Trabalho realizado por:

- **Bruno Silva - A100828**
- **Ema Martins - A97678**
- **Henrique Malheiro – A97455**
- **Manuel Serrano - A100825**
- **Marta Gonçalves - A100593**

Universidade do Minho
Departamento de Informática

Inteligência Artificial

Instrumento de Avaliação em Grupo

Tema: Resolução de Problemas - Algoritmos de Procura

Ano letivo 2022/2023

Índice

Introdução

Este trabalho tem como finalidade a implementação de uma empresa de distribuição de encomendas, a Health Planet, que tem como principal objetivo a entrega de encomendas da forma mais sustentável possível. A Health Planet é composta por vários estafetas que realizam encomendas através de diferentes meios de transportes, sendo estes bicicletas, motos ou carros. Os estafetas têm associado um ranking. Cada cliente deverá poder classificar a entrega do estafeta entre 0 e 5. Caso o estafeta não cumpra com o prazo estipulado para a entrega da encomenda, sofrerá uma penalização no seu ranking. Um cliente pode definir o tempo máximo para entrega de uma encomenda. Uma encomenda tem de ser, pelo menos, caracterizada pelo seu peso e volume e o preço do serviço de entrega deverá ter em conta, no mínimo, a encomenda, o prazo de entrega e o meio de transporte.

Relativamente aos diferentes meios de transporte, temos de ter em atenção que as bicicletas podem transportar no máximo 5 Kg e têm uma velocidade média de 10Km/h sendo esta decrementada 0.6 Km/h por cada Kg transportado. Já as motos podem transportar encomendas até 20Kg a velocidades médias de 35Km/h que diminuem 0.5 Km/h por cada Kg transportado. Por sua vez, os carros podem levar no máximo 100Kg e têm uma velocidade máxima de 50Km/h que é decrementada em 0.1Km/h por cada Kg transportado.

Pretende-se que o programa seja capaz de gerar circuitos que cobrem um determinado território constituído por um conjunto de ruas ou freguesias.

As nossas decisões

O nosso programa cobre algumas das freguesias de Braga, tendo-as representadas através de um grafo, sendo possível fazer entregas nas diferentes freguesias do concelho.

No nosso trabalho consideramos que os estafetas apenas podem ter um meio de transporte associado e que sempre que vão realizar uma encomenda, após a entrega voltam necessariamente ao armazém. No entanto, estes podem ter uma lista com encomendas futuras. Assim, quando um estafeta chegar ao armazém, pode já ter associado a si outras encomendas para realizar.

Além das condições mencionadas na introdução, consideramos que o meio de transporte tem associado uma variável que indica se este é elétrico ou não.

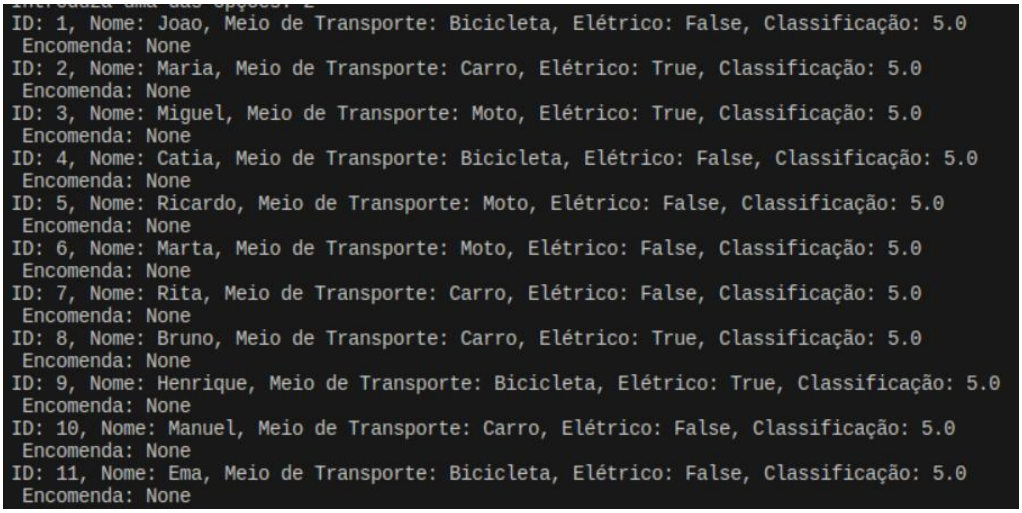
O tempo de entrega de cada encomenda é influenciado por diversos fatores como o percurso a percorrer, a distância entre as freguesias desse percurso, o tipo de estrada que liga essas freguesias, o trânsito que se verifica em cada uma dessas estradas, a altura do dia e a meteorologia.

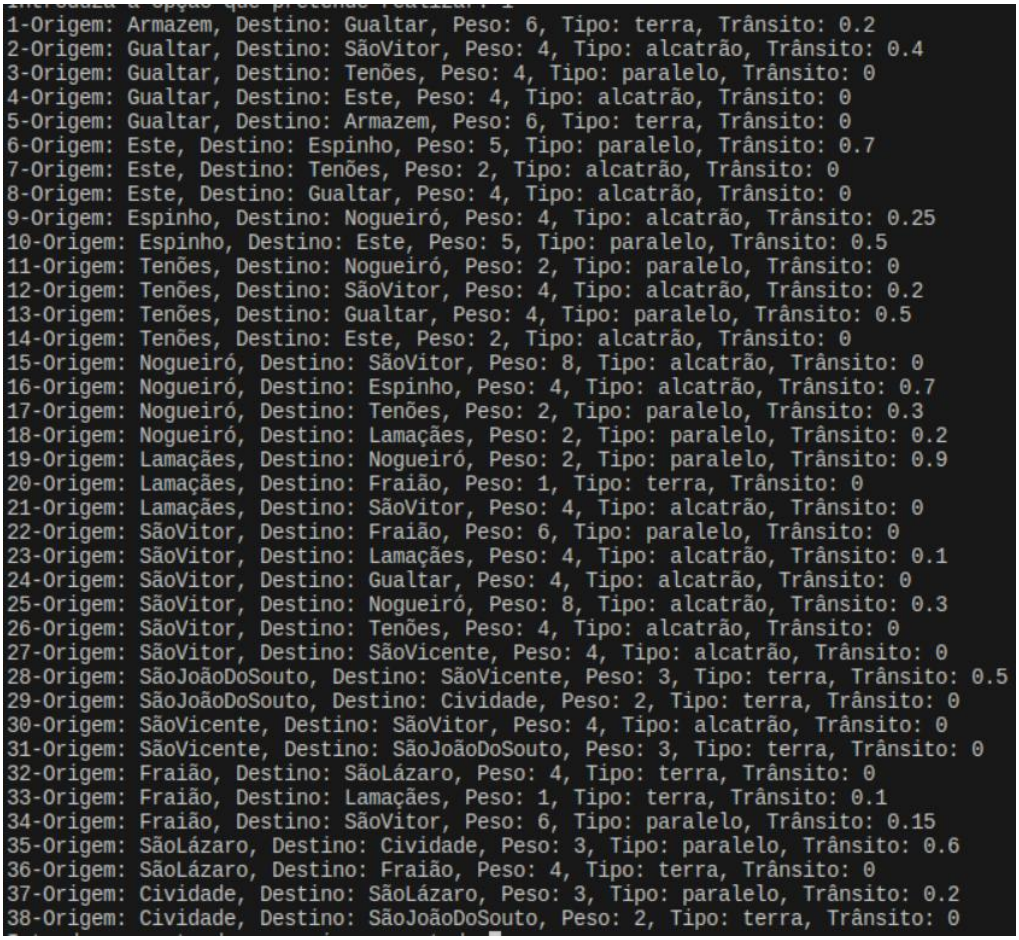
No cálculo do preço do serviço, tivemos em conta não só o prazo de entrega e meio de transporte como também o peso e volume da encomenda.

Introduzimos ainda no programa um tempo virtual que permite simular o decorrer do mesmo, de forma a perceber melhor o seu funcionamento (1 minuto no programa corresponde a 1 segundo real).

Formulação do problema como um problema de procura

Inicialmente, a Health Planet pode ser representada através de um grafo que contém as diferentes as freguesias nas quais vamos realizar as entregas. Além disso, todos os estafetas começam no Armazém e sem encomendas para realizar.





A realização de uma encomenda pode ser definida como um problema de procura. Este é um problema determinístico pois a qualquer momento podem conhecer-se todas as informações sobre o mesmo e as ações realizadas terão efeitos conhecidos. Porém, o trajeto e, consequentemente, o tempo demorado podem sofrer alterações, caso o utilizador decida modificar certos parâmetros ao longo do tempo de entrega.

O estado inicial é a atribuição da encomenda a um estafeta que tenha disponibilidade para a entregar dentro do tempo estipulado pelo utilizador.

O estado objetivo é que a encomenda pedida seja entregue, caso seja possível fazê-lo, pelo estafeta a quem foi atribuída e ele regresse ao Armazém.

Para este problema podemos considerar vários operadores:

- Avançar o tempo
  - Pré-condição: O tempo estar parado
  - Efeitos: O tempo vai passando e o estafeta vai-se aproximando do estado objetivo
- Parar o tempo
  - Pré-condição: O tempo estar a avançar
  - Efeitos: O tempo para de avançar e o estafeta fica estagnado na posição em que se encontra
- Cortar estrada
  - Pré-condição: Existir pelo menos outra estrada a sair da origem e pelo menos outra estrada a chegar ao destino
  - Efeitos: O caminho a percorrer, as velocidades médias e os tempos têm de ser recalculados
- Repor estrada
  - Pré-condição: Existirem estradas que foram cortadas
  - Efeitos: O caminho a percorrer, as velocidades médias e os tempos têm de ser recalculados
- Alterar altura do dia
  - Pré-condição: -
  - Efeitos: O caminho a percorrer, as velocidades médias e os tempos têm de ser recalculados
- Alterar meteorologia
  - Pré-condição: -
  - Efeitos: O caminho a percorrer, as velocidades médias e os tempos têm de ser recalculados

- Alterar trânsito de uma estrada
    - Pré-condição: O valor pretendido para o trânsito da estrada deve estar entre 0 e 0.9
    - Efeitos: O caminho a percorrer, as velocidades médias e os tempos têm de ser recalculados
- Atrasar estafeta
    - Pré-condição: Existir pelo menos um estafeta
    - Efeitos: O tempo deixar de afetar este estafeta até ao fim do atraso, ficando o estafeta estagnado no ponto em que se encontra

O custo da solução é o tempo que a encomenda demora a ser realizada. Este tempo tem em conta diversos parâmetros como o percurso a percorrer, a distância entre as freguesias desse percurso, o tipo de estrada que as liga, o trânsito que se verifica em cada uma dessas estradas, a altura do dia e a meteorologia.

Ficheiros

cria_grafos.py

No ficheiro cria_grafos.py temos presentes as funções cria_grafo, str_arestas_grafo e mover_aresta_entre_grafos.

A primeira função cria o grafo com o qual vamos trabalhar durante a execução do programa, com o auxílio da biblioteca networkx. São assim definidos os vértices, que representam freguesias, e as arestas, que representam as estradas que ligam essas freguesias. As arestas têm um peso que corresponde à distância entre as duas freguesias que ligam, o tipo de estrada que pode ser terra, alcatrão ou paralelo e o trânsito que indica a proporção de trânsito num dado momento (valor entre 0 e 0.9 onde 0 indica a inexistência de trânsito). As estradas contêm ainda uma heurística que é dada pela distância em linha reta entre duas freguesias.

A segunda função, dado um grafo, escreve no terminal todas as estradas desse mesmo grafo, numerando-as.

A última função, dados dois grafos, retira uma aresta do primeiro e coloca-a no segundo, utilizada para cortar/repor estradas.

povoar.py

O ficheiro povoar.py contém a função povoa_estafetas que cria um conjunto de estafetas e os adiciona, facilitando a utilização do programa pois deixa de ser necessário estar a criar os estafetas um a um sempre que se pretende correr o programa.

algoritmos_procura.py

Neste ficheiro, encontram-se as funções que realizam todos os algoritmos de procura que utilizamos no trabalho, sendo estas dijkstra, procura_em_profundidade, bfs, bidirecional_search, interative_depending_dfs, greedy_shortest_path e algoritmoAEstrela.

A função dijkstra corresponde ao algoritmo de procura de custo uniforme. Assim sendo, expande gradualmente para os nós vizinhos por ordem crescente de custo, mantendo uma fila com prioridades baseada no custo acumulado.

A função procura_em_profundidade corresponde ao algoritmo de procura em profundidade (DFS – Depth First Search). Este começa pela raiz e expande um ramo até ao final. Caso não encontre o destino, retrocede e aborda um novo ramo.

O bfs realiza a procura em largura (BFS – Breadth First Search). Assim, o algoritmo começa por explorar a raiz e visita todos os vizinhos do nível em que se encontra antes de passar para o próximo nível.

A função bidirecional_search implementa a procura bidirecional. Este algoritmo realiza simultaneamente duas buscas, uma do estado inicial para o estado e final e outra do estado final para o inicial. Retorna o caminho quando encontra um estado comum.

O iterative_depending_dfs realiza uma procura em profundidade mas só até um certo nível. Inicialmente esse nível é 1. Caso não encontre o caminho no nível em que se encontra, então passa para o nível seguinte.

O greedy_shortest_path implementa a procura gulosa onde é expandido sempre o nó com menor heurística.

A função algoritmoAEstrela implementa a procura A*. Este algoritmo expande o nó cujo custo acumulado somado com a heurística tenha o menor valor. Caso encontre uma melhor solução enquanto está a iterar, pode voltar atrás.

Estes algoritmos foram desenvolvidos com algumas adaptações àquilo que era pretendido neste trabalho, de forma a devolverem o caminho da origem até ao destino, o custo do deslocamento que corresponde à distância do caminho, e ainda a lista com as freguesias pela ordem pela qual foram visitadas pelo algoritmo.

Isto permitiu, além das funcionalidades necessárias à entrega das encomendas, que nós fossemos tendo uma ideia da forma como cada algoritmo percorre um grafo e percebêssemos melhor o que os distingue e os casos em que cada um deles funciona melhor. No tópico Resultados explicamos as conclusões que tiramos acerca de cada um dos algoritmos.

encomenda.py

Cada encomenda efetuada é caracterizada por um identificador único, pelo peso, pelo volume e pelo tempo de entrega máximo escolhido pelo utilizador.

Quando a encomenda é criada, o primeiro passo é atribuí-la a um estafeta que a consiga entregar no tempo pretendido. Se não for possível entregá-la, ela é automaticamente apagada, caso contrário a encomenda é atualizada para que apresente o id do estafeta que a irá transportar e um preço, calculado na função calcula_preco, que depende das características da encomenda (peso, volume e urgência do tempo máximo de entrega) e ainda das características da entrega (meio de transporte utilizado, se é elétrico ou não e distância a percorrer). O preço é tão maior quanto maiores forem os valores do peso e volume da encomenda e da distância a percorrer. Quanto menor for o tempo máximo estabelecido para a entrega, maior será o seu preço. Além disso, existem aumentos de custo quando é selecionado um veículo de um tipo mais poluente.

Se a encomenda for atualizada, isto é, se for possível entregá-la e já estiver decidido o estafeta que o vai fazer, surgem novos atributos relacionados com a entrega da encomenda, como o destino que é a freguesia para onde o estafeta deve levar a encomenda, o caminho que é uma lista das freguesias por onde o estafeta deve passar no seu trajeto de ida e volta, as velocidades_medias calculadas para cada etapa do caminho a percorrer, o tempo_total_viagem que corresponde ao tempo que demora a viagem de ida e volta e o tempo_previsto que foi o tempo calculado para a entrega da encomenda, tendo em conta a disponibilidade do estafeta e uma margem de 2 minutos por encomenda (que inclui a encomenda_atual do estafeta e todas as que se encontram na sua lista de espera).

Existem ainda alguns parâmetros que vão sendo alterados com o decorrer do tempo, como é o caso do tempo_que_percorreu que corresponde ao tempo que já passou desde que a encomenda foi pedida, logo vai aumentando sempre uma unidade com o decorrer do tempo até que a encomenda chegue ao destino. Este valor é posteriormente comparado com o tempo_previsto para verificar a ocorrência de atrasos. Temos ainda o tempo_transporte que se inicia com o valor do tempo_total_viagem e diminui uma unidade quando o estafeta avança no percurso correspondente a esta encomenda, ou seja, quando esta é a encomenda_atual de algum estafeta e o estafeta não está em pausa. Quando o tempo_transporte chega a 0, o estafeta está de volta ao armazém e pronto para começar uma nova encomenda. Além dos tempos, temos ainda o booleano chegou_ao_destino que começa a False e passa a True quando o estafeta atinge o destino e ainda o ultimo_local_passou que indica a última freguesia por onde o estafeta passou e é calculado através da função get_posicao tendo em conta o grafo, o caminho, as velocidades_medias, o tempo_total_viagem e o tempo_transporte. Esta função percorre as arestas correspondentes ao caminho da encomenda_atual, contando o tempo para as percorrer até que este seja igual ou superior ao tempo que já foi percorrido pelo estafeta. Assim, obtemos o último local onde o estafeta passou, observando o vértice de origem da estrada em que o estafeta se encontra.

É ainda importante referir que cada encomenda apresenta o seu próprio lock, uma vez que a atualização do tempo é realizada através de uma thread e não convém ter mais que uma thread a aceder à mesma encomenda em simultâneo.

estafeta.py

O estafeta é a entidade responsável por realizar as entregas, sendo que é caracterizado por um identificador único, pelo seu nome, pelo meio de transporte (bicicleta, moto ou carro) em que se desloca e pelo facto de este ser ou não elétrico.

O estafeta tem ainda associada uma encomenda_atual, que corresponde à encomenda que está a entregar naquele momento e que toma o valor de None

quando não está a entregar nenhuma. Possui ainda uma queue com as encomendas que tem para entregar (fila_encomendas).

Além disso, o estafeta apresenta dois contadores: a soma das classificações obtidas e o número de entregas realizadas, que permitem calcular a classificação do estafeta.

Existe ainda um parâmetro chamado pausa iniciado a 0 que corresponde ao tempo de atraso do estafeta. Este aumenta quando o utilizador escolhe atrasar esse estafeta e diminui uma unidade com o passar do tempo. Durante este período, o estafeta não avança no percurso, ficando parado no ponto em que se encontra até que a pausa atinja o valor 0.

É ainda importante referir que cada estafeta apresenta o seu próprio lock, uma vez que a atualização do tempo é realizada através de uma thread e não convém ter mais que uma thread a aceder ao mesmo estafeta em simultâneo.

health_planet.py

A health_planet contém dois dicionários, um que associa ids de estafetas aos estafetas (dict_estafetas) e outro que associa ids de encomendas a encomendas (dict_encomendas).

A health_planet possui ainda o tempo_virtual que é usado para simular a passagem do tempo dentro do programa. No main.py existe uma thread que atualiza esse tempo, de segundo em segundo, considerando que um segundo real corresponde a 1 minuto no programa. Assim, é possível simular o decorrer do programa mais rapidamente. A cada segundo, todos os estafetas são atualizados, com o auxílio da função atualiza_estado que está presente na health_planet.py. Esta função percorre todos os estafetas presentes no dict_estafetas e, para cada um deles, verifica se este tem uma encomenda atual. Em caso afirmativo, atualiza os tempos da encomenda atual, a posição onde o estafeta se encontra ou então o valor da pausa do estafeta (caso ele não seja 0), recorrendo para isso à função atualiza_estafeta_meio presente no ficheiro estafeta.py. A posição atual do estafeta é calculada utilizando a função get_posicao presente no ficheiro encomenda.py. Caso o estafeta não tenha encomenda atual, então este verifica se o estafeta contém encomendas na sua fila_encomendas. Em caso afirmativo, inicia uma encomenda com o auxílio da função comecar_nova_encomenda presente no ficheiro health_planet.py. Além disso, a atualiza_estado, após fazer as atualizações anteriormente descritas, percorre todas as encomendas, verificando para cada uma se a encomenda ainda não foi entregue, isto é se o atributo chegou_ao_destino é False. Nestes casos, é chamada a função aumenta_tempo_que_percorreu, presente no ficheiro encomenda.py, que atualiza o tempo que percorreu tanto da encomenda_atual, se esta ainda não tiver sido entregue, como das encomendas presentes na fila_encomendas dos estafetas.

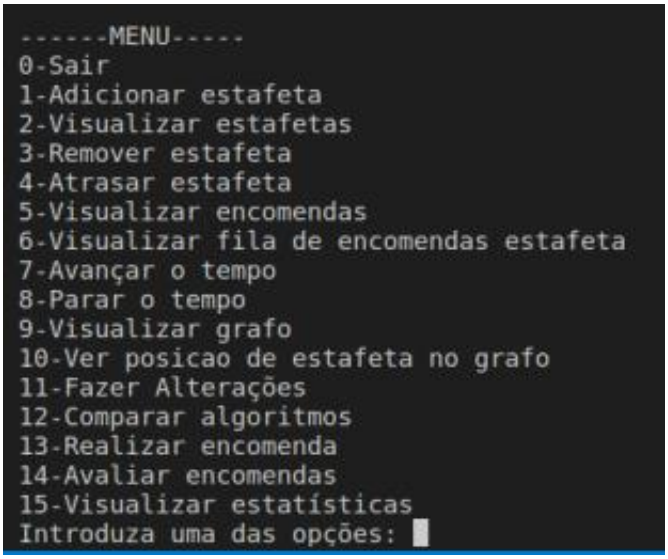
Além disso, a health_planet possui ainda uma lista das encomendas que foram entregues mas ainda não receberam a classificação do cliente, chamada encomendas_por_avaliar, que permanecem lá até que lhes seja atribuída uma classificação.

Com a existência deste tempo virtual, tivemos de criar dois locks para manipular os dicionários da health_planet, sendo estes o lock_estafetas e o lock_encomendas, e ainda um lock para manipular a lista de encomendas por avaliar, o lock_encomendas_por_avaliar.

main.py

Este ficheiro contém uma função chamada main que cria uma instância da health_planet. Em seguida, povoa-se a health_planet com estafetas com o uso da função povoa_estafetas presente no ficheiro povoar.py. Posteriormente criam-se dois grafos, sendo o primeiro criado com o auxílio da função cria_grafo do ficheiro cria_grafos.py (grafo) e um segundo grafo (grafo_cortadas) vazio onde futuramente constarão as estradas que foram cortadas. Após a execução destas tarefas, é criada uma thread que vai executar a função avanca_tempo_virtual que atualiza o tempo virtual como explicado na subsecção health_planet.py. A main contém ainda duas variáveis, altura_do_dia e meteorologia, que indicam, respetivamente, a altura do dia, (noite ou dia) e a meteorologia (sol, vento, chuva, nevoeiro ou tempestade).

A função main fica responsável pela interação com o utilizador, através de um menu interativo que lhe permite adicionar um estafeta, visualizar estafetas, remover um dado estafeta, atrasar um dado estafeta, visualizar as encomendas, visualizar a fila de encomendas de um dado estafeta, avançar no tempo, parar o tempo, visualizar o grafo, ver a posição de um estafeta no grafo, fazer alterações, comparar algoritmos, realizar uma encomenda, avaliar uma encomenda ou visualizar estatísticas.



Opção 1 - Adicionar estafeta

Se o utilizador selecionar a opção de adicionar estafetas, então é-lhe solicitado que introduza as informações do novo estafeta, sendo estas o nome do estafeta, o tipo de veículo que utiliza e se este é ou não elétrico. As informações são verificadas. Se forem validadas pelo sistema, então é criado o estafeta que será adicionado ao dict_estafetas da health_planet com o auxílio da função add_estafeta presente no ficheiro health_planet.py. Caso contrário, é fornecida no terminal uma mensagem de erro ao utilizador.

Opção 2 - Visualizar estafetas

Caso o utilizador pretenda ver os estafetas, então a main chama a função ver_estafetas presente no ficheiro health_planet.py que percorre o dict_estafetas e imprime os mesmos no terminal.

Opção 3 - Remover estafeta

Se for inserida a opção de remover estafeta, então a main pede ao utilizador que introduza o id do estafeta que pretende remover e, em seguida, chama a função remover_estafeta presente no ficheiro health_planet.py. Esta função verifica se o estafeta com o id fornecido existe. Se o estafeta existir e não contiver encomendas, então é removido. Caso contrário, é devolvida uma mensagem de erro.

Opção 4 - Atrasar estafeta

Se o utilizador pretender atrasar um estafeta, é-lhe pedido que insira o id do estafeta a atualizar e é verificado se este existe com a chamada à função existe_estafeta presente no ficheiro health_planet.py. Posteriormente é pedido ao utilizador que introduza o atraso que pretende adicionar ao estafeta e é chamada a função aumenta_pausa presente no estafeta.py. Caso algum dos valores introduzidos seja inválido, então é retornada uma mensagem de erro ao utilizador.

Opção 5 - Visualizar encomendas

Ao ser selecionada a opção de listar encomendas, a main chama a função ver_encomendas presente na health_planet.py, que percorre todas as encomendas e imprime as mesmas no terminal.

Opção 6 - Visualizar fila de encomendas estafeta

A opção de visualizar uma fila de espera de um estafeta, começa por pedir ao utilizador que introduza o id do estafeta de quem pretende visualizar a fila de espera. Em seguida verifica se o estafeta existe com a função existe_estafeta presente na health_planet.py. Em caso afirmativo, chama função ver_fila_estafeta da health_planet.py que por sua vez utiliza a função ver_encomendas_em_fila do

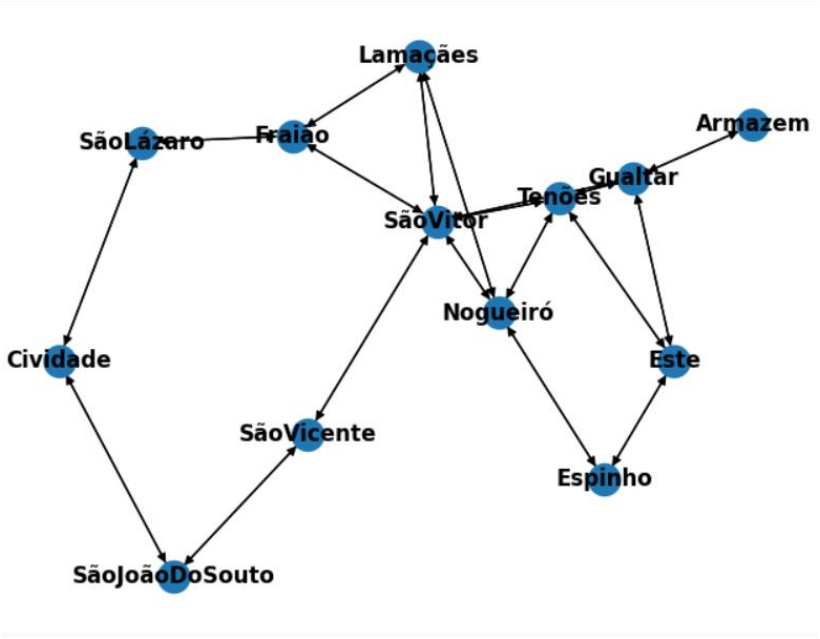
estafeta.py que retorna a lista de espera de um estafeta. Caso os dados introduzidos estejam incorretos, é devolvido ao utilizador uma mensagem de erro.

Opções 7 e 8 - Parar e avançar o tempo

Existem ainda as opções de parar o tempo e avançar o tempo que utilizam os métodos is_set e clear da biblioteca threading para sinalizar a thread em que corre o tempo. Assim, é possível parar o tempo e colocar o tempo de novo a correr, devolvendo uma mensagem de erro caso não seja possível realizar a opção. Estas funcionalidades foram desenvolvidas de modo a poder perceber melhor o decorrer do programa.

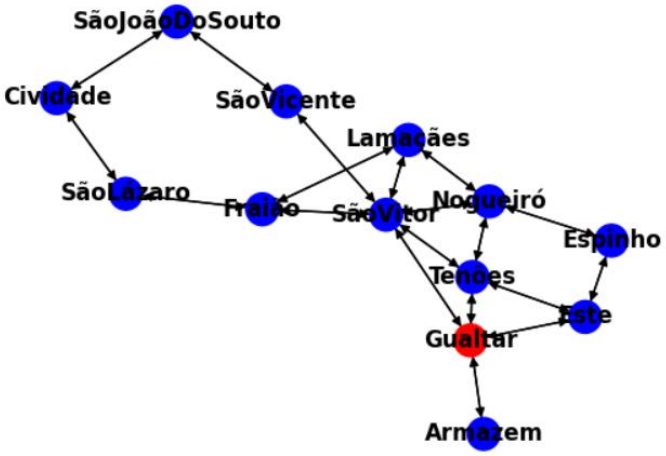
Opção 9 - Visualizar Grafo

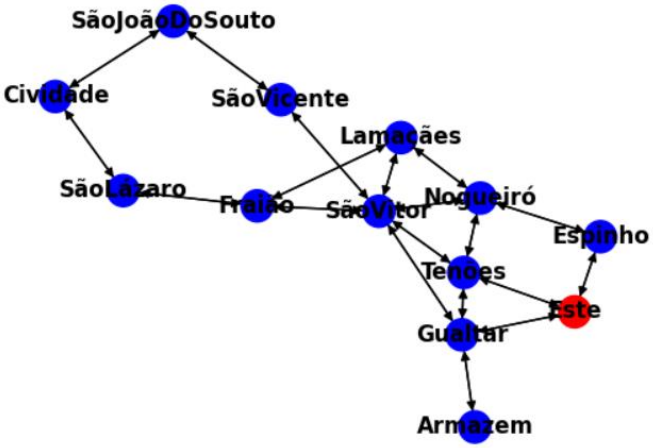
Temos ainda a opção de visualizar o grafo. Esta função utiliza os métodos disponíveis na biblioteca matplotlib para fazer uma representação gráfica do grafo do programa.



Opção 10 - Ver posição de estafeta no grafo

É possível ainda que o utilizador visualize graficamente o último local por onde um estafeta passou. Esta função utiliza também os métodos da biblioteca matplotlib mas a animação do grafo será atualizada a cada 1000 milissegundos com o auxílio da função update que verifica a última posição em que o estafeta que estamos a observar passou, de modo a atualizar a animação. O grafo apresenta todos os vértices a azul exceto o vértice em que o estafeta se encontra, estando este representado a vermelho.





Opção 11 - Fazer Alterações

Quando o utilizador introduz a opção para fazer alterações, o tempo virtual é parado e é exibido ao utilizador um novo menu que lhe permite cortar estradas, repor estradas, alterar a altura do dia, alterar a meteorologia, alterar o trânsito de uma dada estrada ou sair deste menu.

```
------ALTERAR-----
0-Terminar alterações
1-Cortar estrada
2-Repor estrada
3-Alterar altura do dia
4-Alterar meteorologia
5-Alterar trânsito de uma estrada
Introduza a opção que pretende realizar:
```

Se o utilizador pretender cortar uma estrada, primeiro verifica-se se existem estradas que possam ser cortadas (nunca deixando uma freguesia sem estradas que cheguem lá e que saiam de lá). Se existirem estradas que possam ser cortadas, então é exibido ao utilizador uma lista com as mesmas, com auxílio da função str_arestas_grafo presente no ficheiro cria_grafos.py. Em seguida o utilizador introduz o id da estrada que pretende cortar, sendo posteriormente utilizada a função mover_aresta_entre_grafos, também presente no ficheiro cria_grafos.py para retirar a aresta do grafo e colocar no grafo_cortadas. Em caso de erro, é devolvida uma mensagem de erro ao utilizador.

O repor estradas é em tudo semelhante, a única diferença é que em vez de ver as estradas que podem ser cortadas, vê se existem estradas no grafo_cortadas para que o utilizador possa escolher uma para repor. Após a escolha do utilizador, a estrada passa do grafo_cortadas para o grafo grafo.

O utilizador pode ainda mudar a altura do dia. Para tal, é-lhe pedido que insira o valor 1 se pretende mudar para dia ou 2 se pretende mudar para noite. É verificado se se consegue realizar a operação e em caso de não ser possível, é devolvida uma mensagem de erro.

Se o utilizador pretender modificar a altura do dia, a operação é em tudo idêntica à anterior, variando apenas os valores que podem ser introduzidos. Assim, o valor 1 corresponde a sol, o 2 a vento, o 3 a chuva, o 4 a nevoeiro e o 5 a tempestade.

Para alterar a velocidade de uma estrada, são apresentadas todas as estradas ao utilizador com a ajuda da função str_arestas_grafo do cria_grafos.py. Após isso, é pedido ao utilizador que introduza o id da estrada na qual pretende mudar o trânsito bem como o novo valor do trânsito (valor entre 0 e 0.9 onde 0 indica a ausência de trânsito). São validados os parâmetros introduzidos e em caso de erro é devolvida uma mensagem com indicação do mesmo.

Quando o utilizador pretender sair deste menu, todas as encomendas atuais e encomendas nas filas de encomendas de todos os estafetas vão ser atualizados de acordo com os novos parâmetros do programa, com o auxílio da função atualiza_encomenda. Esta função determina os novos caminhos do estafeta, o novo tempo para a realização da encomenda (tempo_transporte) e ainda as novas velocidades médias, nunca alterando o percurso já realizado pelo estafeta, isto é, as velocidades médias do estafeta bem como os locais em que ele já passou vão se manter. Após a atualização de todas as encomendas do sistema, o tempo volta a correr normalmente, caso inicialmente estivesse a correr.

Opção 12 - Comparar algoritmos

O utilizador pode comparar os diversos algoritmos introduzindo a opção comparar algoritmos. Esta opção pede ao utilizador que introduza a origem e o destino para o qual deseja comparar os algoritmos e devolve o trajeto completo (com as freguesias visitadas durante a procura), o trajeto final (que corresponde à solução), e o custo da solução obtido por cada algoritmo para a procura com essa origem e destino. Em caso de serem introduzidos valores inválidos, é devolvida uma mensagem de erro ao utilizador.

Opção 13 - Realizar encomenda

É ainda possível realizar novas encomendas, selecionando a opção respetiva, sendo logo apresentados os diferentes algoritmos de procura estudados para que o utilizador possa escolher aquele que pretende. Temos disponíveis para escolha 5 algoritmos de procura não informada (em largura - BFS, em profundidade - DFS, custo uniforme - Dijkstra, iterativa e bidirecional) e ainda 2 algoritmos de procura informada (Gulosa e A Estrela) que recorrem a uma heurística que corresponde à distância em linha reta entre as diferentes freguesias. Estes são explicados mais detalhadamente em algoritmos_procura.py.

Após ser selecionado o algoritmo pretendido, é pedido que se introduzam alguns detalhes sobre a encomenda como o volume, o peso, o tempo máximo para entrega e o destino pretendido, sendo criada uma encomenda com esses valores. De seguida, são calculados os caminhos de ida e de volta, separadamente, pois as estradas são diferentes nos dois sentidos, mudando sobretudo em termos de trânsito, pelo que os melhores caminhos podem ser diferentes. No fim juntam-se os dois para formar o percurso a percorrer.

O caminho calculado, juntamente com outros parâmetros são passados à função calculos que, começando pelo meio de transporte mesmo poluente (a bicicleta), vai fazendo os cálculos do tempo que demora a entrega da encomenda, utilizando para isso a função altera_velocidade que tem em conta a velocidade do veículo, o peso da encomenda, a meteorologia e a altura do dia, bem como as propriedades das arestas (distância, tipo de estrada e trânsito). Com este tempo calculado, verifica-se se o peso está dentro dos limites, se o tempo é menor do que o tempo pretendido pelo utilizador e ainda se há disponibilidade de um estafeta para realizar a entrega no tempo previsto. Caso não seja possível, tenta-se para os meios de transporte mais poluentes mas que também são, em média, mais rápidos. Se nenhum dos estafetas tiver disponibilidade para entregar a encomenda no tempo pretendido, informa-se da impossibilidade de a entregar e remove-se a encomenda. Caso seja possível entregar a encomenda, informa-se o utilizador do meio de transporte, do percurso que será seguido pelo estafeta, do tempo previsto de entrega, do preço da encomenda e ainda do caminho percorrido ao longo da execução do algoritmo.

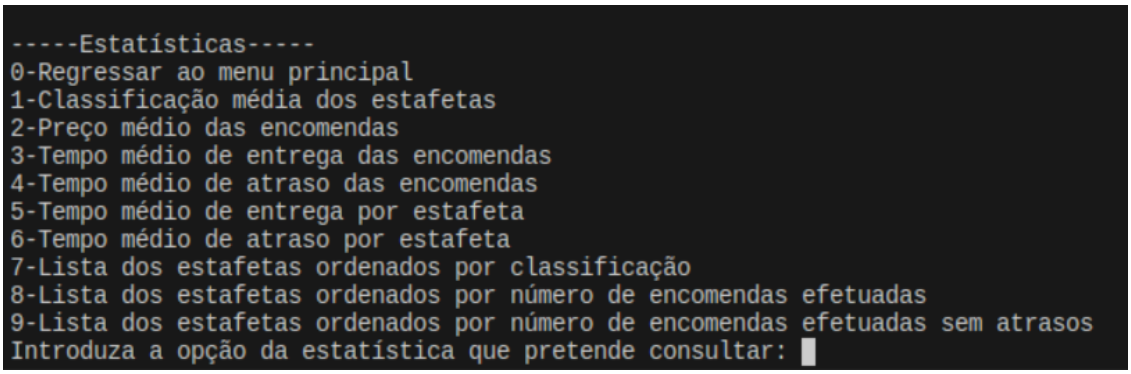
A disponibilidade é calculada através da função verifica_disponibilidade que chama a função disponibilidade presente em health_planet.py com o transporte pretendido como argumento. Esta função percorre todos os estafetas existentes e, para aqueles cujo meio de transporte é o passado como argumento, calcula o tempo que cada um deles demora a terminar todas as entregas que tem para fazer (com um acréscimo de 2 minutos em cada viagem para dar alguma margem e evitar atrasos em encomendas futuras) através da função calcula_tempo_ate_disponivel presente em estafeta.py. Ao percorrer todos os estafetas, vão sendo atualizadas as variáveis tempo_minimo_eletrico e id_condutor_min_tempo_eletrico sempre que o estafeta conduz um veículo elétrico e as variáveis tempo_minimo_sem_ser_eletrico e id_condutor_min_tempo_sem_ser_eletrico sempre que o estafeta conduz um veículo que não é elétrico de forma que fiquem guardadas nessas variáveis o tempo e o id do estafeta que estará disponível mais rapidamente para cada um dos casos. Estas variáveis são retornadas para a função verifica_disponibilidade e adicionadas ao tempo que demora a nova encomenda para que o valor obtido seja comparado com o tempo máximo solicitado. Começa por se verificar se é melhor que o veículo seja elétrico (caso das motos e dos carros) ou se é melhor não ser (caso das bicicletas), sendo para isso passado o argumento queremosEletrico que se for True indica que devemos começar por tentar com o valor do transporte elétrico retornado e se for False com o sem ser elétrico. Quando se obtém um valor menor que o tempo máximo, a encomenda é atribuída ao estafeta cujo id também foi retornado e utiliza-se para isso a função atualiza_inicial da health_planet.py, onde são passadas todas as informações sobre a encomenda à exceção das que foram passadas aquando da criação da mesma. Além disso, também é passado o seu id para que a encomenda possa ser atualizada e adicionada à fila de espera do estafeta (ou à encomenda_atual caso não haja mais nenhuma para entregar).

Opção 14 - Avaliar encomendas

Quando uma encomenda chega ao seu destino, esta é adicionada à lista encomendas_por_avaliar da health_planet.py, surgindo assim a oportunidade de o utilizador a avaliar. Quando é selecionada esta opção, é apresentada uma lista com as encomendas que é necessário avaliar e o utilizador seleciona a encomenda que pretende e atribui-lhe uma classificação. De seguida, é chamada a função regista_classificacao presente no ficheiro health_planet.py que vai buscar o atraso da encomenda e o estafeta que a realizou e chama a função atualiza_classificacao encontrada em estafeta.py. Nesta função, calcula-se uma nova classificação tendo em conta a classificação dada pelo utilizador e o atraso verificado e atualizam-se os valores n_viagens e soma_classificacoes do estafeta. A encomenda em causa é ainda retirada da lista encomendas_por_avaliar.

Opção 15 - Visualizar estatísticas

Por último, o utilizador pode visualizar algumas estatísticas relativas ao programa. Assim, quando selecionada esta opção, é apresentado ao utilizador um novo menu que lhe permite ver a classificação média dos estafetas, o preço médio das encomendas, o tempo médio das entregas, o tempo médio de atraso das encomendas, o tempo médio de entrega por estafeta e o tempo médio de atraso por estafeta, e ainda listar os estafetas ordenados por classificação, listar os estafetas por ordem de número de encomendas efetuadas ou de número de encomendas efetuadas sem atrasos. Para voltar ao menu principal o utilizador pode selecionar essa opção.



Para ver a classificação média dos estafetas, é chamada a função classificacao_media do health_planet.py que percorre todos os estafetas presentes no dict_estafetas da health_planet e devolve a sua média. Se existirem estafetas, então é exibida a média dos mesmos, caso contrário, é devolvida uma mensagem erro.

Se for pretendido ver o preço médio das encomendas, então é chamado o método preco_medio do health_planet.py que devolve as médias das encomendas presentes no dict_encomendas. Caso existam encomendas, é mostrada ao utilizador a média pretendida. Caso contrário, é mostrado ao utilizador uma mensagem de erro.

Se se pretender saber o tempo médio das encomendas, é chamada a função tempo_encomenda_medio da health_planet.py que percorre todas as encomendas presentes no dict_encomendas e devolve a média dos tempos de entrega das encomendas. Se não existirem encomendas, ao invés de ser retornado o valor pretendido, é exibida uma mensagem de erro ao utilizador.

Se se pretender saber qual o tempo médio de atraso das encomendas, utiliza-se a função tempo_atraso_medio presente na health_planet.py. Este método percorre as encomendas, verifica se estas já chegaram ao destino e em caso afirmativo soma o atraso das mesmas (método get_chegou_ao_destino do encomenda.py) para poder fazer a média dos atrasos, dividindo a soma pelo número de encomendas entregues. Se não existirem encomendas entregues, ao invés de ser apresentado este valor, é exibida uma mensagem de erro.

Para ver o tempo médio de um estafeta, utiliza-se o método tempo_encomenda_medio_por_estafeta da health_planet.py que cria um dicionário em que associa o estafeta à média dos tempos de entrega das suas encomendas e no final imprime no terminal as mesmas.

Para se ver o tempo médio de atraso dos estafetas, a main chama o método tempo_atraso_medio_por_estafeta da health_planet.py que cria um dicionário que associa o estafeta à média dos atrasos das encomendas que lhe pertencem. No final imprime no terminal os valores encontrados.

Se se pretender ordenar os estafetas por classificação, é chamado o método ordena_estafetas_classificacao da health_planet.py que cria um dicionário que associa os estafetas à sua classificação (com o uso da função get_classificacao). Posteriormente ordena-se os estafetas por ordem decrescente de classificação e imprime-se os mesmos.

Para ordenar os estafetas por número de encomendas e pelo número de encomendas sem atrasos, o processo é semelhante ao anterior. Diferem apenas nas chamadas dos métodos, que neste caso são o ordena_estafetas_nr_encomendas e o ordena_estafetas_nr_encomendas_sem_atraso, e dentro desses métodos, os valores aos quais os estafetas são associados no dicionário, sendo no primeiro caso o número de encomendas e no segundo caso o número de encomendas sem atrasos.

Ecologia no nosso trabalho

O nosso programa escolhe, por ordem de preferência, os estafetas que conduzem bicicletas não elétricas, bicicletas elétricas, motos elétricas, motos não elétricas, carros elétricos e carros não elétricos, de modo a ser o mais ecológico possível.

As velocidades das diversas estradas são afetadas pela meteorologia e altura do dia (variáveis que afetam todas as estradas), bem como pelo tipo de estrada e trânsito presente nas mesmas (variáveis associadas a cada estrada). Se for de noite, se as condições meteorológicas forem adversas, se houver trânsito e se a estrada for de um tipo pior, as viaturas têm tendência para poluir mais pois demoram mais a entregar as encomendas, gastando mais combustível ou energia (quando aplicável). Consequentemente, estes trajetos terão menor probabilidade de serem escolhidos pois, demorando mais tempo, podem não se encontrar dentro do tempo limite estipulado pelo cliente para entrega da encomenda.

Além disso, as questões ecológicas refletem-se também no preço das encomendas uma vez que quanto mais pesada for a encomenda, quanto maior for a distância para entrega e quanto menor for o tempo disponível para entrega, maior será o preço da encomenda pois estes fatores tornam mais provável a necessidade de utilização de um meio de transporte mais poluente. Ao ser utilizado um meio de transporte mais poluente, há ainda um acréscimo ao preço.

Resultados

Observando os resultados obtidos com várias chamadas à opção 12 do menu, comparar algoritmos, podemos chegar à conclusão que os algoritmos Guloso e A* retornam praticamente sempre o melhor caminho. Isto deve-se ao facto de serem algoritmos de procura informada e recorrerem ao uso de heurísticas que fornecem informações de como encontrar a solução, permitindo focar a procura nas áreas mais promissoras. Tomando decisões informadas, o algoritmo aproxima-se, normalmente, mais rapidamente de uma solução semi-ótima ou até mesmo ótima, utilizando menos recursos computacionais (compara menos nodos).

O BFS encontra sempre o caminho mais curto mas pode ser mais lento e requer mais recursos computacionais pois quando a solução fica no último nível de profundidade, é percorrido o grafo quase todo.

A procura iterativa é a que geralmente ocupa mais memória uma vez que está constantemente a visitar a raiz.

O DFS é o que usa menos recursos, porém, para quando encontra a primeira solução, acabando na maioria dos casos por não oferecer a melhor solução.

Segue um exemplo da utilização desta funcionalidade:

Neste caso em concreto, são apresentados os resultados obtidos pelos diferentes algoritmos de procura na procura do melhor caminho entre o Armazem e Cividade. Como seria de esperar, os algoritmos de procura informada (Gulosa e A*) conseguiram encontrar a solução ótima que apresentava um custo de 19 e cujo trajeto final começava em Armazem, seguia para Gualtar, depois para SãoVitor, seguido de SãoVicente e SãoJoãoDoSouto e acabando no destino pretendido, Cividade. O algoritmo de procura Gulosa passou apenas pelas freguesias do trajeto final uma vez que não tem em conta o custo acumulado e vai-se guiando apenas pelas heurísticas não voltando para trás para verificar outras possíveis soluções. Já o algoritmo de procura A* voltou atrás para verificar a freguesia de Tenões mas não obteve melhores resultados e acabou por seguir por SãoVitor até ao destino pretendido.

Houve mais 2 algoritmos a conseguir alcançar a solução ótima: o de Dijkstra que testou diversas possibilidades, passando por inúmeras freguesias, mas que acabou por chegar à solução ótima. Temos ainda o algoritmo de procura bidirecional cujos focos de procura acabaram por se encontrar em SãoVicente após alguma exploração, tendo sido beneficiado por terem sido escolhidos dois pontos que se encontram em extremidades do grafo tendo apenas 1 ou 2 caminhos por onde seguir.

Os restantes algoritmos não encontraram a solução ótima pois não têm em conta os valores do peso das arestas, logo acabaram por encontrar uma solução diferente e cujo custo é superior. O algoritmo de procura em largura foi percorrendo diversas freguesias até chegar ao destino pretendido, tendo encontrado uma solução com a mesma profundidade da solução ótima mas que não era a melhor. O algoritmo de procura em profundidade, começou um pouco desviado mas acabou por voltar para perto da solução. Encontrou uma solução logo no primeiro ramo, verificando-se que o caminho final corresponde ao trajeto completo mas tem uma profundidade superior à da solução ótima e, consequentemente, apresenta um maior valor para o custo (o maior valor verificado entre todos os algoritmos).

Por fim, o algoritmo de procura iterativa foi, de longe, o que percorreu mais vezes as freguesias, uma vez que teve de percorrer vários níveis, alguns deles diversas vezes, pois a solução apresentava uma profundidade considerável. Mesmo assim, encontrou uma solução melhor do que o algoritmo de procura em profundidade pois encontrou uma com a mesma profundidade da solução ótima tendo gasto, no entanto, bastantes mais recursos computacionais.

Conclusões

Este trabalho permitiu conhecer melhor os diversos algoritmos de procura, perceber melhor o seu funcionamento e quais as diferenças entre eles, de modo a utilizar o algoritmo mais adequado ao objetivo do programa. Além disso, permitiu perceber a grande diferença que boas heurísticas podem fazer no processo de procura de caminhos mais curtos e na velocidade de convergência dos algoritmos que as implementam.

Acrescenta o facto de colocar numa aplicação real os conteúdos abordados na UC, tornando mais clara a sua utilidade.

Achamos que cumprimos com os objetivos pretendidos para o trabalho na sua totalidade. Numa perspetiva futura poderíamos distinguir as funcionalidades por diversas vistas de utilização (cliente, estafeta e administrador) e tornar o programa uma aplicação distribuída, que pudesse ser utilizada por diversos utilizadores simultaneamente e em localizações diferentes.