



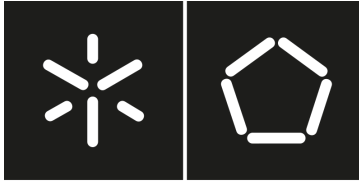
Universidade do Minho
Escola de Engenharia

Catarina Pereira
Inês Neves
Francisco Martins
Leonardo Martins

**Problem Solving
Search Algorithms**

Catarina da Cunha Malheiro da Silva Pereira
Francisco de Sousa Martins
Inês Cabral Neves
Leonardo Dias Martins

Problem Solving - Search Algorithms



Universidade do Minho

Escola de Engenharia

Catarina da Cunha Malheiro da Silva Pereira
Francisco de Sousa Martins
Inês Cabral Neves
Leonardo Dias Martins





Problem Solving - Search Algorithms

Relatório de Inteligência Artificial
para as Telecomunicações
Mestrado em Engenharia
Telecomunicações e Informática

Trabalho efetuado sob a orientação de:
Professora Doutora Dalila Durães

Identificação do Grupo e Avaliação

O Grupo 02 é composto pelos seguintes membros, três dos membros pertencem ao 1º ano do *Mestrado em Engenharia de Telecomunicações e Informática* (METI) e um dos membros pertence ao 2º ano de *Mestrado Integrado em Engenharia de Telecomunicações e Informática* (MIETI):

Imagem	Nome / Número Mecanográfico / E-mail institucional
	Catarina da Cunha Malheiro da Silva Pereira PG53733 pg537336@alunos.uminho.pt
	Francisco de Sousa Martins A93079 a93079@alunos.uminho.pt
	Inês Cabral Neves PG53864 pg53864@alunos.uminho.pt
	Leonardo Dias Martins PG53996 pg53996@alunos.uminho.pt

Os resultados desta avaliação foram consolidados e refletidos em deltas atribuídos a cada membro do grupo. Os deltas representam a parcela a ser somada à nota desta componente de avaliação e variam de forma a refletir o grau de contribuição de cada estudante.

- PG53733 Catarina Pereira DELTA = 0
- A93079 Francisco Martins DELTA = 0
- PG53864 Inês Neves DELTA = 0
- PG53996 Leonardo Martins DELTA = 0

Índice

Identificação do Grupo e Avaliação	ii
Índice de Figuras	v
Índice de Tabelas	vi
Índice de Código	vii
Lista de Acrónimos	viii
Acrónimos	viii
1 Introdução	1
2 Descrição do Problema	2
2.1 Modos de deslocação	2
2.2 Custos relativos à deslocação do veículo	2
3 Metodologia do Problema	3
3.1 Representação do Estado	3
3.2 Operadores	3
3.2.1 Operador: Adicionar Cliente	3
3.2.2 Operador: Deslocação (Ida e Volta)	4
3.2.3 Operador: Avaliar Entrega	4
3.2.4 Operador: Adicionar Entrega	4
3.2.5 Operador: Escolha de Meio de Transporte	4
3.2.6 Operador: Velocidade do Transporte	5
3.3 Custo da Solução	5
4 Algoritmos e estratégias	7
4.1 Cliente	7
4.1.1 Classe “Lista_Clientes”	7
4.1.2 Classe “Cliente”	7
4.2 Encomenda	7
4.2.1 Classe “Lista_Encomendas”	7
4.2.2 Classe “Encomenda”	8
4.2.3 Observações	8
4.3 Estafeta	8
4.3.1 Classe “Lista_Estafetas”	8
4.3.2 Classe “Estafeta”	8
4.3.3 Observações	9
4.4 Grafo	9
4.4.1 Classe “Graph”	9
4.4.2 Observações	10
4.5 Nodo	10
4.5.1 Classe “Node”	10
4.5.2 Observações	11
4.6 Main	11
4.6.1 Classe “Main”	11
4.6.2 Observações	13

Índice

5	Análise de resultados	14
6	Conclusão	15

Índice de Figuras

1	Mapa do Xpress Delta.	2
---	-------------------------------	---

Índice de Tabelas

Tabela 1: Análise de dois dos resultados obtidos para os diferentes algoritmos de procura. . . .	14
--	----

Índice de Código

Importações do ficheiro “main.py”	11
Começo do ficheiro “main.py”	11
Representação da criação dos Grafos.	11
Criação de Listas de Clientes, Estafetas e Encomendas.	12
Criação e Adição de Entregas à Lista de Entregas.	12
Iteração sobre Entregas.	12
Menu Interativo para o Utilizador.	12
Tratamento de Opções e Saída do Programa.	13

Acrónimos

METI	<i>Mestrado em Engenharia de Telecomunicações e Informática</i>
MIETI	<i>Mestrado Integrado em Engenharia de Telecomunicações e Informática</i>
UC	<i>Unidade Curricular</i>

1 Introdução

A resolução de problemas desempenha um papel fundamental no campo da Inteligência Artificial, onde algoritmos são aplicados para encontrar soluções eficazes numa variedade de contextos. Neste relatório, é apresentado os resultados de um trabalho prático realizado no âmbito da *Unidade Curricular* (UC) Inteligência Artificial para as Telecomunicações, do 1º semestre do 1º ano do Mestrado (Integrado) em Engenharia de Telecomunicações e Informática, que se concentra na conceção e implementação de algoritmos de procura aplicados a um cenário realista e relevante para a sociedade: a entrega sustentável de encomendas.

Neste contexto, a empresa de distribuição *Xpress Delta* tenta otimizar o meio de entrega de encomendas, considerando não apenas a eficiência operacional, mas também a pegada ecológica, a satisfação do cliente e diversos outros fatores críticos. Este desafio abrange a escolha de meios de transporte, prazos de entrega, classificações de clientes, características das encomendas e o custo associado aos serviços de entrega.

Este relatório descreve em detalhe a formulação do problema, a modelagem do cenário, a representação dos pontos de entrega como um grafo, a implementação de diversas estratégias de procura (tanto informada quanto não informada) e uma análise comparativa dos resultados obtidos. Além disso, também se explora a inclusão de funcionalidades adicionais, como a consideração de ações não determinísticas e outros fatores que podem afetar as estratégias de entrega sustentável.

Ao longo deste relatório, é apresentado uma visão abrangente do trabalho, destacando os desafios enfrentados, as soluções propostas e os *insights* adquiridos durante o processo de resolução deste problema complexo. Este relatório servirá como uma documentação crítica do trabalho, proporcionando uma compreensão completa do problema, da metodologia utilizada e dos resultados alcançados.

A entrega sustentável de encomendas é um tópico de crescente importância num mundo que valoriza a responsabilidade ambiental e a eficiência dos serviços. Espera-se que este relatório não apenas demonstre a compreensão e aplicação dos algoritmos de procura, mas também contribua para a discussão sobre como a inteligência artificial pode ser aplicada para abordar desafios relevantes e atuais na sociedade.

2 Descrição do Problema

O problema pode ser formulado como um problema de otimização de rotas, onde o objetivo é minimizar o custo total da entrega, considerando a distância percorrida, o meio de transporte utilizado, e a satisfação do cliente.

2.1 Modos de deslocação

Neste capítulo, serão abordadas as especificações relacionadas ao transporte de encomendas, considerando as capacidades de peso e as velocidades médias associadas às bicicletas e motos utilizadas pelos estafetas da *Xpress Delta*. Sendo consideradas as seguintes especificações:

- As bicicletas podem transportar encomendas no máximo até 5 Kg, tendo em conta uma velocidade média de 10km/h;
- As motos, podem transportar encomendas com um limite máximo de 20 Kg e com uma velocidade média de cerca de 35km/h.

2.2 Custos relativos à deslocação do veículo

A aplicação desenvolvida simula um mapa de freguesias para a entrega de encomendas, logo, existe a probabilidade de haver múltiplos caminhos para uma freguesia. Caso ocorra a situação citada, esta irá ser avaliada com base nos seguintes algoritmos:

1. Procura não Informada em Largura (BFS);
2. Procura não Informada em Profundidade (DFS);
3. Procura Informada (AStar).
4. Procura Informada (Greedy)

Para um determinado serviço de entrega por parte do estafeta (do momento que sai da Xpress Delta, realiza a entrega as encomendas e a viagem de regresso à Xpress Delta), cada deslocação de uma certa posição para outra, terá o custo de diferente, que está representado na Figura 1.

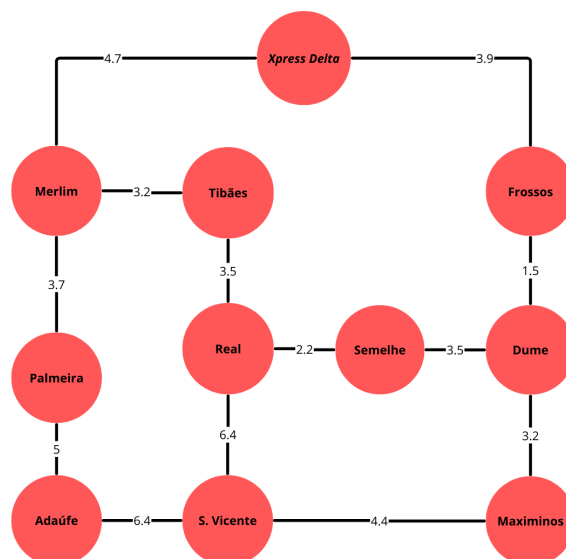


Figura 1: Mapa do Xpress Delta.

3 Metodologia do Problema

Através da análise do enunciado foi possível proceder à formulação do problema. Esta formulação foi desenvolvida com recurso a cinco componentes.

- Representação do Estado:
 - Estado Inicial
 - Estado Intermédio
 - Estado Objetivo
- Operadores (Nome, Pré-Condições, Efeitos, Custo)
- Custo da solução

3.1 Representação do Estado

Esta componente revelou ser a mais propulsora e crucial para o desenvolvimento deste projeto.

No contexto do projeto sobre uma empresa de entregas a *Xpress Delta*, os “estados” representam as diferentes situações ou configurações em que o sistema se pode encontrar durante a resolução de um problema. Cada estado representa uma condição específica do sistema, e a transição de um estado para outro é gerida pela aplicação de operadores ou ações.

- 1. Estado Inicial:** É o ponto de partida ou quando o processo de entrega começa. Representa a situação ou configuração a partir da qual os algoritmos de otimização de entrega iniciarão a sua execução.
 - **Localização:** Centro de distribuição da *Xpress Delta* (ponto de recolha).
 - **Descrição:** Este é o ponto de partida para os estafetas antes de começarem a efetuar as entregas.
- 2. Estado Intermediário:** Refere-se aos estados pelos quais o sistema de entrega passa durante o processo. Pode incluir várias condições, como o estafeta a viajar entre pontos de entrega, concluir entregas ou encontrar situações específicas durante o percurso.
 - **Localização:** Pontos de entrega ao longo do percurso.
 - **Descrição:** Cada estado intermediário representa a posição do estafeta durante o processo de entrega. Isto inclui os pontos onde as encomendas são entregues aos clientes.
- 3. Estado Objetivo:** Este é o estado objetivo, indicando a conclusão do processo de entrega. Neste caso, é o retorno ao centro de distribuição após a realização de todas as entregas.
 - **Localização:** Centro de distribuição da *Xpress Delta* (regresso).
 - **Descrição:** Este estado representa o objetivo final, onde o estafeta retorna ao centro após concluir todas as entregas.

3.2 Operadores

Os operadores são ações ou transformações que alteram o estado do sistema. No contexto da entrega sustentável de encomendas, eles representam as ações realizadas pelos estafetas para efetuar as entregas. Ao detalhar e explicar os principais operadores:

3.2.1 Operador: Adicionar Cliente

- **Nome:** Adicionar Cliente
- **Pré-Condições:** Nenhuma
- **Efeitos:** Cria um novo cliente com um nome dado e um ID sequencial. Inicializa a lista de entregas do cliente.
- **Custo:** Mínimo (geralmente não é considerado explicitamente)

3.2.2 Operador: Deslocação (Ida e Volta)

- **Nome:** Deslocação (Ida e Volta)
- **Pré-Condições:** O estafeta está numa determinada posição.
- **Efeitos:** O estafeta move-se para outra posição, alterando a sua localização.
- **Custo:** O custo associado a esta deslocação é determinado pela distância percorrida entre os nós, verificar a Figura 1.

3.2.3 Operador: Avaliar Entrega

- **Nome:** Avaliar Entrega
- **Pré-Condições:** Prazo da encomenda e tempo de entrega são conhecidos.
- **Efeitos:** O cliente atribui uma avaliação (de 1 a 5 estrelas) com base na comparação entre o prazo estipulado e o tempo real de entrega.
- **Custo:** Pode envolver custos adicionais, como o tempo de espera ou avaliação do cliente.
 - Se o tempo de entrega for igual ou inferior ao prazo estipulado, o cliente avaliará a entrega com 5 estrelas, indicando uma avaliação máxima.
 - Se o tempo de entrega estiver atrasado, mas o atraso for menor ou igual a um décimo do prazo estipulado, o cliente avaliará a entrega com 4 estrelas.
 - Se o atraso no tempo de entrega for maior que um décimo, mas menor ou igual a um quinto do prazo estipulado, o cliente avaliará a entrega com 3 estrelas.
 - Se o atraso no tempo de entrega for maior que um quinto, mas menor ou igual à metade do prazo estipulado, o cliente avaliará a entrega com 2 estrelas.
 - Se o atraso no tempo de entrega exceder a metade do prazo estipulado, o cliente avaliará a entrega com 1 estrela, indicando a menor satisfação possível.

3.2.4 Operador: Adicionar Entrega

- **Nome:** Adicionar Entrega
- **Pré-Condições:** Estafeta e encomenda são conhecidos.
- **Efeitos:** Cria uma nova entrega associada a um estafeta e a uma encomenda específicos. Inicializa informações como ponto de partida e avaliação do cliente.
- **Custo:** Mínimo (geralmente não é considerado explicitamente).

3.2.5 Operador: Escolha de Meio de Transporte

- **Nome:** Escolha de Meio de Transporte
- **Pré-Condições:** O estafeta está pronto para iniciar uma entrega.
- **Efeitos:** O estafeta retorna à base da empresa.
- **Custo:** Determinado pelas características da encomenda (peso) e pelo meio de transporte escolhido..
 - Se o veículo for uma bicicleta, o custo é 0.5.
 - Se o veículo for uma mota, o custo é 1.

3.2.6 Operador: Velocidade do Transporte

- **Nome:** Velocidade do Transporte
- **Pré-Condições:** A entrega está sendo realizada por meio de uma bicicleta ou moto.
- **Efeitos:** Atualiza a velocidade do transporte com base no meio de entrega (bicicleta ou moto).
- **Custo:** O custo é influenciado pela velocidade do transporte e pode afetar o tempo total de entrega e, consequentemente, o custo financeiro associado.
 - Se o veículo for uma bicicleta, a velocidade média de 10km/h.
 - Se o veículo for uma moto, a velocidade média de cerca de 35km/h.

Estes operadores refletem as ações chave envolvidas no processo de entrega sustentável de encomendas. A sua correta implementação e gestão são cruciais para otimizar os custos, garantir a satisfação do cliente e minimizar o impacto ambiental.

3.3 Custo da Solução

O custo da solução incorpora vários parâmetros como peso da encomenda, o tipo do veículo e a distância. Sendo a expressão do custo total a seguinte:

$$\text{preco} = \frac{\text{peso} \times \text{custo_base_por_kg}[\text{veiculo}]}{\frac{100}{\text{prazo}}} + \text{custo_por_km} \times (\text{distancia} \times 2) \quad (3.1)$$

A expressão fornecida descreve o cálculo do custo de uma solução de entrega, levando em consideração diferentes parâmetros, como o peso da encomenda, o tipo de veículo e a distância. Passo a passo, a expressão significa:

1. **Peso da Encomenda:** é multiplicado pelo custo do veículo.
2. **Custo Base por Quilograma (custo_base_por_kg):** Um custo base associado ao tipo de veículo. Este custo é uma taxa por quilograma de peso da encomenda.
3. **Custo Adicional por Quilômetro (custo_por_km):** Um custo adicional por quilômetro percorrido, aplicado à distância total (ida e volta).
4. **Cálculo do Custo Final (preco):**
 - O numerador representa o custo associado ao peso da encomenda, ajustado pelo tipo de veículo e prazo de entrega.
 - O denominador ($\frac{100}{\text{prazo}}$) normaliza o custo com base no prazo de entrega.
 - O segundo termo ($\text{custo_por_km} \times (\text{distancia} \times 2)$) adiciona o custo associado à distância total (ida e volta).

Em resumo, a expressão calcula o custo total da solução de entrega, considerando o peso da encomenda, o tipo de veículo, o custo base por quilograma, o prazo de entrega e o custo adicional por quilômetro.

O tempo de entrega é baseado na distância a ser percorrida e na velocidade média do meio de transporte escolhido.

$$\text{tempo_entrega} = \frac{\text{distancia}}{\text{velocidade}[\text{veiculo}]} \times 60 \quad (3.2)$$

Uma explicação detalhada sobre o custo do tempo de entrega:

- **Velocidade:** Esta variável associa a cada meio de transporte (“bicicleta” ou “moto”) a sua velocidade média máxima em km/h.
- Esta expressão calcula o tempo de entrega em minutos. Explicando por partes:

3.3. CUSTO DA SOLUÇÃO

- `velocidade[veiculo]`: Identifica a velocidade máxima correspondente ao meio de transporte escolhido (bicicleta ou moto).
- `distância / velocidade[veiculo]`: Calcula o tempo necessário para percorrer a distância dada a velocidade máxima do meio de transporte. Isto resultará no tempo em horas.
- $\frac{\text{distancia}}{\text{velocidade[veiculo]}} \times 60$: Multiplica o tempo em horas por 60 para converter para minutos, que é uma unidade de tempo mais comum em contextos de entrega.

Portanto, a expressão final fornece o tempo estimado de entrega em minutos com base na distância a ser percorrida e na velocidade máxima do meio de transporte escolhido. Isto é uma estimativa simples, assumindo uma velocidade constante ao longo de todo o percurso, o que pode não ser totalmente realista em situações práticas de entrega.

Para avaliar entrega recebe dois parâmetros: `prazo` e `tempo_entrega`. Esta expressão é responsável por avaliar a entrega com base no prazo estipulado e no tempo real de entrega.

Essencialmente, esta expressão modela a satisfação do cliente com a entrega com base no tempo real de entrega em relação ao prazo estipulado. Quanto mais próximo do prazo (ou antes), maior será a avaliação, enquanto atrasos progressivos resultam em avaliações mais baixas.

4 Algoritmos e estratégias

Nesta secção, serão delineados os algoritmos e estratégias adotados pelo grupo para a concepção e execução do projeto em questão. Será discutido detalhadamente o conjunto de metodologias e abordagens utilizadas, proporcionando uma visão abrangente sobre as escolhas feitas durante o processo de desenvolvimento e implementação.

Além disso, esta secção servirá como um guia abrangente para compreender a fundo a fundamentação técnica por trás do desenvolvimento do projeto, proporcionando *insights* valiosos sobre as razões por trás das escolhas algorítmicas e estratégicas do grupo.

4.1 Cliente

O ficheiro “Cliente” define duas classes relacionadas a clientes e entregas: a Classe “Lista_Clientes” e a Classe “Cliente”.

4.1.1 Classe “Lista_Clientes”

- **“__init__(self)”**: Método de inicialização da classe. Cria uma lista vazia para armazenar os clientes.
- **“getClientes(self)”**: Método que retorna a lista de clientes.
- **“addCliente(self, nome)”**: Método para adicionar um cliente à lista. Cada cliente recebe um ID sequencial baseado no número de clientes já existentes e é representado por um objeto da classe “Cliente”.

4.1.2 Classe “Cliente”

- **“__init__(self, id_cliente, nome)”**: Método de inicialização da classe “Cliente”. Cada cliente é identificado por um ID único e possui um nome. Além disso, há uma lista vazia para armazenar as entregas feitas ao cliente.
- **“getIdCliente(self)”**: Método que retorna o ID do cliente.
- **“getNomeCliente(self)”**: Método que retorna o nome do cliente.
- **“getEncomendasCliente(self)”**: Método que deveria retornar a lista de encomendas do cliente, mas o atributo correspondente (“encomendas”) não está definido na classe. .
- **“getEntregasCliente(self)”**: Método que retorna a lista de entregas feitas ao cliente.
- **“avaliar_entrega(self, prazo, tempo_entrega)”**: Método que avalia uma entrega com base no prazo estipulado e no tempo real de entrega. Retorna uma pontuação de satisfação (de 1 a 5 estrelas) com base em critérios específicos relacionados ao atraso.
- **“adicionar_entrega(self, entrega)”**: Método para adicionar uma entrega à lista de entregas do cliente.

4.2 Encomenda

O ficheiro “Encomenda” define duas classes relacionadas a encomendas: a classe “Lista_Encomendas” e a classe “Encomenda”. Nas subsecções a seguir demonstram a explicação de cada função de cada classe

4.2.1 Classe “Lista_Encomendas”

- **“__init__(self)”**: Método de inicialização da classe. Cria uma lista vazia para armazenar as encomendas.
- **“getEncomendas(self)”**: Método que retorna a lista de encomendas.
- **“addEncomenda(self, peso, prazo_maximo, cliente, morada)”**: Método para adicionar uma encomenda à lista. Cada encomenda recebe um ID sequencial baseado no número de encomendas já existentes e é representada por um objeto da classe “Encomenda”.

4.2.2 Classe “Encomenda”

- **“__init__(self, id_encomenda, peso, prazo_maximo, cliente, morada)”**: Método de inicialização da classe “Encomenda”. Cada encomenda é identificada por um ID único e possui atributos como peso, prazo máximo de entrega, cliente associado e morada.
- **“getIdEncomenda(self)”**: Método que retorna o ID da encomenda.
- **“getPeso(self)”**: Método que retorna o peso da encomenda.
- **“getPrazo(self)”**: Método que retorna o prazo máximo de entrega da encomenda.
- **“getClientEncomenda(self)”**: Método que retorna o cliente associado à encomenda.
- **“getMorada(self)”**: Método que retorna a morada da encomenda, representada como um tuplo (rua, freguesia).

4.2.3 Observações

- A função “addEncomenda” é usada para adicionar uma encomenda à lista de encomendas. Cada encomenda é representada por um objeto da classe “Encomenda” e é inicializada com os parâmetros fornecidos.
- A classe “Encomenda” possui métodos getters para acessar os atributos específicos de uma encomenda, como ID, peso, prazo, cliente associado e morada.
- No método “getMorada”, a linha “return self.morada” retorna o tuplo (rua, freguesia) associado à morada da encomenda.

Em resumo, estas classes fornecem uma estrutura para armazenar e manipular encomendas, onde cada encomenda é representada por um objeto da classe “Encomenda”, e a lista de encomendas é gerida pela classe “Lista_Encomendas”.

4.3 Estafeta

O ficheiro “estafeta.py” contém a implementação de duas classes: “Lista_Estafetas” e “Estafeta”. Estas classes modelam estafetas, que são responsáveis por realizar entregas, e são parte essencial de um sistema de gestão de entregas.

4.3.1 Classe ‘Lista_Estafetas’

- **“__init__(self)”**: Método de inicialização da classe. Inicializa uma lista vazia para armazenar estafetas.
- **“getEstafetas(self)”**: Método que retorna a lista de estafetas.
- **“addEstafeta(self, nome, meio_transporte)”**: Método para adicionar um estafeta à lista. Cada estafeta recebe um ID sequencial baseado no número de estafetas já existentes e é representado por um objeto da classe “Estafeta”.

4.3.2 Classe “Estafeta”

- **“__init__(self, id_estafeta, nome, meio_transporte)”**: Método de inicialização da classe “Estafeta”. Cada estafeta é identificado por um ID único e possui atributos como nome, meio de transporte associado e uma lista de entregas feitas por este estafeta.
- **“getAvaliacao(self)”**: Método que calcula e retorna a avaliação média do estafeta com base nas avaliações de todas as entregas realizadas por ele.
- **“adicionar_entrega(self, entrega)”**: Método para adicionar uma entrega à lista de entregas do estafeta.

- **“getId(self)”**: Método que retorna o ID do estafeta.
- **“getNome(self)”**: Método que retorna o nome do estafeta.
- **“getMeio(self)”**: Método que retorna o meio de transporte associado ao estafeta.
- **“getEntregas(self)”**: Método que retorna a lista de entregas feitas pelo estafeta.

4.3.3 Observações

- A função “getAvaliacao” calcula a avaliação média do estafeta agregando as avaliações de todas as entregas associadas a ele. Isso é feito somando as avaliações para cada critério e dividindo pelo número total de entregas.
- A função “adicionar_entrega” é usada para adicionar uma entrega à lista de entregas do estafeta.
- As funções “getId”, “getNome”, “getMeio”, e “getEntregas” são utilizadas para obter informações específicas sobre o estafeta, como o ID, o nome, o meio de transporte associado e a lista de entregas realizadas.

Estas classes fornecem uma estrutura básica para gerir e avaliar estafetas num sistema de gestão de entregas, integrando-se de forma coesa com as classes relacionadas.

4.4 Grafo

O ficheiro “grafo.py” contém a implementação da classe “Graph”, que é responsável por representar e manipular grafos. Esta classe oferece funcionalidades essenciais para a modelagem e pesquisa em grafos, e inclui métodos para adicionar arestas, procurar caminhos utilizando os algoritmos de procura em Largura (BFS) e procura em Profundidade (DFS), calcular custos de caminhos, e desenhar graficamente o grafo.

4.4.1 Classe “Graph”

- **“__init__(self, directed=False)”**: Método de inicialização da classe. Cria um grafo vazio, onde o parâmetro “directed” indica se o grafo é direcionado.
- **“__str__(self)”**: Método que retorna uma representação em *string* do grafo, exibindo os nós e as suas arestas.
- **“get_node_by_name(self, name)”**: Método que procura e retorna um nó específico pelo nome.
- **“imprime_aresta(self)”**: Método que retorna uma *string* representando as arestas do grafo.
- **“add_edge(self, node1, node2, weight)”**: Método para adicionar uma aresta ao grafo, juntamente com os seus pesos. Adiciona os nós ao grafo se ainda não existirem.
- **“getNodes(self)”**: Método que retorna a lista de nós do grafo.
- **“get_arc_cost(self, node1, node2)”**: Método que retorna o custo de uma aresta específica.
- **“calcula_custo(self, caminho)”**: Método que, dado um caminho, calcula o seu custo total.
- **“procura_DFS(self, start, end, path=[], visited=set())”**: Método que realiza uma pesquisa em profundidade no grafo, procurando um caminho entre dois nós.
- **“procura_BFS(self, start, end)”**: Método que realiza uma procura em largura no grafo, procurando um caminho entre dois nós.
- **“desenha(self)”**: Método que desenha graficamente o grafo usando a biblioteca NetworkX e Matplotlib.
- **“getNeighbours(self, nodo)”**: Método que retorna uma lista de vizinhos de um nó específico.

4.5. NODO

- **“add_heuristica(self, node1, node2, estima)”**: Método para adicionar heurísticas entre dois nós para pesquisa informada.
- **“getH(self, nodo, nodo2)”**: Método que retorna a heurística entre dois nós.
- **“greedy(self, start, end)”**: Método que implementa o algoritmo de procura gulosa (*greedy*) para encontrar um caminho entre dois nós.
- **“procura_aStar(self, start, end)”**: Método que implementa o algoritmo de A* para encontrar um caminho entre dois nós.
- **“calcula_est(self, estima)”**: Método que calcula a menor estimativa entre um conjunto de valores.

4.4.2 Observações

- A classe utiliza a biblioteca NetworkX para desenhar graficamente o grafo.
- Os métodos “greedy” e “procura_aStar” implementam algoritmos de pesquisa informada, utilizando heurísticas para melhorar a eficiência da procura.
- O método “calcula_est” é utilizado internamente para calcular a menor estimativa entre um conjunto de valores.
- O código é estruturado de forma a proporcionar uma variedade de funcionalidades de manipulação e pesquisa em grafos, tornando-o flexível e adaptável a diferentes necessidades.

4.5 Nodo

O ficheiro “nodo.py” contém a implementação da classe “Node”, que representa um nó num grafo. Esta classe é fundamental para a estruturação e manipulação dos grafos definidos no contexto do problema abordado pelo sistema.

4.5.1 Classe “Node”

- **“__init__(self, name, id=1)”**: Método de inicialização da classe. Recebe o nome (“name”) do nó como argumento e, opcionalmente, um identificador (“id”) que pode ser usado para atribuir uma numeração única ao nó.
- **“__str__(self)”**: Método que retorna uma representação em string do nó. A representação inclui o prefixo “node ’ ’” seguido do nome do nó.
- **“__repr__(self)”**: Método que retorna uma representação “oficial” do objeto. Neste caso, é similar ao método “__str__”.
- **“setid(self, id)”**: Método que define o identificador (“id”) do nó.
- **“getid(self)”**: Método que retorna o identificador do nó.
- **“getName(self)”**: Método que retorna o nome do nó.
- **“setName(self, name)”**: Método que define o nome do nó.
- **“__eq__(self, other)”**: Método para comparar dois nós. Dois nós são considerados iguais se tiverem o mesmo nome.
- **“__hash__(self)”**: Método que retorna o valor *hash* do nó. Este método é necessário quando se implementa o método “__eq__” para garantir que o objeto é “hashable” (pode ser utilizado como chave em estruturas de dados baseadas em *hash*).

4.5.2 Observações

- A classe “Node” fornece métodos básicos para a manipulação e comparação de nós em grafos.
- O uso do identificador (“id”) é opcional e não interfere diretamente nas comparações de igualdade ou cálculos de *hash*.
- A implementação dos métodos “__eq__” e “__hash__” permite que os objetos da classe “Node” sejam utilizados eficientemente em estruturas de dados que requerem a distinção de igualdade e *hash*.
- A simplicidade e clareza desta classe a tornam adequada para representar os nós em um grafo, fornecendo uma base sólida para a construção de estruturas mais complexas no contexto do sistema em questão.

4.6 Main

O ficheiro “main.py” é o ponto de entrada principal do sistema, contendo a função main() que coordena a execução do programa. Analisando as suas principais características nos subcapítulos seguintes.

4.6.1 Classe “Main”

- 1. Importações:** O código começa com a importação de módulos necessários, Código 4.1, como “defaultdict” para criar dicionários com valores padrão, e diversas classes do próprio sistema (“Graph”, “Node”, “Lista_Clientes”, “Lista_Encomendas”, “Encomenda”, “Lista_Entregas”, “Entrega”, “Lista_Estafetas”, “Estafeta”).

```
1 from collections import defaultdict
2 from grafo import Graph
3 from nodo import Node
4 from cliente import Lista_Clientes
5 from encomenda import Lista_Encomendas, Encomenda
6 from entrega import Lista_Entregas, Entrega
7 from estafeta import Lista_Estafetas, Estafeta
```

Código 4.1: Importações do ficheiro “main.py”.

- 2. Função “main()”:** A função “main()” é o ponto de entrada principal do programa.

```
1 def main():
2     # ... (código omitido para brevidade)
```

Código 4.2: Começo do ficheiro “main.py”.

- 3. Criação do Grafo:** Um objeto da classe “Graph” é criado, representando o grafo. São adicionadas arestas e heurísticas ao grafo para posterior utilização como representado no Código 4.3.

```
1 g = Graph()
2
3 g.add_edge("Xpress", "Frossos", 3.9) # Xpress - Frossos
4 # ... (código omitido para brevidade)
5
6 g.add_heuristica("Xpress", "Xpress", 0)
7 # ... (código omitido para brevidade)
8
9 g.add_heuristica("Frossos", "Frossos", 0)
10 # ... (código omitido para brevidade)
11
12 g.add_heuristica("Dume", "Dume", 0)
13 # ... (código omitido para brevidade)
14
15 g.add_heuristica("Semelhe", "Semelhe", 0)
16 # ... (código omitido para brevidade)
17
18 g.add_heuristica("Real", "Real", 0)
```

4.6. MAIN

```
19 # ... (código omitido para brevidade)
20
21 g.add_heuristica("Merlim", "Merlim", 0)
22 # ... (código omitido para brevidade)
23
24 g.add_heuristica("Palmeira", "Palmeira", 0)
25 # ... (código omitido para brevidade)
26
27 g.add_heuristica("S.Vicente", "S.Vicente", 0)
28 # ... (código omitido para brevidade)
29
30 g.add_heuristica("Tibães", "Tibães", 0)
31 # ... (código omitido para brevidade)
32
33 g.add_heuristica("Maximinos", "Maximinos", 0)
34 # ... (código omitido para brevidade)
```

Código 4.3: Representação da criação dos Grafos.

- 4. Criação de Listas de Clientes, Estafetas e Encomendas:** Listas para armazenar clientes, estafetas e encomendas são criadas e populadas, como representado no Código 4.4.

```
1 lc = Lista_Clientes()
2 lc.addCliente("Leonardo")
3 le = Lista_Estafetas()
4 le.addEstafeta("Francisco", "bicicleta")
5 lenc = Lista_Encomendas()
6 lenc.addEncomenda(5, 80, lc.getClientes()[0], ("RuaX", "S.Vicente"))
7 # ... (adição de clientes, estafetas e encomendas às listas)
8 # ... (código omitido para brevidade)
```

Código 4.4: Criação de Listas de Clientes, Estafetas e Encomendas.

- 5. Criação e Adição de Entregas à Lista de Entregas:** Uma lista de entregas é criada e preenchida com objetos de entrega, que são associados a estafetas e encomendas, representado no Código 4.5.

```
1 lent = Lista_Entregas()
2 lent.addEntrega(le.getEstafetas()[0], lenc.getEncomendas()[0])
3 # ... (adição de entregas à lista)
4 # ... (código omitido para brevidade)
```

Código 4.5: Criação e Adição de Entregas à Lista de Entregas.

- 6. Iteração sobre Entregas e Execução do Método “realizar_entrega()”:** As entregas são iteradas, e o método “realizar_entrega()” é chamado para cada entrega. Este método utiliza quatro métodos de procura diferentes, calcula custos e preços e pede avaliação ao cliente.

```
1 for entrega in lent.getEntregas():
2     entrega.realizar_entrega(g)
3     # ... (atualizações nos objetos estafeta e cliente após a entrega)
4     # ... (código omitido para brevidade)
```

Código 4.6: Iteração sobre Entregas.

- 7. Menu Interativo para o Utilizador:** Um menu interativo, Código 4.7, é apresentado ao utilizador, permitindo-lhe escolher diferentes opções para interagir com o sistema, como imprimir o grafo, desenhar o grafo, visualizar informações de entregas, entre outras.

```
1 saida = -1
2 while saida != 0:
3     # ... (menu interativo com várias opções)
4     # ... (código omitido para brevidade)
```

Código 4.7: Menu Interativo para o Utilizador.

8. Tratamento de Opções e Saída do Programa: A função “main()” é chamada se o *script* estiver a ser executado diretamente, Código 4.8.

```
1 if __name__ == "__main__":  
2     main()
```

Código 4.8: Tratamento de Opções e Saída do Programa.

4.6.2 Observações

- O código organiza as diferentes partes do sistema, como grafos, clientes, estafetas e entregas, criando instâncias e populando-as com dados iniciais.
- A estrutura modular do código permite uma fácil extensibilidade e manutenção.
- A utilização de um menu interativo proporciona uma interação flexível com o utilizador, permitindo a visualização e análise de várias informações do sistema.
- A execução do método “realizar_entrega()” e as atualizações subsequentes nos objetos estafeta e cliente refletem o ciclo de vida das entregas e a interação entre as diferentes partes do sistema.

5 Análise de resultados

A análise dos resultados das entregas é essencial para avaliar o desempenho do sistema de logística e a eficácia das estratégias de entrega. Abaixo estão os resultados de três entregas representativas, cada uma utilizando diferentes algoritmos de procura.

A Tabela 1 demonstra dois dos resultados obtidos (menor e maior entrega de prazo) para a posição inicial e posição final *Xpress Delta*.

Tabela 1: Análise de dois dos resultados obtidos para os diferentes algoritmos de procura.

Circuito de entrega do Xpress Delta					
Percurso da Entrega		Tempo Entrega (minutos)	Custo da Entrega (Km)	Custo da Entrega (€)	Avaliação (1 a 5)
Entrega com Prazo de 10 minutos					
DFS	['Xpress', 'Frossos', 'Dume', 'Semelhe', 'Real', 'Tibães', 'Merlim', 'Palmeira', 'Adaúfe', 'S.Vicente', 'Adaúfe', 'Palmeira', 'Merlim', 'Tibães', 'Real', 'Semelhe', 'Dume', 'Frossos', 'Xpress']	197.4	65.8	33.05	1
BFS	['Xpress', 'Frossos', 'Dume', 'Maximinos', 'S.Vicente', 'Maximinos', 'Dume', 'Frossos', 'Xpress']	78.0	26.0	13.15	1
Greedy	['Xpress', 'Frossos', 'Dume', 'Maximinos', 'S.Vicente', 'Maximinos', 'Dume', 'Frossos', 'Xpress']	78.0	26.0	13.15	1
AStar	['Xpress', 'Frossos', 'Dume', 'Maximinos', 'S.Vicente', 'Maximinos', 'Dume', 'Frossos', 'Xpress']	78.0	26.0	13.15	1
Entrega com Prazo de 120 minutos					
DFS	['Xpress', 'Frossos', 'Dume', 'Semelhe', 'Real', 'Tibães', 'Merlim', 'Palmeira', 'Adaúfe', 'Palmeira', 'Merlim', 'Tibães', 'Real', 'Semelhe', 'Dume', 'Frossos', 'Xpress']	159.0	53.0	27.7	2
BFS	['Xpress', 'Merlim', 'Palmeira', 'Adaúfe', 'Palmeira', 'Merlim', 'Xpress']	80.4	26.8	14.6	5
Greedy	['Xpress', 'Frossos', 'Dume', 'Maximinos', 'S.Vicente', 'Adaúfe', 'S.Vicente', 'Maximinos', 'Dume', 'Frossos', 'Xpress']	116.4	38.8	20.6	5
AStar	['Xpress', 'Merlim', 'Palmeira', 'Adaúfe', 'Palmeira', 'Merlim', 'Xpress']	80.4	26.8	14.6	5

Os resultados indicam uma clara diferença entre os algoritmos de procura, especialmente no tempo de entrega e no custo associado. Os algoritmos como BFS, Greedy e AStar mostram desempenho semelhante, superando o DFS em termos de tempo de entrega e custo.

A avaliação dos clientes está correlacionada com o tempo de entrega, mostrando que entregas mais rápidas são avaliadas mais positivamente. O custo da entrega é afetado principalmente pela escolha do algoritmo, com o DFS mostrando custos mais elevados em comparação com os outros algoritmos.

6 Conclusão

CO desenvolvimento deste projeto no âmbito da Unidade Curricular de Inteligência Artificial para as Telecomunicações proporcionou uma abordagem prática e aprofundada à resolução de problemas complexos relacionados com a entrega sustentável de encomendas. O foco principal foi a aplicação de algoritmos de procura, tanto informada quanto não informada, para otimizar o processo de entrega da empresa de distribuição *Xpress Delta*.

O modelo do problema como um problema de otimização de rotas permitiu a aplicação de algoritmos de procura para encontrar soluções eficazes, levando em conta a distância percorrida, o meio de transporte utilizado, e a satisfação do cliente.

Ao longo do relatório, detalha-se a formulação do problema, o modelo do cenário utilizando grafos, a implementação de diferentes estratégias de procura e uma análise comparativa dos resultados obtidos. Explora-se, também, funcionalidades adicionais, como a consideração de ações não determinísticas e a avaliação da entrega com base no prazo e tempo real de entrega.

As classes implementadas, como “Cliente”, “Encomenda” e “Estafeta”, forneceram uma estrutura organizada para representar e manipular os elementos essenciais do sistema de entregas.

A avaliação das entregas com base no prazo e tempo real de entrega, bem como a consideração de fatores como peso da encomenda e meio de transporte, permitiram uma abordagem realista e abrangente na busca por soluções sustentáveis.

Em resumo, o relatório servirá como uma documentação crítica do trabalho desenvolvido, proporcionando uma compreensão completa do problema, da metodologia utilizada e dos resultados alcançados.