a busca em grafos. A lista aberta armazena os nós a serem explorados, priorizando os de menor custo estimado. A lista fechada guarda os nós já explorados, evitando revisões desnecessárias. O algoritmo equilibra a exploração de novas possibilidades e a reutilização de informações previamente analisadas. Iterativamente, seleciona e expande nós, eficientemente encontrando o caminho mais curto. Retorna um tuplo com o caminho (lista de nós) e o custo. Se o caminho não existir, retorna None.

IDA* (Iterative Deepening A star)

O algoritmo IDA* realiza uma busca com aprofundamento iterativo para encontrar o caminho ótimo utilizando um limite de profundidade.

O algoritmo base funciona de iguala forma quando comparado com o A estrela, pelo que expande os nós considerando o custo do caminho desde o nó de partida e o custo estimado até o nó de destino. A procura continua a aumentar gradualmente o limite de profundidade até encontrar um caminho ou determinar que nenhum caminho existe. Se encontrar um caminho, retorna um tuplo com o caminho e seu custo; se não, retorna None.

Greedy (gulosa)

O algoritmo de Pesquisa Gulosa escolhe sempre expandir o nó que parece estar mais próximo do destino. A pesquisa continua a expandir os vizinhos do nó atual e selciona aquele com o menor valor heurístico. Se encontrar um caminho, retorna um tuplo com o caminho como uma lista de nós e o custo total do caminho; se não, retorna None. O algoritmo é eficiente na exploração, mas pode não garantir a solução mais ótima.

Interface

Vamos apresentar como está organizada a interface utilizada durante os testes, ou seja, as escolhas que o utilizador terá para executar as diversas funcionalidades do programa. Nesta estão compreendidas todas as opções necessárias para o utilizador poder executar qualquer função do nosso programa.

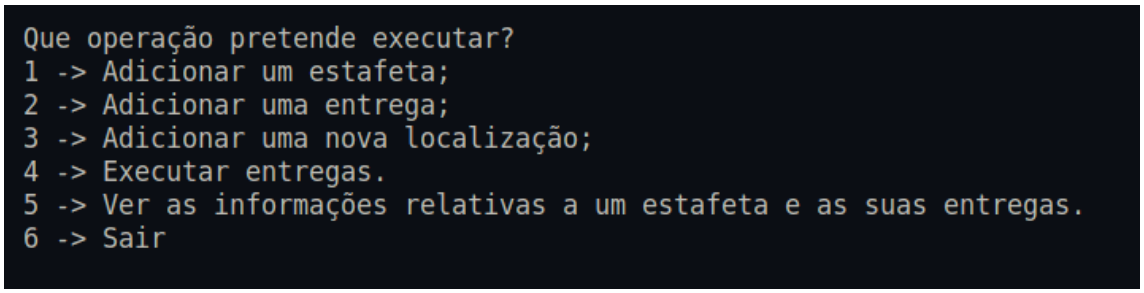

Fig. 2 - Menu Inicial

Na figura 2 podemos encontrar aquilo que é o menu inicial, com opções de adicionar novos elementos, de obter informações de estafetas e de execução de entregas. No contexto do trabalho prático vamos aprofundar mais a execução de entregas, visto que é nesta que serão aplicados os algoritmos de procura.

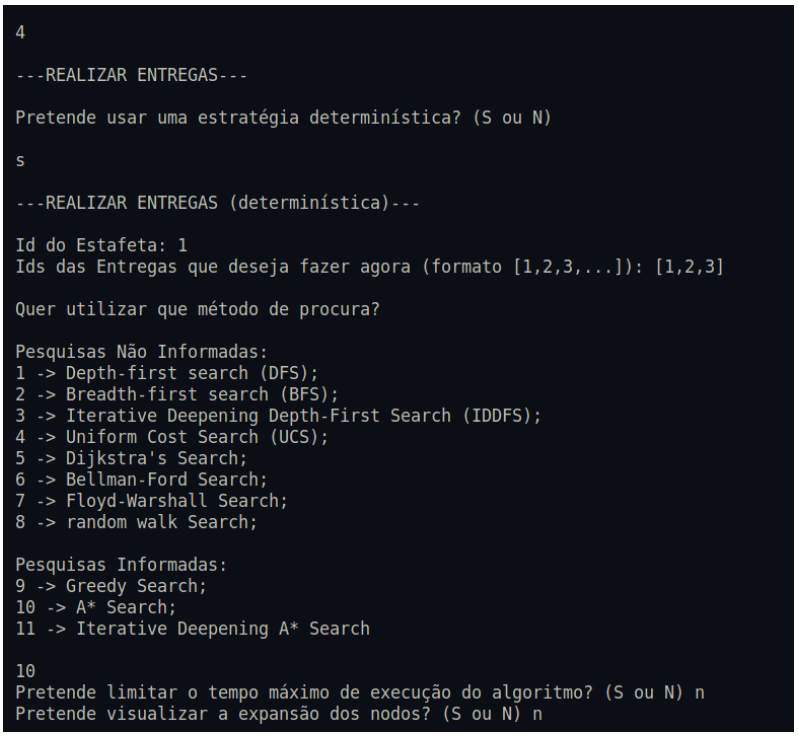

Fig. 3 – Pedido de Execução de Entregas

Na figura 3 encontramos a sequência de inputs requisitados pelo programa ao utilizador para executar o seu pedido de realizar entregas (opção 4 no menu da figura 2) bem como as opções que este fornece ao utilizador, nomeadamente, qual a estratégia de procura que pretende utilizar na determinação do melhor caminho para realizar as entregas.

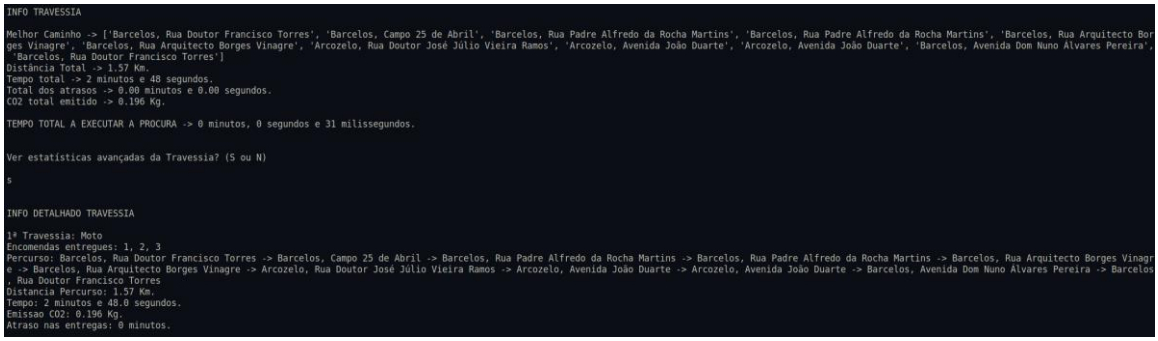

Fig. 4 - Apresentação de Resultados

Após o programa calcular a solução, retornará algumas informações sobre a mesma. Na figura 4 encontramos essas informações, onde conseguimos visualizar a distância total em Km da melhor solução encontrada, a emissão de CO2, a soma total dos atrasos das entregas, o caminho escolhido, bem como o tempo que demorou a encontrar a melhor solução.

Importante referir que após apresentar estes resultados é ainda possível aceder a informações mais detalhadas do melhor caminho (exemplo na figura 16), como por exemplo, as viagens que foram feitas, as encomendas que o estafeta levou em cada viagem, os veículos usados, entre outras informações pertinentes.

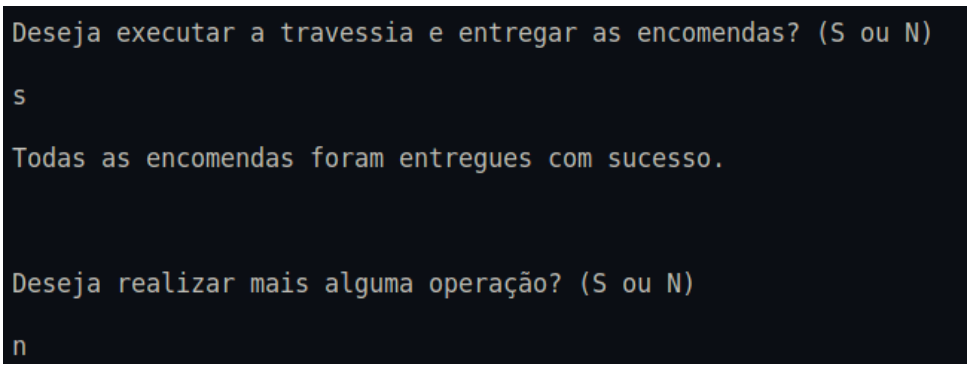

Fig. 5 - Entrega das Encomendas

Por fim, é questionado ao usuário se desejará realizar a travessia para a entrega das encomendas e, após responder a essa questão, o programa pergunta se desejará realizar mais alguma operação, caso em que se responder afirmativamente, será novamente apresentado o menu da figura 2.

Testes e Resultados Obtidos

Como prometido na secção “Execução do Programa” realizamos vários testes de forma a mostrar o comportamento do nosso algoritmo principal utilizando diferentes estratégias de procura face a um mesmo cenário, determinando assim quais as mais eficazes em obter a solução ótima, bem como o tempo que cada uma demora a executar. De seguida, apresentamos os vários resultados ao executar o algoritmo que determina o melhor percurso para entregar todas as entregas utilizando diferentes funções de procura, para o mesmo estafeta e as mesmas encomendas a entregar (id do estafeta = 5 e ids das encomendas a entregar = [12,13,14,16,16,17]). A comparação dos algoritmos é feita no final de apresentar todos os resultados das diferentes estratégias. (Recomendamos ampliar o ficheiro de forma a conseguir ver os resultados)

Algoritmo DFS

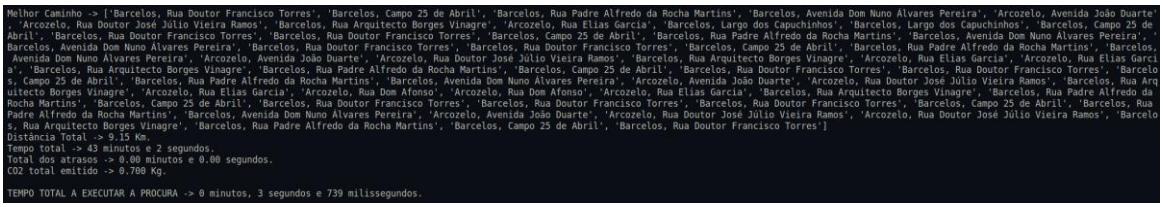

Fig. 6 - DFS

Algoritmo BFS

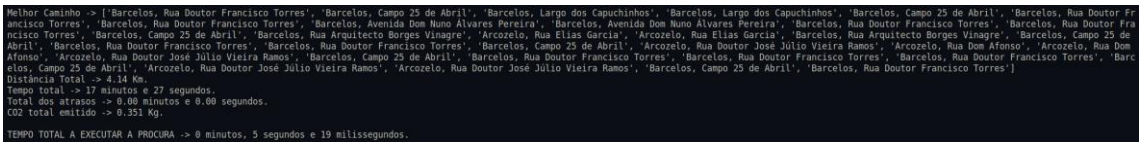

Fig. 7 - BFS

Algoritmo IDDFS

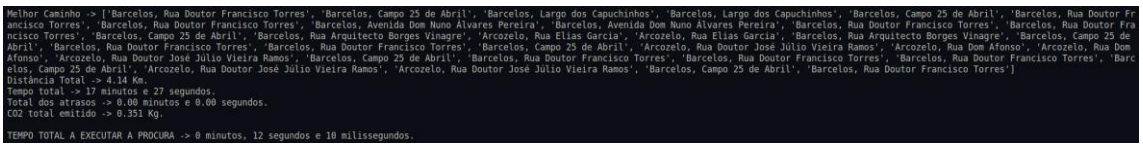

Fig. 8 - IDDFS

Algoritmo UCS

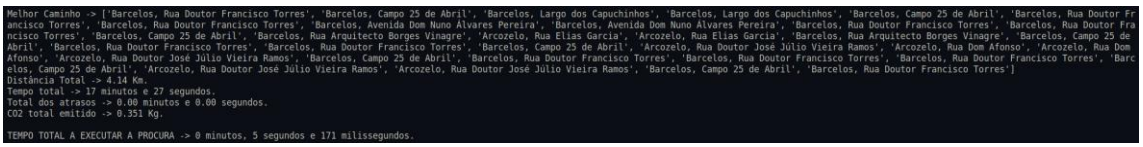

Fig. 9 - UCS

Algoritmo Dijkstra

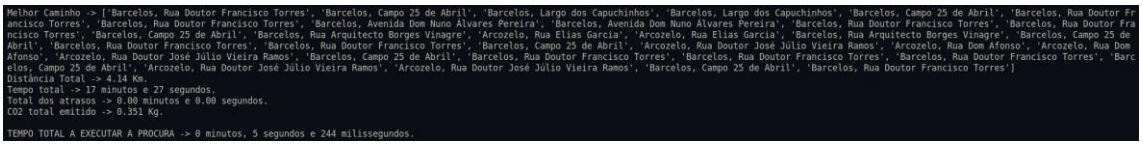

Fig. 10 - Dijkstra

Algoritmo Bellman-Ford

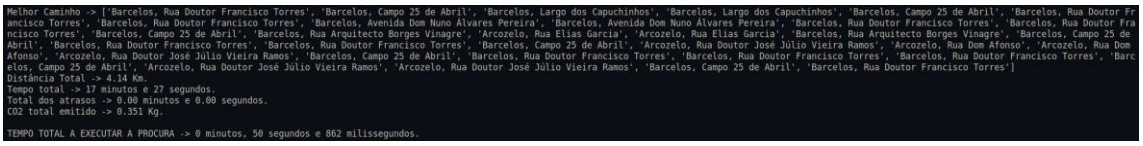

Fig. 11 – Bellman-Ford

Algoritmo Floyd-Warshall

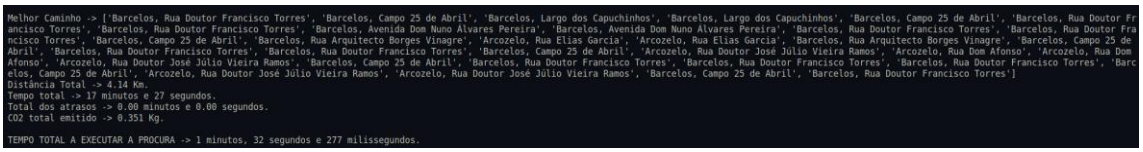

Fig. 12 – Floyd-Warshall

Algoritmo Greedy

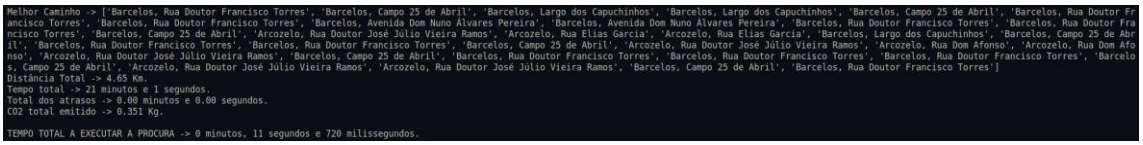

Fig. 13 – Greedy

Algoritmo A*

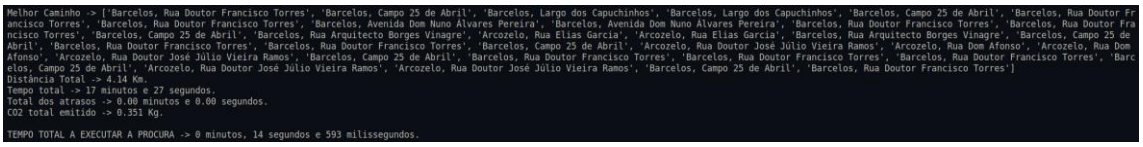

Fig. 14 – A*

Algoritmo Iterative Deepening A*

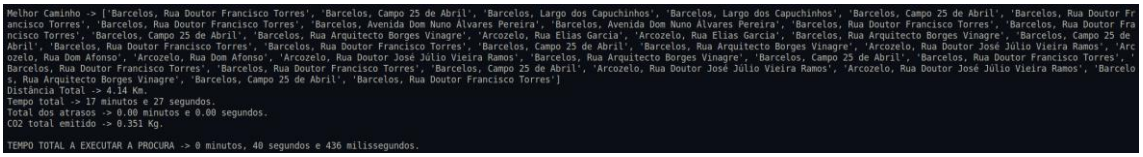

Fig. 15 –Iterative Deepening A*

Análise Resultados

A partir dos resultados acima apresentados e ao comparar os diversos valores obtidos podemos concluir que houve discrepância nos valores fornecidos por alguns métodos de procura, sendo que uns encontraram resultados mais satisfatórios que outros. Neste caso, apesar dos caminhos retornados poderem ser diferentes, encontraram a solução ótima os algoritmos A*, IDA*, Floyd-Warshall, Bellman-Ford, Dijkstra, UCS, IDDFS e BFS. Contudo, é importante de notar que alguns destes algoritmos podem não retornar sempre a solução ótima para outros inputs, visto a determinação da solução ótima para alguns estar dependente de fatores como a organização do grafo, a origem e o destino, entre outros. Assim, quando o objetivo for obter mesmo a solução ótima, recomendamos utilizar, por exemplo, o algoritmo A*, que devolve sempre a solução ótima, sendo um bom termo de comparação para os resultados obtidos com os outros algoritmos.

De seguida, podemos ainda observar as estatísticas detalhadas de uma travessia executada pelo estafeta 5 com o intuito de entregar as encomendas de ids 12, 13, 14, 15, 16 e 17 utilizando o algoritmo A*. Na imagem observamos que pela aplicação do algoritmo a melhor solução resultaria de realizar 5 viagens, 3 de bicicleta, 1 de mota e 1 de carro, com as distribuições de entregas observadas na figura. Relativamente ao custo total, emissões de CO2 totais e soma dos tempos de atraso estes encontrar-se-iam apresentados acima deste menu (semelhante à imagem 4) e resultariam da soma dos resultados das 5 viagens apresentadas. Realçamos apenas que os resultados obtidos por quaisquer outros algoritmos de procura são obrigatoriamente piores ou tão bons quanto este.

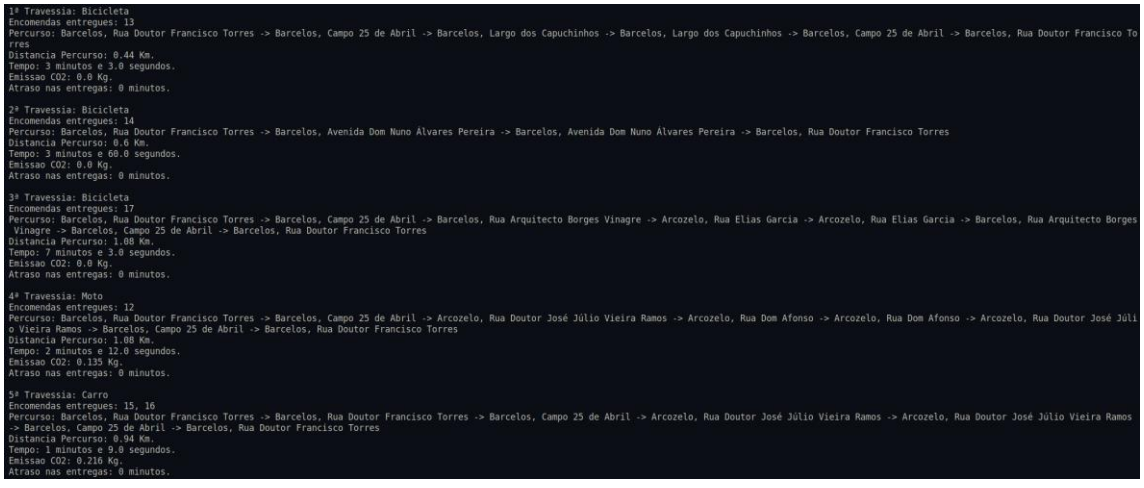

Fig. 16 - 1Travessias da A*

Conclusão

Dada a conclusão deste trabalho realizado no âmbito da disciplina de Inteligência Artificial, acreditamos ter retido e compreendido, com sucesso, todos os algoritmos de procura mencionados, desde as ideias por detrás da sua criação até ao seu modo de funcionamento e implementação.

Em suma, consideramos ter respondido e correspondido a todos os objetivos e metas estabelecidas no trabalho prático, sendo que o objetivo principal era desenvolver um programa capaz de retribuir os caminhos mais ecológicos quer para ambientes totalmente observáveis determinísticos quer para ambientes totalmente observáveis e não determinísticos e, sem dúvida, alcançamos esse objetivo, pelo menos para algoritmos de procura como o A*, que retornam o valor ótimo. Em adição, desenvolvemos ainda uma interface para o utilizador muito intuitiva, de forma a permitir ao mesmo inserir os seus próprios mapas e testar os algoritmos quer de forma recreativa quer para fazer o seu próprio sistema de entregas.