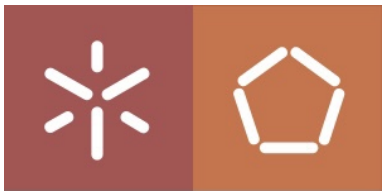

Universidade Do Minho
Engenharia Informática
Inteligência Artificial

# Trabalho Prático - Health Planet
# Grupo 15 - 2023/2024

António Silva [A100533] = 0
Diogo Barros [A100600] = 0
Duarte Leitão [A100550] = 0
Pedro Silva [A100745] = 0

02/01/2024

# Índice

# 1 Introdução

Este trabalho foi realizado no âmbito da unidade curricular de Inteligência Artificial, com o objetivo de se resolver problemas através da criação e implementação de algoritmos de procura.

Neste projeto, iniciou-se com uma descrição e formulação do problema em questão, que será solucionado, através da criação de circuitos e estratégias para resolver o problema, com o uso de vários algoritmos de procura, para se puderem obter as melhores soluções possíveis para o projeto.

Ao longo deste relatório, vai ser demonstrado todo o processo de desenvolvimento dos mecanismos de procura implementados neste projeto.

# 2 Descrição do Problema

A Health Planet, uma empresa de distribuição comprometida com a sustentabilidade ambiental, enfrenta um desafio de otimizar as suas operações de entrega de encomendas. No contexto atual, a empresa propõe-se a encontrar o **equilíbrio entre a eficiência na distribuição e a minimização do impacto ambiental.**

O centro deste desafio reside na escolha do meio de transporte ideal para cada entrega, levando em consideração a eficiência no tempo de entrega, o consumo energético e a capacidade de cada veículo.

Além disso, cada entrega está associada a um prazo máximo estipulado pelo cliente. O não cumprimento do prazo resulta em penalizações na avaliação de satisfação dos clientes, o que influencia diretamente a empresa e cada estafeta.

Neste contexto, a otimização das rotas de entrega torna-se uma peça fundamental. Considerando que cada estafeta tem um conjunto de entregas em diferentes ruas e freguesias, a empresa procura estratégias eficazes para cobrir todas as áreas de maneira eficiente e sustentável.

# 3 Formulação do Problema

O nosso problema, encontra-se num ambiente determinístico, ou seja, num ambiente que o nosso estafeta "conhece/sabe" sempre o estado em que se encontra, e onde o caminho a percorrer do estado inicial ao objetivo, é um conjunto de ações, que são percorridas como uma sequência.

- **Tipo de Problema:** Problema de Estado Único.
- **Representação do Estado Inicial:** pode ser representado pela localização atual do estafeta, ou seja, um dos nodos/freguesias do nosso grafo de Terras De Bouro.
- **Estado/Teste Objetivo:** seria o estafeta ter entregue a encomenda com sucesso, ou seja, este conseguiu chegar ao destino ao qual está associado a encomenda, com um custo reduzido e antes de atingir, ou até mesmo, ultrapassar o prazo estabelecido para se entregar a encomenda.

**Operador:**

- **Nome:** Movimento.
- **Pré-condições:** o estafeta está localizado na Health Planet, pronto para transportar a encomenda para uma localização adjacente.
- **Efeitos:** altera a localização/estado atual do estafeta no grafo.
- **Custo da Solução:** o custo da solução pode ser calculado pela soma do custo de todas as arestas ligadas dos vários nodos percorridos, até atingir o estado objetivo.
- **Estado do Problema:** o estado do problema é dinâmico, com o entregador mudando de localização à medida que se move para entregar encomendas. As encomendas são marcadas como entregues quando o entregador alcança sua localização.

# 4 Arquitetura da Solução

## 4.1 Ficheiros e pastas utilizadas

Para a realização deste trabalho prático criamos vários ficheiros Python, sendo eles:

- **Cliente** - onde tratamos das informações de cada cliente
- **Encomenda** - onde tratamos das informações de cada encomenda
- **Entregas** - onde tratamos das informações de cada entrega
- **Estafeta** - onde tratamos das informações de cada estafeta
- **Graph** - onde tratamos das informações do grafo, das funções para a criação dele, do calculo das heurísticas a partir das coordenadas dadas, do desenho do grafo e dos algoritmos que utilizamos.
- **Node** - onde tratamos das informações de cada nodo
- **Bicicleta** - onde tratamos das informações de uma bicicleta
- **Carro** - onde tratamos das informações de um carro
- **Mota** - onde tratamos das informações de uma mota
- **main** - onde criamos o grafo, carregamos dados de ficheiros para o programa, caso existam, abrimos o menu inicial e tratamos da abertura de cada menu pedido e chamamos as funções que tratam dos pedidos.
- **ui** - onde tratamos da maior parte das funções pedidas pela main, tal como as funções auxiliares que ajudam na resolução desses pedidos, e das funções encarregues do carregamento dos dados dos ficheiros ou da criação das pastas onde vamos colocá-los, caso estas não existam.

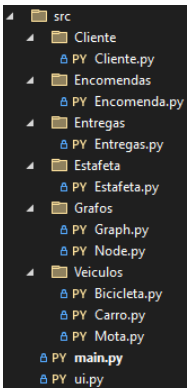


Figure 1: Ficheiros Python para a resolução deste Trabalho Prático

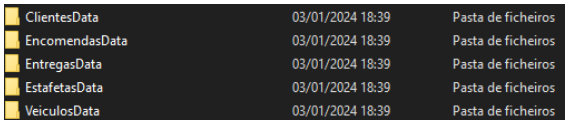


Figure 2: Pasta onde guardamos os dados

Também temos 5 pastas de dados para guardar os dados sobre os clientes, encomendas, entregas, estafetas e veículos criados durante o runtime.

## 4.2 Criação do Grafo

O nosso programa usa como circuito de entregas as freguesias de Terras de Bouro. Para representar o circuito criamos um grafo, logo ao inicializar o programa, sinalizando todas as arestas existentes.

```
def main():

    g = Graph()
    g.add_edge("Valdosende", "Balança", 21.7)
    g.add_edge("Valdosende", "Souto", 22.9)
    g.add_edge("Valdosende", "Ribeira", 20.7)
    g.add_edge("Souto", "Gondoriz", 10.9)
    g.add_edge("Ribeira", "Gondoriz", 10.6)
    g.add_edge("Balança", "Gondoriz", 8.9)
    g.add_edge("Souto", "Ribeira", 2.7)
    g.add_edge("Ribeira", "Balança", 1.9)
    g.add_edge("Balança", "Health Planet", 3.7)
    g.add_edge("Balança", "Chorense e Monte", 2.7)
    g.add_edge("Health Planet", "Gondoriz", 5.6)
    g.add_edge("Health Planet", "Chamoim e Vilar", 4.9)
    g.add_edge("Health Planet", "Chorense e Monte", 2)
    g.add_edge("Chorense e Monte", "Chamoim e Vilar", 6.5)
    g.add_edge("Chorense e Monte", "Covide", 11.7)
    g.add_edge("Chorense e Monte", "Rio Caldo", 20.4)
    g.add_edge("Chorense e Monte", "Valdosende", 19.3)
    g.add_edge("Valdosende", "Rio Caldo", 6.7)
    g.add_edge("Rio Caldo", "Covide", 9.3)
    g.add_edge("Rio Caldo", "Vilar da Veiga", 4.3)
    g.add_edge("Covide", "Vilar da Veiga", 13.5)
    g.add_edge("Covide", "Campo do Gerês", 4.5)
    g.add_edge("Covide", "Carvalheira", 7.1)
    g.add_edge("Covide", "Chamoim e Vilar", 5.9)
    g.add_edge("Carvalheira", "Campo do Gerês", 4.5)
    g.add_edge("Carvalheira", "Cibões e Brufe", 16.7)
    g.add_edge("Gondoriz", "Carvalheira", 14.6)
    g.add_edge("Gondoriz", "Cibões e Brufe", 3.9)
    g.add_edge("Gondoriz", "Chamoim e Vilar", 8.2)
    g.add_edge("Chamoim e Vilar", "Carvalheira", 7.1)
    g.add_edge("Cibões e Brufe", "Campo do Gerês", 12.7)
    g.add_edge("Vilar da Veiga", "Campo do Gerês", 16.4)
```

Figure 3: Criação do Grafo

Quando queremos visualizá-lo, temos de ir ao menu dos grafos (que vamos depois mencionar) e pedir para desenhá-lo. Ele vai questionar-nos a partir de que nodo queremos, pois ele assim vai calcular as heurísticas a partir dele.

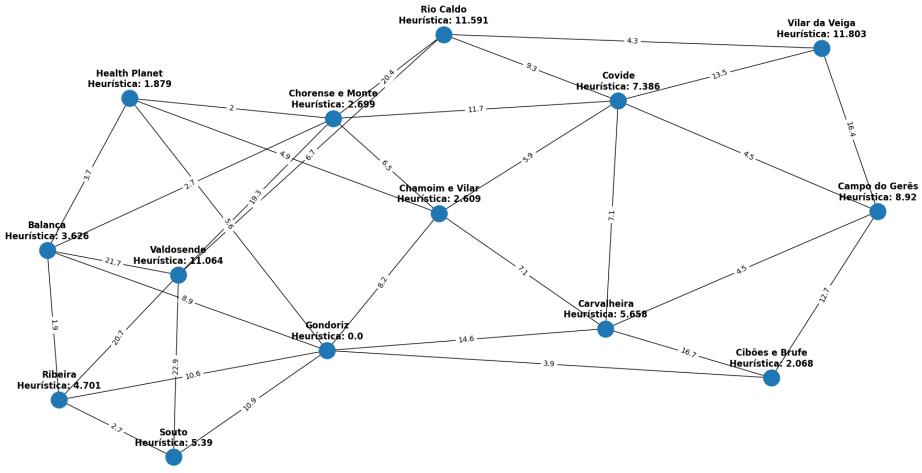


Figure 4: Grafo com Heurísticas de Gondoriz

## 4.3 Menus

Ao iniciarmos o programa deparamo-nos com o menu inicial, onde estão todos os menus que iremos utilizar neste programa.

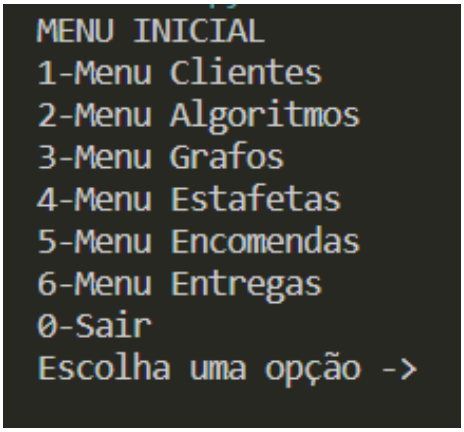


Figure 5: Menu Inicial

### 4.3.1 Menu Clientes

No menu de clientes estão disponíveis todas as funcionalidades relacionadas com os clientes da Health Planet.

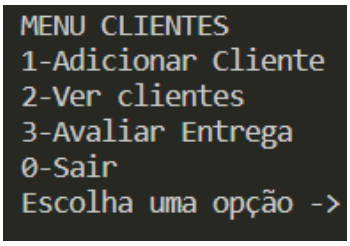


Figure 6: Menu Cliente

A primeira funcionalidade do menu consiste na introdução de um cliente no sistema. Serão necessários dois inputs: Nome do cliente e Freguesia. Será posteriormente atribuído um ID ao cliente antes da sua introdução na lista de clientes do sistema. É ainda criada uma linha no ficheiro onde são armazenados os dados dos clientes com todos os atributos do cliente em questão. Caso o sistema seja reiniciado, este ficheiro é usado para carregar todos os clientes de volta.

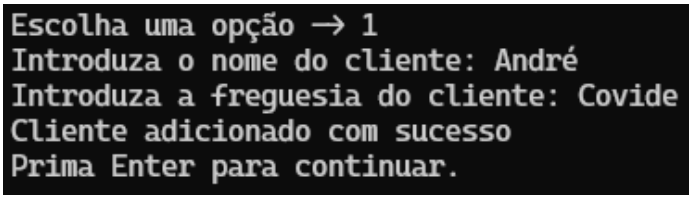


Figure 7: Criar cliente

A segunda funcionalidade presente no menu de clientes, como o próprio nome indica, tem como finalidade expor todos os clientes que já foram introduzidos no sistema, bem como todos os seus atributos.

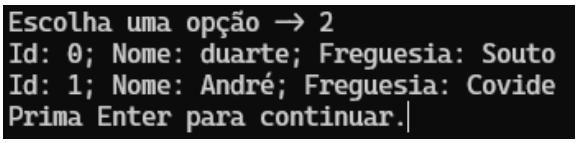


Figure 8: Ver clientes

A última funcionalidade do menu de clientes permite a um dado cliente avaliar uma entrega de uma encomenda que o próprio fez. Em primeiro lugar o cliente deve introduzir o seu ID. De seguida deve introduzir o ID da entrega que pretende avaliar. Se o ID do cliente estiver associado à entrega introduzida, o cliente pode avaliar a entrega com um número entre 0 e 5. Caso contrário, a avaliação falha.

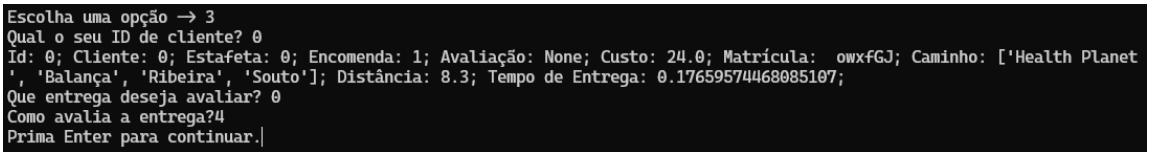


Figure 9: Avaliar entrega

### 4.3.2 Menu Algoritmos

No menu de algoritmos é possível escolher um nodo inicial e um nodo final do grafo para determinar um caminho usando um de quatro algoritmos disponíveis(DFS, BFS, Greedy, A*). Depois de executado o algoritmo escolhido, será devolvido o caminho determinado pelo algoritmo, o custo associado a esse caminho e o conjunto de nós visitados.

### 4.3.3 Menu Grafos

No menu dos grafos existem funcionalidades relacionadas com a visualização dos grafos. A opção imprimir grafo devolve no terminal um dicionário com os nodos introduzidos no grafo, e uma lista com as arestas que partem de cada nodo. As arestas encontram-se acompanhadas do seu custo.

A opção "Desenhar Grafo" requer a introdução de um nodo inicial. Será de seguida devolvida uma representação gráfica do grafo com os valores das heurísticas calculados. Esta funcionalidade recorre à biblioteca networkx.

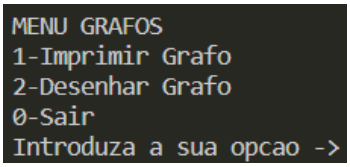


Figure 10: Menu Grafos

#### 4.3.4 Menu Estafetas

No menu de estafetas, existem funcionalidades relacionadas com a introdução de estafetas no sistema e com a visualização de informação sobre os mesmos.

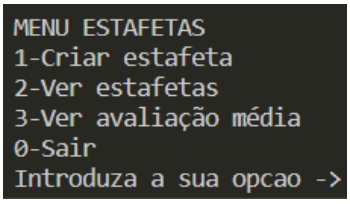


Figure 11: Menu Estafetas

A primeira funcionalidade consiste na criação do estafeta. Para tal são apenas necessários o nome do estafeta e o seu tipo de veículo. À semelhança dos clientes, o ID do estafeta é criado automaticamente pelo sistema. Para efeitos do trabalho prático, a matrículas dos veículos são geradas automaticamente, juntando 6 letras aleatórias. É então criada o veículo do estafeta. De seguida é criado o estafeta e introduzido no sistema e no ficheiro que guarda a informação dos estafetas e os seus veículos. É ainda criada uma linha no ficheiro que guarda a informação dos veículos. Este ficheiro não tem utilidade no trabalho prático, mas foi mantido para auxiliar o desenvolvimento. Vale a pena mencionar que quando um estafeta é criado, o booleano "disponivel" encontra-se com o valor True indicando que este se encontra disponível para realizar entregas. Quando o estafeta é chamado para uma entrega, o valor passa a False e só voltará a ser True depois da última entrega que o mesmo fez ser avaliada.

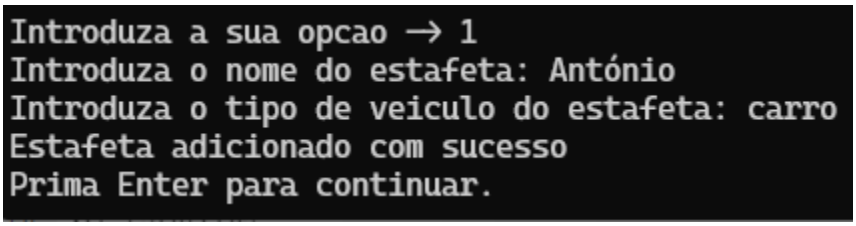


Figure 12: Criar estafeta

A segunda funcionalidade deste menu permite visualizar os estafetas introduzidos no sistema, bem como os seus atributos. Esta opção expões também informação sobre o veículo de cada estafeta.

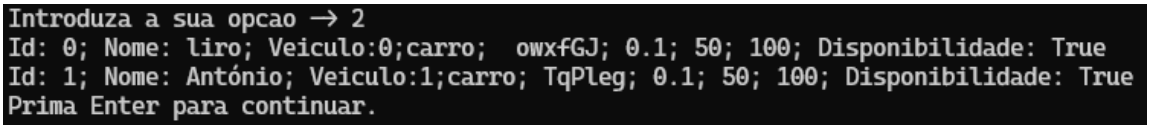


Figure 13: Ver estafetas

A última funcionalidade deste menu permite consultar a avaliação média de um estafeta. Vale a pena mencionar que o sistema irá detetar se um estafeta não existe ou não realizaou entregas.

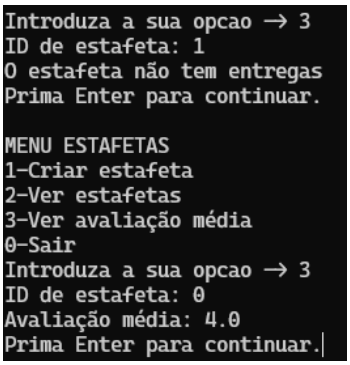


Figure 14: Ver avaliação média

### 4.3.5 Menu Encomendas

O menu das encomendas inclui funcionalidades relacionadas com a criação de uma encomenda que ficará guarda no sistema como pendente, ou seja, que não saiu para entrega.

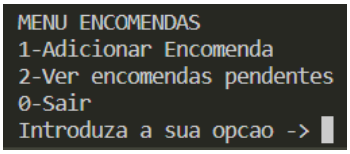


Figure 15: Avaliar entrega

A primeira funcionalidade deste menu permite criar encomenda, introduzi-la no sistema e criar uma linha no ficheiro de encomendas pendentes. É primeiro pedido o ID do cliente para fazer associações no futuro. De seguida são pedidos o peso, volume e prazo de entrega em horas. À semelhança de outras entidades do sistema, será criado automaticamente um ID para a encomenda.

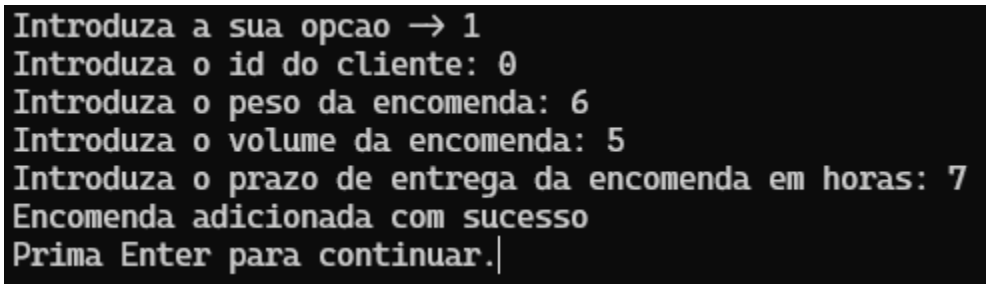


Figure 16: Criar encomenda

A última funcionalidade deste menu permite visualizar todas as encomendas pendentes introduzidas no sistema, bem como todos os seus atributos.

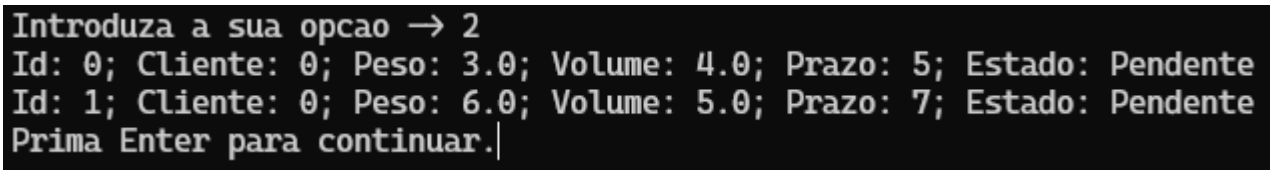


Figure 17: Ver encomendas

### 4.3.6 Menu Entregas

O menu das entregas inclui funcionalidades relacionadas com efetuar entregas e consultar o historio de entregas.

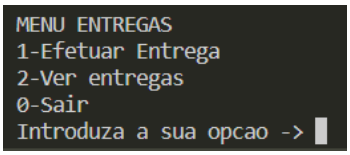


Figure 18: Avaliar entrega

Para proceder à entrega de uma encomenda que se encontrava em estado pendente, primeiro é necessário introduzir o ID da encomenda pretendida para que o sistema possa verificar se a mesma existe. De seguida é calculado o custo da entrega da encomenda com recurso ao algoritmo A*. É criado um ID automaticamente para registar a entrega. Será a seguir calculado o veiculo mais adequado com base no peso da encomenda. Será depois necessário verificar se o veículo escolhido irá conseguir realizar a entrega dentro do prazo. Irão ser consideras as diferentes velocidades máximas dos veículos e como a velocidade diminui com o aumento do peso. Se o tempo calculado for maior ou igual ao prazo especificado, será calculado o tempo com um veículo de capacidade superior. Depois de validado o veículo ideal para a entrega, será necessário verificar se existe algum estafeta com o veiculo em questão no sistema. Caso exista um veiculo disponível, a entrega é realizada e introduzida no sistema e no ficheiro das entregas.

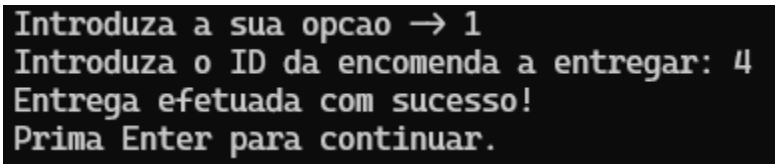


Figure 19: Criar entrega

Neste menu existe ainda a funcionalidade de visualizar todas a entregas realizadas que se encontram registadas no sistema.

```
Introduza a sua opcao → 2
Id: 0; Cliente: 0; Estafeta: 0; Encomenda: 1; Avaliação: 4; Custo: 24.0; Matrícula:  owxfGJ; Caminho: ['Health Planet',
'Balança', 'Ribeira', 'Souto']; Distância: 8.3; Tempo de Entrega: 0.17659574468085107;
Id: 1; Cliente: 0; Estafeta: 3; Encomenda: 4; Avaliação: None; Custo: 6; Matrícula:  fAPDNa; Caminho: ['Health Planet',
'Balança', 'Ribeira', 'Souto']; Distância: 8.3; Tempo de Entrega: 1.1857142857142857;
Prima Enter para continuar.
```

Figure 20: Ver entregas

# 5 Estratégias de Procura Não Informada

## 5.1 Procura em Profundidade (DFS)

A primeira estratégia que implementamos foi o DFS, que apesar de simples é a que dá os piores resultados. Conseguimos ver na figura abaixo a sua implementação:

```python
def procura_DFS(self, start, end, path=[], visited=set()):
    path.append(start)
    visited.add(start)

    if start == end:
        custoT = self.calcula_custo(path)
        return (path,round(custoT,1),visited)

    for (adjacente,peso) in self.m_graph[start]:
        if adjacente not in visited:
            resultado = self.procura_DFS(adjacente,end,path,visited)
            if resultado is not None:
                return resultado

    path.pop()
    return None
```

Figure 21: Código do DFS

### 5.1.1 Exemplo

Temos aqui um exemplo usando o DFS para encontrar um caminho entre o Health Planet e Covide. Conseguimos ver o seu caminho, o custo que foi de 80.9 e todos os nodos visitados

```
MENU ALGORITMOS
1-DFS
2-BFS
3-Gulosa
4-A*
0-Sair
Escolha uma opção -> 1
Nodo inicial->Health Planet
Nodo final->Covide
(['Health Planet', 'Balança', 'Valdosende', 'Souto', 'Gondoriz', 'Carvalheira', 'Covide'], 80.9, {'Carvalheira',
 'Souto', 'Gondoriz', 'Balança', 'Ribeira', 'Valdosende', 'Covide', 'Health Planet'})
```

Figure 22: Exemplo do DFS: Health Planet -> Covide

## 5.2 Procura em Largura (BFS)

Depois do DFS, a outra estratégia de procura não informada que implementamos foi o BFS. A implementação deste já é um pouco mais complicada, mas é muito mais eficaz comparada com o outro método não informado, como conseguimos ver pela figura abaixo:

```
def procura_BFS(self, start, end):
    visited = set()
    fila = Queue()
    custo = 0
    fila.put(start)
    visited.add(start)

    parent = dict()
    parent[start] = None

    path_found = False
    while not fila.empty() and path_found == False:
        nodo_atual = fila.get()
        if nodo_atual == end:
            path_found = True
        else:
            for (adjacente,peso) in self.m_graph[nodo_atual]:
                if adjacente not in visited:
                    fila.put(adjacente)
                    parent[adjacente] = nodo_atual
                    visited.add(adjacente)

    path = []
    if path_found:
        path.append(end)
        while parent[end] is not None:
            path.append(parent[end])
            end = parent[end]
        path.reverse()

        # função que calcula o custo do caminho
        custo = self.calcula_custo(path)
    return (path,round(custo,1),visited)
```

Figure 23: Código do BFS

### 5.2.1 Exemplo

Utilizamos o mesmo caminho para o exemplo do BFS e conseguimos ver que o caminho e o custo reduziu muito mais, mas o número de nodos visitados aumentou como consequência.

```
MENU ALGORITMOS
1-DFS
2-BFS
3-Gulosa
4-A*
0-Sair
Escolha uma opção -> 2
Nodo inicial->Health Planet
Nodo final->Covide
(['Health Planet', 'Chamoim e Vilar', 'Covide'], 10.8, {'Carvalheira', 'Souto', 'Chorense e Monte', 'Ci
bões e Brufe', 'Campo do Gerês', 'Gondoriz', 'Balança', 'Ribeira', 'Valdosende', 'Rio Caldo', 'Covide',
 'Health Planet', 'Chamoim e Vilar'})
```

Figure 24: Exemplo do BFS: Health Planet -> Covide

# 6 Estratégias de Procura Informada

## 6.1 Heurísticas

As heurísticas que nós escolhemos foi a distância em linha reta entre cada nodo, utilizando as coordenadas que calculamos a partir do Google Maps e uma função do geopy ("geodesic") para calcular essas distâncias. Decidimos utilizar estas heurísticas, pois como o custo que utilizamos deriva da distância percorrida em estrada entre cada freguesia, esta distância direta pode certificar de melhor forma que estamos a utilizar o melhor caminho com o menor custo possível.

```python
def add_heuristica(self, node1, node2):
    coord1 = self.get_coord_by_name(node1)
    coord2 = self.get_coord_by_name(node2)

    if coord1 is not None and coord2 is not None:
        distancia_km = geodesic(coord1, coord2).kilometers
        distancia_int = round(distancia_km,3)
        self.m_h[node1] = distancia_int
        return distancia_int
    else:
        print("Coordenadas não encontradas para calcular a heurística.")


def get_coord_by_name(self, name):
    coordenadas = {
        "Health Planet": (41.71807978492159, -8.30917203803056),
        "Balança": (41.704898431813334, -8.321451365436666),
        "Covide": (41.73746645742784, -8.213419535956106),
        "Souto": (41.69798855136702, -8.34528975491547),
        "Ribeira": (41.69704217187975, -8.32928045699718),
        "Valdosende": (41.65456719741683, -8.222147650257954),
        "Rio Caldo": (41.6782800365772, -8.184467528752261),
        "Chorense e Monte": (41.70991309207952, -8.304550695299849),
        "Vilar da Veiga": (41.703049799112755, -8.16645961862092),
        "Chamoim e Vilar": (41.734604350198396, -8.27073986996217),
        "Gondoriz": (41.73414387416662, -8.302094615042998),
        "Carvalheira": (41.746663597981545, -8.236168994389564),
        "Campo do Gerês": (41.758671476196184, -8.199980773826091),
        "Cibões e Brufe": (41.7438212353067, -8.280861241830227)
    }

    return coordenadas.get(name)
```

Figure 25: Funções e dados para cálculo das heurísticas

## 6.2 Procura Gulosa *(Greedy)*

A primeira estratégia informada que aplicamos foi a Procura Gulosa que só se basea nas heurísticas, ignorando os custos, logo a sua implementação não será tão complicado quanto a A* como podemos ver na figura:

```
def gulosa(self, start, end):
    open_list = set([start])
    closed_list = set([])

    parents = {}
    parents[start] = start

    while len(open_list) > 0:
        n = None

        for v in open_list:
            if n == None or self.m_h[v] < self.m_h[n]:
                n = v

        if n == None:
            print('Path does not exist!')
            return None

        if n == end:
            reconst_path = []

            while parents[n] != n:
                reconst_path.append(n)
                n = parents[n]

            reconst_path.append(start)

            reconst_path.reverse()

            return (reconst_path, round(self.calcula_custo(reconst_path),1),closed_list)

        for (m, weight) in self.getNeighbours(n):
            if m not in open_list and m not in closed_list:
                open_list.add(m)
                parents[m] = n

        open_list.remove(n)
        closed_list.add(n)

    print('Path does not exist!')
    return None
```

Figure 26: Código do Gulosa

### 6.2.1 Exemplo

No caso da Gulosa, o custo vai ser o mesmo que no BFS. Visto que o nosso grafo não é muito grande, as heurísticas somente ajudam na procura a visitar menos nodos, como podemos ver pelo número de nós visitados neste resultado:

```
MENU ALGORITMOS
1-DFS
2-BFS
3-Gulosa
4-A*
0-Sair
Escolha uma opção -> 3
Nodo inicial->Health Planet
Nodo final->Covide
(['Health Planet', 'Chamoim e Vilar', 'Covide'], 10.8, {'Health Planet', 'Chamoim e Vilar'})
```

Figure 27: Exemplo do Gulosa: Health Planet -> Covide

## 6.3 Procura A*

A última procura que implementamos, neste caso informada, foi o A*, que é conhecido como um dos melhores algoritmos de procura, pois além de se basear nas heurísticas também utiliza o custo acumulado até ao lugar que se encontra, tornando a sua implementação um pouco mais complexa.

```
def procura_aStar(self, start, end):
    open_list = {start}
    closed_list = set([])

    g = {}

    g[start] = 0

    parents = {}
    parents[start] = start

    while len(open_list) > 0:
        n = None

        for v in open_list:
            if (n == None) or (g[v] + self.getH(v) < g[n] + self.getH(n)):
                n = v
        if n == None:
            print('Path does not exist!')
            return None

        if n == end:
            reconst_path = []

            while parents[n] != n:
                reconst_path.append(n)
                n = parents[n]

            reconst_path.append(start)

            reconst_path.reverse()

            return (reconst_path, round(self.calcula_custo(reconst_path),1),closed_list)

        for (m, weight) in self.getNeighbours(n):
            if m not in open_list and m not in closed_list:
                open_list.add(m)
                parents[m] = n
                g[m] = g[n] + weight

            else:
                if g[m] > g[n] + weight:
                    g[m] = g[n] + weight
                    parents[m] = n

                    if m in closed_list:
                        closed_list.remove(m)
                        open_list.add(m)

        open_list.remove(n)
        closed_list.add(n)

    print('Path does not exist!')
    return None
```

Figure 28: Código do A*

### 6.3.1 Exemplo

No nosso último exemplo é utilizado o A*. Nota-se que tem o menor custo como as outras procuras, utilizando o melhor caminho, mas visita um nodo a mais que a Gulosa pois ele trabalha com o custo acumulado, para além da heurística, para verificar se não faria um caminho melhor/menor custo a partir do outro nodo.

```
MENU ALGORITMOS
1-DFS
2-BFS
3-Gulosa
4-A*
0-Sair
Escolha uma opção -> 4
Nodo inicial->Health Planet
Nodo final->Covide
(['Health Planet', 'Chamoim e Vilar', 'Covide'], 10.8, {'Chorense e Monte', 'Health Planet', 'Chamoim e
 Vilar'})
```

Figure 29: Exemplo do A*: Health Planet -> Covide

# 7 Comparação de resultados

Com estes exemplos, conseguimos ver que o DFS dá custos absurdamente elevados e, neste caso, os outros três algoritmos deram o mesmo custo, sendo este o menor custo possível. Isto acontece devido ao facto de o nosso grafo não ser muito grande, mas grande o suficiente para verificarmos que os algoritmos funcionam. Conseguimos também ver que quer o DFS e o BFS consultam um número elevado de freguesias enquanto que os algoritmos informadas consultam menos nodos, especialmente a Procura Gulosa pois simplesmente segue as heurísticas ignorando tudo o resto.

# 8 Conclusão

Em suma, o nosso grupo desenvolveu um programa que dado um circuito (grafo), é possível verificar as arestas de ligação entre nodos, com o seu custo associado e as heurísticas calculadas para cada nodo.

Este projeto permite utilizar algoritmos de procura para explorar os vários caminhos possíveis e conseguir atingir uma solução otimizada para cada um deles.

Foi também implementado várias funcionalidades de inserção de dados, como clientes, estafetas, encomendas e etc.... Desenvolvemos funcionalidades para criação de encomendas através dos clientes, atribuição de entregas a estafetas, cálculo de tempos de entrega com base no tipo de veículo e prazo definido pelos clientes.

A estrutura do nosso projeto, os seus dados e os algoritmos de procura explorados permitem que seja selecionada de forma inteligente as rotas e veículos adequados para otimizar o processo de entrega, considerando os diversos parâmetros como o peso, a distância, o prazo e a capacidade do veículo.

Para concluir, o projeto oferece uma gestão e otimização do processo de entrega de encomendas, desde a criação das encomendas até à sua efetiva entrega, utilizando algoritmos de procura para garantir eficiência e precisão nas rotas e nos tempos de entrega.