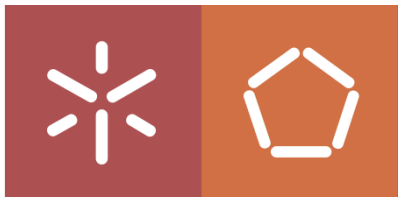
**Universidade do Minho**

Licenciatura em Engenharia Informática



Inteligência Artificial

2025/2026

# Relatório Trabalho Prático

Resolução de Problemas - Algoritmos de procura

**Grupo 015**

**106853, Ana Beatriz Freitas**

**107365, Beatriz Martins Miranda**

**106877, José Miguel Cação**

**106793, Lucas André Fernandes**

Janeiro 2026

# Avaliação pelos Pares

- A106853 Ana Beatriz Freitas DELTA = 0
- A107365 Beatriz Martins Miranda DELTA = 0
- A106877 José Miguel Cação DELTA = 0
- A106793 Lucas André Fernandes DELTA = 0

# Índice


**1 Introdução** **2**

**2 Descrição do Problema** **2**

**3 Formulação como Problema de Procura** **3**

3.1 Representação de Estado . . . 3

3.2 Estado Inicial . . . 3

3.3 Teste Objetivo . . . 3

3.4 Operadores . . . 3

3.5 Função de Custo . . . 4

3.6 Solução do Problema . . . 4

3.7 Propriedades do Espaço de Estados . . . 5

3.8 Abstração do Problema Real . . . 5

**4 Arquitetura e Estrutura do Sistema** **6**

4.1 Diagrama de Componentes . . . 6

4.2 Modelo de Dados . . . 6

4.2.1 Grafo . . . 6

4.2.2 Veículos . . . 7

4.2.3 Pedidos . . . 8

**5 Padrões de Design Utilizados** **8**

5.1 Estratégias de Seleção (Strategy) . . . 9

5.2 Separação de Responsabilidades . . . 9

5.3 Caching (Otimização) . . . 9

**6 Algoritmos de Procura** **10**

6.1 Algoritmos Pesquisa não Informada . . . 10

6.1.1 UCS - *Uniform Cost Search* . . . 10

6.1.2 BFS - *Breadth First Search* . . . 10

6.1.3 DFS - *Depth First Search* . . . 10

6.2 Algoritmos Pesquisa Informada . . . 11

6.2.1 A* . . . 11

6.2.2 *Greedy* . . . 11

6.3 Funções de Custo . . . 12

6.3.1 Custo por Distância (base geométrica) . . . 12

6.3.2 Custo por Tempo (com trânsito dinâmico) . . . 12

6.3.3 Custo Multiobjetivo (atribuição de veículos) . . . 12

6.4 Heurísticas . . . 13

6.4.1 Heurística Euclidiana (distância) . . . 13
6.4.2 Heurística Avançada (tempo + trânsito + autonomia) . . . 13
6.4.3 Prova explícita de admissibilidade da heurística avançada . . . 14

**7 Estratégias de Gestão de Frota** **15**
7.1 Seleção de Veículos . . . 15
7.1.1 Fluxo de decisão . . . 15
7.1.2 SelecaoMenorDistancia . . . 15
7.1.3 SelecaoCustoComposto . . . 15
7.1.4 SelecaoDeadMileage . . . 15
7.1.5 SelecaoEquilibrada . . . 16
7.1.6 SelecaoPriorizarEletricos . . . 16
7.2 Gestão da Autonomia e Políticas de Recarga . . . 16
7.2.1 Verificação de autonomia na atribuição . . . 16
7.2.2 Atribuição com paragem para recarga/abastecimento . . . 16
7.2.3 Encaminhamento preventivo para recarga . . . 17
7.2.4 Critério de escolha de estação . . . 17

**8 Simulador e Dinamismo** **18**
8.1 Arquitetura do Simulador . . . 18
8.1.1 Ciclo Principal . . . 18
8.1.2 Gestão de Tempo e Eventos . . . 18
8.2 Geração de Pedidos . . . 18
8.2.1 Geração Aleatória . . . 19
8.2.2 Agendamento Pré-definido . . . 19
8.3 Trânsito Dinâmico . . . 19
8.3.1 Variação Temporal . . . 19
8.3.2 Fatores de Congestionamento Zonais . . . 20
8.3.3 Impacto nas Rotas . . . 20
8.4 Simulação de Falhas em Estações . . . 21
8.4.1 Modelo Probabilístico de Falhas . . . 21

**9 Avaliação de Desempenho e Métricas** **22**
9.1 Objetivos de Avaliação . . . 22
9.2 Recolha e Agregação das Métricas . . . 22
9.3 Dead Mileage por Veículo e Rankings . . . 23

**10 Interface e Visualização** **24**
10.1 Interface Gráfica . . . 24
10.1.1 Componentes da *sidebar* . . . 24

10.1.2 Janelas de configuração (pré-simulação) . . . 24
10.2 Visualização do Grafo e Rotas . . . 25
10.2.1 Desenho do grafo e adaptação ao ecrã . . . 25
10.2.2 Representação de pedidos . . . 25
10.2.3 Representação e animação de veículos . . . 25
10.2.4 Rasto de rotas (trail) por pedido . . . 25
10.2.5 *Tooltips* e legenda . . . 25
10.3 Métricas em Tempo Real . . . 25
10.4 Ciclo de Atualização e Responsividade . . . 26
10.5 Controlo de Execução e Gestão de Estados . . . 26
10.6 Apresentação de Resultados Finais . . . 26

**11 Resultados Experimentais e Análise** **27**
11.1 Arquitetura do Ciclo de Testes . . . 27
11.2 Ferramentas de Automação e Diagnóstico . . . 27
11.3 Resultados dos testes e Análise de desempenho . . . 27
11.4 Comparação em ambiente com falhas . . . 31
11.5 Comparação de algoritmo de seleção de táxi . . . 31
11.6 Comparação com Reposicionamento Ativo . . . 32
11.7 Comparação com *Ride Sharing* . . . 33

**12 Conclusões** **34**
12.1 Análise Crítica e Resultados Alcançados . . . 34
12.2 Limitações e Trabalho Futuro . . . 34

# 1 Introdução

O presente trabalho, desenvolvido no âmbito da unidade curricular de Inteligência Artificial do 3º ano da Licenciatura em Engenharia Informática da Universidade do Minho, aborda o problema de gestão inteligente de frotas de veículos através de algoritmos de procura. O sistema foi concebido para a *TaxiGreen*, uma empresa de táxis urbanos que opera com uma frota heterogénea composta por veículos elétricos e a combustão, enfrentando o desafio de equilibrar eficiência operacional, custos e sustentabilidade ambiental.

O problema central consiste em otimizar a alocação de veículos a pedidos de transporte que chegam dinamicamente, minimizando tempos de resposta e quilómetros percorridos sem passageiros, enquanto se gere autonomia limitada dos veículos elétricos e se adapta a condições dinâmicas como variação de trânsito e falhas em estações de recarga. Pretende-se desenvolver e avaliar algoritmos de procura que otimizem a alocação de veículos minimizando custos operacionais, quilómetros sem passageiros e emissões de $CO_2$, garantindo tempos de resposta aceitáveis e adaptação a condições dinâmicas.

O relatório aborda a formulação formal do problema como problema de procura, a arquitetura e implementação do sistema desenvolvido, os algoritmos de procura utilizados, os mecanismos de dinamismo e robustez implementados, culminando com uma síntese crítica dos resultados alcançados e identificação de direções futuras.

# 2 Descrição do Problema

A *TaxiGreen* é uma empresa de táxis urbanos que opera com uma frota mista, composta por veículos elétricos e a combustão. O desafio central consiste em otimizar a gestão desta frota, garantindo que todos os pedidos de transporte sejam atendidos dentro de um período de tempo aceitável, minimizando simultaneamente custos operacionais, emissões de $CO_2$ e quilómetros percorridos sem passageiros.

As principais restrições a considerar são:

- Veículos elétricos têm autonomia limitada (80 km) e requerem recarga em estações específicas (tempo: 30 min)
- Veículos a combustão têm maior autonomia (120 km) mas custos operacionais superiores e maior impacto ambiental
- Pedidos chegam dinamicamente com diferentes prioridades e preferências ambientais
- Trânsito varia ao longo do dia (*rush hour*, hora de almoço)
- Estações de recarga podem falhar temporariamente
- Cada veículo tem capacidade máxima de passageiros

# 3 Formulação como Problema de Procura

O problema central abordado neste trabalho é o planeamento de rotas para veículos da frota *TaxiGreen* num grafo urbano, respeitando restrições de autonomia e otimizando o custo de deslocação. Seguindo a metodologia clássica de formulação de problemas de procura, define-se formalmente o espaço de estados, o estado inicial, o teste objetivo, os operadores disponíveis e a função de custo.

## 3.1 Representação de Estado

Um estado $s$ no espaço de procura é representado pelo par:

$$s = (\text{nó\_atual}, \text{autonomia\_atual}) \tag{1}$$

onde:

- nó_atual $\in N$: Localização corrente do veículo no grafo $G = (N, A)$
- autonomia_atual $\in \mathbb{R}^+$: Autonomia restante do veículo (km)

## 3.2 Estado Inicial

O estado inicial $s_0$ corresponde à configuração do veículo no momento em que recebe a atribuição do pedido:

$$s_0 = (\text{posição\_veículo}, \text{autonomia\_veículo}) \tag{2}$$

**Exemplo concreto:** Veículo elétrico E1 na localização "Centro"com 60 km de autonomia disponível:

$$s_0 = (\text{Centro}, 60) \tag{3}$$

## 3.3 Teste Objetivo

O teste objetivo $\text{Goal}(s)$ verifica se o veículo alcançou o destino pretendido:

$$\text{Goal}(s) \iff s.\text{nó\_atual} = \text{nó\_destino} \tag{4}$$

Não é necessário verificar autonomia no estado objetivo, apenas garantir que o caminho até ao destino foi viável (autonomia suficiente em cada transição).

## 3.4 Operadores

O problema dispõe de um único operador que modela a navegação no grafo:

**MOVER_PARA($n'$)** : Move o veículo do nó atual para um nó adjacente $n'$.

| Operador | Pré-condições | Efeitos |
|---|---|---|
| {MOVER_PARA($n'$) | 1. Existe aresta $(s.\text{nó}, n') \in A$<br>2. Aresta não está bloqueada: aresta.blocked = False<br>3. Autonomia suficiente: $s.\text{autonomia} \geq$ aresta.distância | Produz novo estado $s'$:<br>$s' = (n',\ s.\text{autonomia} - d(s.\text{nó\_atual}, n'))$<br>onde $d(u, v)$ representa a distância (km) da aresta entre os nós $u$ e $v$. |

Tabela 1: Descrição do operador de movimento e respetivas pré-condições e efeitos.

### 3.5 Função de Custo

O custo de uma ação corresponde ao tempo de viagem na aresta, ajustado pelo fator de congestionamento dinâmico:

$$c(\text{MOVER\_PARA}(n')) = t_{\text{base}}(s.\text{nó\_atual}, n') \times c_{\text{trânsito}}(s.\text{nó\_atual}, n', t) \tag{5}$$

onde:

- $t_{\text{base}}(u, v)$: Tempo base de viagem (minutos) sem congestionamento
- $c_{\text{trânsito}}(u, v, t)$: Fator de congestionamento no instante $t$

Se aresta.blocked = True, então $c = \infty$ (ação inviável).

**Custo do caminho:** O custo total de um caminho $\pi = [n_0, n_1, \ldots, n_k]$ é a soma dos custos das arestas percorridas:

$$\text{Custo}(\pi) = \sum_{i=0}^{k-1} c(\text{MOVER\_PARA}(n_{i+1})) \tag{6}$$

**Exemplo numérico:**

Caminho Centro → Praça → Shopping com:

- Aresta Centro-Praça: $t_{\text{base}} = 5$ min, $c_{\text{trânsito}} = 1.5$ (hora de ponta)
- Aresta Praça-Shopping: $t_{\text{base}} = 8$ min, $c_{\text{trânsito}} = 1.0$ (normal)

Custo total:

$$\text{Custo}(\pi) = (5 \times 1.5) + (8 \times 1.0) = 7.5 + 8.0 = 15.5 \text{ minutos} \tag{7}$$

### 3.6 Solução do Problema

Uma solução é qualquer caminho $\pi$ que liga o estado inicial $s_0$ a um estado objetivo $s_{\text{goal}}$ satisfazendo $\text{Goal}(s_{\text{goal}}) = \text{True}$.

A solução ótima $\pi^*$ é aquela que minimiza o custo total entre todas as soluções possíveis:

$$\pi^* = \arg\min_{\pi \in \Pi} \text{Custo}(\pi) \tag{8}$$

onde $\Pi$ é o conjunto de todos os caminhos válidos de $s_0$ até ao objetivo.

## 3.7 Propriedades do Espaço de Estados

**Tamanho do espaço:** Para um grafo com $|N| = 45$ nós e autonomia em incrementos de 1 km:

$$|S| = |N| \times A_{\max}$$

- Veículos elétricos: $|S| = 45 \times 80 = 3600$ estados
- Veículos combustão: $|S| = 45 \times 120 = 5400$ estados

Na prática, apenas uma fração destes estados é alcançável a partir de $s_0$ devido à topologia do grafo e restrições de autonomia.

**Fator de ramificação médio:** O fator de ramificação $b$ depende do grau médio dos nós no grafo:

$$b = \frac{2|A|}{|N|} = \frac{2 \times 54}{45} \approx 2.4$$

Cada nó tem, em média, 2-3 vizinhos acessíveis (excluindo arestas bloqueadas ou transições que esgotam a autonomia).

## 3.8 Abstração do Problema Real

Seguindo os princípios de abstração de problemas de procura:

**Estado (abstrato) = conjunto de estados reais:** O par (nó, autonomia) representa múltiplas configurações do mundo real (diferentes instantes temporais, diferentes pedidos atribuídos, diferentes estados de outras variáveis do sistema).

**Ação (abstrata) = conjunto de ações reais:** A ação "MOVER_PARA(n)"abstrai detalhes como aceleração, travagem, escolha de faixas, manobras específicas, etc.

**Garantia de realização:** Se existe caminho abstrato de $s_0$ a $s_{\text{goal}}$, então existe pelo menos um caminho real correspondente no mundo concreto. Esta propriedade é assegurada pela construção conservadora do modelo abstrato.

**Facilitação computacional:** A abstração reduz drasticamente a complexidade, tornando o problema tratável por algoritmos de procura clássicos (A*, UCS, etc.) em tempo útil para aplicação prática.

# 4 Arquitetura e Estrutura do Sistema

O *TaxiGreen* foi desenvolvido em *Python* seguindo uma arquitetura modular, com separação clara de responsabilidades entre modelo de dados, mecanismos de simulação/gestão e interface. Esta organização facilita a manutenção, a extensibilidade e a validação através de testes automatizados.

## 4.1 Diagrama de Componentes

A estrutura lógica do sistema pode ser descrita pelos seguintes componentes principais:

- **Fábrica de dados**: gera instâncias de demonstração do grafo, frota e pedidos (GrafoDemo, VeiculosDemo, PedidosDemo).
- **Modelo**: contém as estruturas fundamentais do domínio: grafo, pedidos e veículos.
- **Gestão e Simulação**: implementa a lógica de atribuição, planeamento e evolução temporal (gestor de frota, simulador, algoritmos de procura e estratégias).
- **Interface**: visualização do mapa e painéis de estado, com controlo da execução e apresentação de métricas.
- **Entrada principal e orquestração**: carrega configurações, instancia os componentes e inicia a interface gráfica.

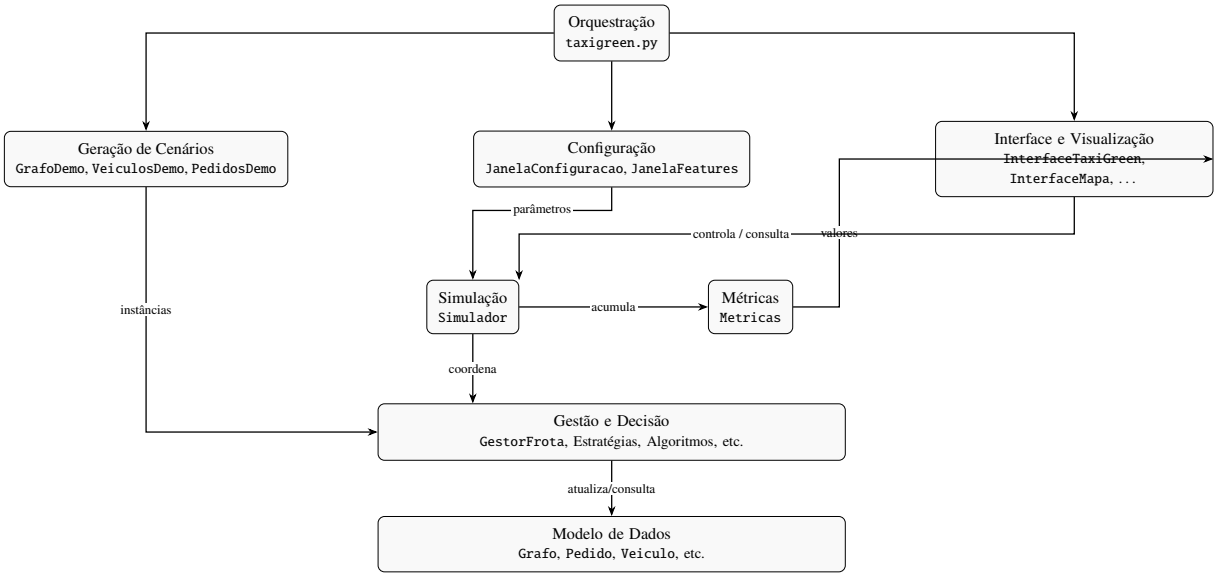



Figura 1: Diagrama de camadas do *TaxiGreen* com arestas corrigidas.

Em execução, o Simulador atua como coordenador do ciclo temporal, chamando o GestorFrota (decisões e estado da frota) e atualizando, quando aplicável, módulos dinâmicos (trânsito, falhas e *ride-sharing*). A interface consulta continuamente o estado do simulador e do gestor, apresentando-o de forma interativa.

## 4.2 Modelo de Dados

### 4.2.1 Grafo

O ambiente urbano é representado por um grafo não dirigido $G = (N, A)$. Os nós (`No`) representam localizações com coordenadas $(x, y)$ e um tipo (TipoNo) que distingue:

- zonas urbanas de recolha/entrega (`RECOLHA_PASSAGEIROS`);
- estações de recarga (`ESTACAO_RECARGA`);
- postos de abastecimento (`POSTO_ABASTECIMENTO`).

As arestas (`Aresta`) armazenam distância (km), tempo base de viagem (min), e dois mecanismos para representar perturbações:

- **congestionamento** (*congestion*): multiplicador do tempo de viagem;
- **bloqueio** (*blocked*): representa vias indisponíveis.

O tempo de viagem efetivo é calculado por:

$$t_{\text{real}} = \begin{cases} +\infty, & \text{se } \textit{blocked} = \text{True} \\ t_{\text{base}} \times c, & \text{caso contrário} \end{cases}$$

onde $c$ corresponde ao fator de congestionamento armazenado na aresta.

O grafo disponibiliza operações para consulta estrutural e suporte aos algoritmos de planeamento, nomeadamente obtenção de vizinhos ({vizinhos), distância entre nós ({distancia) e acesso direto a uma aresta ({get_aresta).

O mapa urbano é representado por um grafo com 45 nós:

- **26 zonas urbanas**: Pontos de recolha/entrega (Centro, Praça, Shopping, etc.)
- **12 estações de recarga**: Exclusivas para veículos elétricos
- **7 postos de abastecimento**: Exclusivos para veículos a combustão

### 4.2.2 Veículos

Os veículos são modelados por uma hierarquia abstrata (`Veiculo`) e duas subclasses concretas: `VeiculoEletrico` e `VeiculoCombustao`. Cada veículo mantém:

- estado operacional e posição atual;
- autonomia atual e máxima;
- capacidade de passageiros;
- métricas acumuladas por veículo (km totais e km sem passageiros);
- informação de rota (`rota` e `indice_rota`) e `id_pedido_atual`.

A dinâmica do veículo é governada por uma máquina de estados (`EstadoVeiculo`) com estados como `DISPONIVEL`, `EM_DESLOCACAO`, `A_SERVICO`, `A_CARREGAR` e `A_ABASTECER`. O método `mover_um_passo` implementa a transição ao longo de uma rota, respeitando:

- tempos de ocupação (`tempo_ocupado_ate`) para modelar duração de viagens e de recarga/abastecimento;
- consumo de autonomia e atualização de métricas de distância;
- diferenciação entre deslocação com passageiros e sem passageiros (impacto em *dead mileage*).

Os modelos de custo e emissões dependem do tipo de veículo:

**Veículos Elétricos:**

- Autonomia: 80 km
- Tempo de recarga: 30 min (recarga completa)
- Custo operacional: €0.09/km
- Emissões: 0 kg CO/km (operação direta)

Inclui custo por km, maior desgaste e penalização ambiental proporcional às emissões.

**Veículos a Combustão:**

- Autonomia: 120 km
- Tempo de abastecimento: 10 min (abastecimento completo)
- Custo operacional: €0.20/km (combustível) + €0.03/km (desgaste)
- Emissões: 0.12 kg CO/km

Inclui custo por km, desgaste reduzido e um bónus ambiental (incentivo) que reduz o custo total, nunca permitindo valores negativos.

### 4.2.3 Pedidos

Os pedidos de transporte são definidos através da classe `Pedido`. Cada pedido inclui:

- origem e destino (nós do grafo);
- número de passageiros;
- instante do pedido (minutos de simulação);
- prioridade (0–3);
- preferência ambiental (`eletrico`, `combustao` ou `qualquer`);
- estado (`EstadoPedido`) e eventual veículo atribuído.

O ciclo de vida segue, tipicamente:

`PENDENTE` → `ATRIBUIDO` → `EM_EXECUCAO` → `CONCLUIDO`

podendo ainda ocorrer `REJEITADO` quando não existe um veículo viável, ou `CANCELADO` em cenários específicos. O modelo inclui validações (ex.: passageiros positivos e origem diferente do destino) e suporte para expiração por tempo máximo de espera (`tempo_max_espera`).

# 5 Padrões de Design Utilizados

O *TaxiGreen* foi concebido para ser modular e extensível. Para isso, recorre a um padrão de variação explícita na seleção de veículos (Strategy), à separação clara de responsabilidades entre coordenação temporal e decisões de domínio, e ao caching para reduzir recomputações no planeamento.

## 5.1 Estratégias de Seleção (Strategy)

A escolha do veículo é parametrizável através do padrão *Strategy*. O {GestorFrota recebe uma instância de {EstrategiaSelecao e delega a decisão:

$$v^{\star} = \{estrategia\_selecao.selecionar(pedido, candidatos, self, tempo\_atual).$$

Desta forma, é possível trocar/adicionar estratégias sem alterar a lógica de atribuição. A descrição funcional das estratégias é apresentada na Secção 7.

## 5.2 Separação de Responsabilidades

A coordenação do tempo e eventos pertence ao {Simulador (introdução de pedidos, avanço do tempo, movimento e conclusão), enquanto o {GestorFrota concentra decisões e operações de domínio (seleção/atribuição, cálculo e viabilidade de rotas, encaminhamento para recarga). Esta divisão reduz acoplamento e facilita testes.

## 5.3 Caching (Otimização)

Para reduzir custo computacional, o {GestorFrota usa:

- {CacheRotas: reutiliza rotas calculadas (com validade temporal).
- {CacheDistancias: memoriza distâncias euclidianas entre pares de nós, explorando simetria $d(a, b) = d(b, a)$.

# 6 Algoritmos de Procura

O sistema *TaxiGreen* implementa cinco algoritmos clássicos de procura em grafos para o cálculo de rotas entre pontos da cidade. Todos os algoritmos partilham a estrutura de grafo representada pela classe Grafo.

## 6.1 Algoritmos Pesquisa não Informada

### 6.1.1 UCS - *Uniform Cost Search*

O UCS é um algoritmo de procura não informada que expande sempre o nó com menor custo acumulado. Garante a solução ótima sem necessitar de heurística.

**Função de avaliação:**

$$f(n) = g(n) \tag{9}$$

**Características da implementação:**

- Utiliza fila de prioridade ordenada por $g(n)$
- Considera trânsito dinâmico baseado em {aresta.tempo_real()
- Ignora arestas bloqueadas
- Garantia de otimalidade incondicional
- Útil como referência de comparação (solução ótima sem heurística)

### 6.1.2 BFS - *Breadth First Search*

A Procura em Largura explora o grafo nível a nível, utilizando uma fila FIFO. Encontra o caminho com menor número de nós, mas não considera custos das arestas.

**Características da implementação:**

- Utiliza *collections.deque*
- Explora nós por ordem de distância (em número de arestas) à origem
- Não considera custos - encontra caminho com menos nós, não caminho mais rápido
- Otimalidade apenas em termos de número de arestas

**Variante com *checkpoint*:** A função {bfs_com_checkpoint força a passagem por um ponto intermédio, útil para cenários onde o veículo precisa de passar por uma estação de recarga ou abastecimento.

### 6.1.3 DFS - *Depth First Search*

A Procura em Profundidade explora um ramo do grafo até à sua máxima profundidade antes de retroceder. Utiliza uma pilha LIFO e não garante otimalidade.

**Características da implementação:**

- Explora ramos em profundidade primeiro
- Mantém conjunto de nós visitados para evitar ciclos

- Não garante otimalidade - retorna qualquer caminho encontrado
- Útil para exploração do grafo, não para otimização de rotas
- Baixo consumo de memória comparado com BFS

## 6.2 Algoritmos Pesquisa Informada

### 6.2.1 A*

O algoritmo A* (*A-Star*) é um algoritmo de procura informada que combina o custo acumulado desde a origem com uma estimativa heurística até ao destino. A implementação sutiliza uma fila de prioridade para expandir os nós de forma eficiente.

**Função de avaliação:**

$$f(n) = g(n) + h(n) \tag{10}$$

onde:

- $g(n)$ - custo real acumulado desde a origem até ao nó $n$
- $h(n)$ - estimativa heurística do custo de $n$ até ao destino

**Características da implementação:**

- Utiliza *heapq* do *Python* para gestão da fila de prioridade
- Considera trânsito dinâmico através de {aresta.tempo_real()
- Ignora arestas bloqueadas (custo = $\infty$)
- Suporta heurística avançada quando há informação sobre o veículo
- Retorna tuplo (custo_total, caminho)
- Garantia de otimalidade quando a heurística é admissível

### 6.2.2 *Greedy*

O algoritmo Guloso é uma procura informada que expande sempre o nó com menor valor de heurística, ignorando completamente o custo acumulado.

**Função de avaliação:**

$$f(n) = h(n) \tag{11}$$

**Características da implementação:**

- Utiliza apenas a heurística $h(n)$ para decisões
- Ignora custo acumulado $g(n)$ durante a procura
- Não garante otimalidade - prioriza velocidade sobre qualidade
- Calcula custo total apenas após encontrar caminho
- Muito rápido, ideal para sistemas em tempo real

## 6.3 Funções de Custo

O sistema utiliza funções de custo para suportar duas decisões centrais: o planeamento de rotas, minimizando tempo de viagem no grafo, e a atribuição multi-objetivo de veículos a pedidos, quando se recorre a estratégias que equilibram tempo, custo e sustentabilidade.

### 6.3.1 Custo por Distância (base geométrica)

A distância euclidiana entre dois nós $n_a$ e $n_b$ (com coordenadas $(x_a, y_a)$ e $(x_b, y_b)$) é definida por:

$$d(n_a, n_b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}. \tag{12}$$

Esta medida é usada como base para estimativas heurísticas rápidas e para derivar custos operacionais proporcionais à distância. Por ser consultada frequentemente, o sistema otimiza o seu cálculo através de *cache* simétrico, explorando $d(a, b) = d(b, a)$.

### 6.3.2 Custo por Tempo (com trânsito dinâmico)

O custo relevante para os algoritmos de procura é o tempo real de viagem em cada aresta. O tempo real é calculado como:

$$t_{\text{real}}(\text{aresta}) = \begin{cases} +\infty, & \text{se a aresta estiver bloqueada,} \\ t_{\text{base}} \cdot c(t), & \text{caso contrário,} \end{cases} \tag{13}$$

onde $c(t)$ é um fator de congestionamento dinâmico. O {GestorTransito atualiza este fator em função da hora do dia e da zona, permitindo que o planeamento se adapte a condições variáveis.

**Estimativa heurística de tempo:** Para converter distância em tempo (minutos) é usada a aproximação:

$$h_{\text{tempo}}(n_a, n_b) = \frac{d(n_a, n_b)}{v_{\text{média}}} \cdot 60, \tag{14}$$

onde $v_{\text{média}}$ é a velocidade média em km/h (por omissão 40 km/h). Esta estimativa é usada como heurística base em procura informada.

### 6.3.3 Custo Multiobjetivo (atribuição de veículos)

A classe {FuncaoCustoComposta implementa uma função de custo para avaliar a atribuição de um veículo a um pedido, combinando objetivos com pesos distintos. O custo total é:

$$C_{\text{total}} = \alpha \cdot C_{\text{tempo}} + \beta \cdot C_{\text{operacional}} + \gamma \cdot C_{\text{emissao}} + P_{\text{prioridade}}. \tag{15}$$

**Distribuição e justificação dos pesos:** Os pesos por omissão normalizados refletem um compromisso entre qualidade de serviço e eficiência:

- $\alpha = 0.4$ (tempo): prioriza satisfação do cliente, reduzindo tempo de resposta;
- $\beta = 0.3$ (custo): controla eficiência económica, mas sem sacrificar a rapidez;
- $\gamma = 0.2$ (emissões): introduz sustentabilidade como critério explícito;
- existe ainda um peso reservado $\delta = 0.1$ para rejeições, usado quando se quantifica custo de rejeitar pedidos ({calcular_custo_rejeicao) ou em avaliações globais.

Os pesos são escolhidos de forma a manter $\alpha + \beta + \gamma + \delta = 1.0$, permitindo comparar termos após normalização.

**Componentes normalizadas:** As componentes são normalizadas por valores típicos ({NORM_*) para serem comparáveis:

$$C_{\text{emissao}} = \begin{matrix} 50.0, \\ 5.0. \end{matrix}$$

A normalização reduz dependência da escala do mapa e impede que uma componente domine a função apenas por ter valores numéricos maiores.

**Penalização por Prioridade:** A penalização por prioridade é aplicada apenas a pedidos urgentes com espera elevada:

$$P_{\text{prioridade}} = \begin{cases} 0.5, & \text{se prioridade} \geq 3 \text{ e tempo\_resposta} > 10, \\ 0, & \text{caso contrário.} \end{cases} \tag{16}$$

**Penalização por rejeição:** A função inclui ainda {calcular_custo_rejeicao, com um peso reservado $\delta = 0.1$, útil para avaliações agregadas:

$$C_{\text{rej}} = \delta \cdot \big(1 + 0.5 \cdot \text{prioridade}\big). \tag{17}$$

## 6.4 Heurísticas

O sistema usa heurísticas com diferentes níveis de informação para suportar algoritmos de procura. Dado que o A* acumula custo em minutos (soma de $t_{\text{real}}$), as heurísticas relevantes são formuladas em minutos.

### 6.4.1 Heurística Euclidiana (distância)

A heurística mais simples baseia-se em linha reta:

$$h_{\text{euclid}}(n, g) = d(n, g). \tag{18}$$

É útil como medida geométrica, e é consistente no espaço euclidiano, mas, para ser comparável com custos temporais, deve ser convertida para minutos através de uma velocidade assumida (cf. $h_{\text{tempo}}$).

### 6.4.2 Heurística Avançada (tempo + trânsito + autonomia)

A função {heuristica_avancada estima diretamente tempo em minutos e incorpora velocidade máxima (otimista), um fator de trânsito mínimo esperado e uma penalização mínima caso a autonomia seja insuficiente:

$$h_{\text{adv}}(n) = t_{\text{base}}(n) \cdot f_{\text{tr}}(t) + P_{\text{aut}}(n), \tag{19}$$

onde:

$$t_{\text{base}}(n) = \frac{d(n,g)}{v_{\max}} \cdot 60, \quad v_{\max} = 50 \text{ km/h}, \tag{20}$$

$$f_{\text{tr}}(t) = \begin{cases} 1.2, & \text{se } 7 \leq \text{hora}(t) \leq 9 \text{ ou } 17 \leq \text{hora}(t) \leq 19, \\ 1.0, & \text{caso contrário}, \end{cases} \tag{21}$$

$$P_{\text{aut}}(n) = \begin{cases} 6, & \text{se elétrico e autonomia} < d(n,g), \\ 2, & \text{se combustão e autonomia} < d(n,g), \\ 0, & \text{caso contrário}. \end{cases} \tag{22}$$

Os valores 6 e 2 minutos correspondem a uma reposição mínima de 20% (recarga: $30 * 0.2$; abastecimento: $10 * 0.2$), mantendo a estimativa otimista.

### 6.4.3 Prova explícita de admissibilidade da heurística avançada

**Teorema.** A heurística avançada $h_{\text{adv}}$ é admissível, isto é,

$$h_{\text{adv}}(n) \leq h^*(n) \quad \forall n, \tag{23}$$

onde $h^*(n)$ é o custo real ótimo (tempo mínimo, em minutos) de $n$ até ao destino $g$.

**Prova.** Fixe um nó $n$ e considere um percurso ótimo real de $n$ até $g$ com tempo total $h^*(n)$.

*(1) Distância em linha reta é limite inferior.*

Pela desigualdade triangular:

$$d(n,g) \leq D^*(n), \tag{24}$$

onde $D^*(n)$ é a distância total percorrida (km) no percurso ótimo.

*(2) Velocidade máxima fornece tempo mínimo.*

Nenhum percurso pode ter tempo inferior ao tempo de percorrer $D^*(n)$ à velocidade máxima $v_{\max}$:

$$h^*(n) \geq \frac{D^*(n)}{v_{\max}} \cdot 60 \geq \frac{d(n,g)}{v_{\max}} \cdot 60 = t_{\text{base}}(n). \tag{25}$$

*(3) Fator de trânsito usado é otimista.*

Em horas de ponta, o congestionamento real pode atingir valores até 2.0, enquanto a heurística usa $f_{\text{tr}}(t) = 1.2$ (mínimo esperado). Logo:

$$t_{\text{base}}(n) \cdot f_{\text{tr}}(t) \leq \text{tempo real necessário no percurso ótimo}. \tag{26}$$

*(4) Penalização de autonomia é mínima.*

Se autonomia $< d(n,g)$, então é necessário repor energia/combustível em algum momento para alcançar $g$. A heurística adiciona apenas o tempo mínimo de reposição parcial (20%), pelo que:

$$P_{\text{aut}}(n) \leq \text{tempo real total de reposição no percurso ótimo}. \tag{27}$$

*(5) Conclusão.*

Somando as cotas inferiores (3) e (4), obtemos:

$$h_{\text{adv}}(n) = t_{\text{base}}(n) \cdot f_{\text{tr}}(t) + P_{\text{aut}}(n) \leq h^*(n). \tag{28}$$

Logo, $h_{\text{adv}}$ é admissível.

# 7 Estratégias de Gestão de Frota

## 7.1 Seleção de Veículos

O sistema disponibiliza cinco estratégias de seleção, permitindo comparar critérios de decisão distintos para atribuir veículos a pedidos. Todas seguem a mesma interface ({EstrategiaSelecao) e operam sobre um conjunto de candidatos elegíveis preparado pelo {GestorFrota.

### 7.1.1 Fluxo de decisão

A atribuição de um pedido decorre em duas fases:

1. **Seleção**: escolha do melhor veículo entre os candidatos, usando a estratégia ativa

$$\{selecionar\_veiculo\_pedido \rightarrow \{estrategia\_selecao.selecionar$$

2. **Atribuição**: após seleção, é construída a rota completa, *pickup* + viagem, e atualizados os estados do pedido e do veículo.

As estratégias diferem apenas no critério usado para ordenar/avaliar candidatos, mantendo consistentes as verificações de viabilidade e autonomia.

### 7.1.2 SelecaoMenorDistancia

Objetivo: minimizar o tempo até à origem do pedido (*pickup*). Para cada candidato, calcula-se o custo temporal da rota veículo → pickup e escolhe-se o menor. É uma política centrada em tempo de resposta imediato, adequada quando a prioridade é reduzir espera do cliente.

No código, o critério é o custo devolvido por {gestor.verificar_viabilidade_rota(...) no percurso até ao *pickup*, mantendo a restrição de autonomia para *pickup* + destino.

### 7.1.3 SelecaoCustoComposto

Objetivo: selecionar com base num custo multiobjetivo. Para cada candidato são estimados:

- tempo até ao *pickup* (via custo temporal da rota);
- distância total (*pickup* + viagem);

e calcula-se um custo composto através de {FuncaoCustoComposta.calcular_custo_atribuicao. Esta estratégia permite ajustar o compromisso entre rapidez, custo operacional e impacto ambiental, sendo indicada quando se procura otimização mais global do serviço.

Um detalhe relevante da implementação é que o tempo de resposta usado na função composta inclui a espera desde o instante do pedido e o tempo até ao *pickup*:

$$\text{tempo\_resposta} = (t_{\text{atual}} - t_{\text{pedido}}) + \text{custo\_}pickup.$$

### 7.1.4 SelecaoDeadMileage

Objetivo: reduzir quilómetros sem passageiros. A estratégia atribui uma penalização à distância até ao *pickup* e combina-a com a distância útil da viagem:

$$C = \alpha \cdot d_{pickup} + d_{\text{viagem}},$$

onde $\alpha$ é um parâmetro (por omissão, $\alpha = 2.0$). Assim, veículos longe do *pickup* tornam-se menos atrativos, mesmo que a viagem em si seja curta. Esta política tende a melhorar eficiência operacional em cenários com distribuição geográfica dispersa de pedidos.

#### 7.1.5 SelecaoEquilibrada

Objetivo: combinar distância, custo e emissões num *score* único. Para cada candidato, calcula-se:

- distância até ao *pickup*;
- custo operacional estimado para a distância total;
- emissões estimadas para a distância total;

e aplica-se uma soma ponderada com pesos configuráveis (por omissão: 0.5 distância, 0.3 custo, 0.2 emissões), após normalização por máximos típicos. É uma opção estável quando se pretende um critério de compromisso simples e interpretável.

#### 7.1.6 SelecaoPriorizarEletricos

Objetivo: privilegiar veículos elétricos quando possível. A estratégia procede em duas etapas:

1. tenta selecionar entre candidatos elétricos;
2. se não existir solução viável, aplica a estratégia base aos veículos a combustão.

Por omissão, a estratégia base é SelecaoMenorDistancia, mas pode ser substituída. Este desenho permite promover sustentabilidade sem comprometer a capacidade de resposta em cenários com poucos elétricos disponíveis.

### 7.2 Gestão da Autonomia e Políticas de Recarga

A gestão de autonomia é tratada como um requisito de segurança operacional. Assim, o sistema evita atribuições que esgotem a autonomia e, quando necessário, encaminha veículos para repor energia/combustível.

#### 7.2.1 Verificação de autonomia na atribuição

Em todas as estratégias, após obter as distâncias relevantes, é verificado se o veículo consegue percorrer a distância total necessária para completar a operação:

$$d_{\text{total}} = d_{pickup} + d_{\text{viagem}}, \qquad \{veiculo.consegue\_percorrer(d_{\text{total}}).$$

Se esta condição falhar, o gestor tenta ainda uma atribuição alternativa que inclua uma paragem intermédia para reposição de autonomia (Secção 7.2.2).

#### 7.2.2 Atribuição com paragem para recarga/abastecimento

Quando uma atribuição direta (sem paragem) não é viável, o GestorFrota tenta construir uma rota com *checkpoint* numa estação/posto compatível com o tipo de veículo. O procedimento é implementado em `{atribuir_com_recarga}` e segue o padrão:

$$\text{posição atual} \rightarrow \text{estação} \rightarrow \textit{pickup} \rightarrow \text{destino}.$$

Para cada estação candidata, são avaliadas três rotas e o seu custo temporal:

1. posição atual → estação,
2. estação → origem do pedido,
3. origem → destino.

Entre as alternativas viáveis, seleciona-se a que minimiza o custo total ($c_1 + c_2 + c_3$) e a rota resultante é atribuída ao veículo.

### 7.2.3 Encaminhamento preventivo para recarga

Além da recarga reativa, quando necessária para servir um pedido, existe uma política preventiva executada por {verificar_necessidade_recarga. Sempre que um veículo está disponível e a sua autonomia cai abaixo de um limiar, este é encaminhado para a estação viável mais próximo:

$$\{autonomia\_km \leq \theta \cdot \{autonomiaMax\_km,$$

onde $\theta$ é um parâmetro (por omissão, $\theta = 0.3$). Esta política reduz a probabilidade de indisponibilidade súbita em períodos de maior procura e mantém a frota operacional.

### 7.2.4 Critério de escolha de estação

Nas políticas de recarga, a escolha da estação é feita por proximidade efetiva no grafo, isto é, compara-se a menor distância de rota viável até cada estação candidata, não apenas distância geométrica. O gestor calcula a rota para cada estação, descarta opções inviáveis e seleciona a alternativa com menor distância/custo, atribuindo de seguida a rota ao veículo.

# 8 Simulador e Dinamismo

## 8.1 Arquitetura do Simulador

O simulador é responsável por gerir a progressão temporal e a coordenação de todos os subsistemas dinâmicos através de um ciclo principal que executa minuto-a-minuto.

### 8.1.1 Ciclo Principal

O ciclo de simulação executa as seguintes operações a cada iteração:

1. **Atualizar trânsito**: Ajusta congestionamento de todas as arestas baseado na hora atual
2. **Processar novos pedidos**: Retira pedidos da fila de prioridade cujo instante chegou
3. **Atribuir pedidos pendentes**: Aloca veículos disponíveis a pedidos (com espera mínima de 2 minutos)
4. **Mover veículos**: Avança veículos uma posição nas suas rotas
5. **Verificar conclusões**: Deteta pedidos completados e liberta veículos
6. **Verificar recargas**: Identifica veículos com autonomia <20% e agenda recarga automática
7. **Reposicionamento**: A cada 5 minutos, move veículos ociosos para zonas de alta demanda
8. **Atualizar interface**: Redesenha mapa e métricas em tempo real
9. **Avançar tempo**: $t := t + 1$ minuto

### 8.1.2 Gestão de Tempo e Eventos

O simulador utiliza uma fila de prioridade para gestão eficiente de eventos temporais:

$$\text{fila\_pedidos} = [(t_1, -p_1, \text{id}_1, \text{pedido}_1), \ldots] \tag{29}$$

onde:

- $t_i$ - instante de chegada do pedido (chave primária)
- $-p_i$ - prioridade negada (para ordenação decrescente)
- $\text{id}_i$ - identificador único do pedido (desempate)
- $\text{pedido}_i$ - objeto Pedido completo

**Controlo de velocidade:** A simulação suporta diferentes velocidades de execução (1x, 2x, 5x) através de ajuste do intervalo de atualização da interface gráfica, permitindo análise acelerada ou observação detalhada do comportamento do sistema.

## 8.2 Geração de Pedidos

O simulador suporta dois modos de geração de pedidos, adequados a diferentes cenários de teste.

### 8.2.1 Geração Aleatória

A função gerar_pedidos_aleatorios(n, zonas) cria *n* pedidos com parâmetros aleatórios:

**Parâmetros aleatorizados:**

- **Origem/destino:** Seleção uniforme de zonas distintas
- **Passageiros:** Uniforme$(1, 4)$
- **Instante:** Uniforme$(0, T_{max})$
- **Preferência ambiental:** {elétrico, combustão} com igual probabilidade
- **Prioridade:** Uniforme$(0, 3)$

**Aplicação:** Este modo é útil para testes de stress com elevado volume de pedidos, avaliação de comportamento médio do sistema e simulações estocásticas com múltiplas execuções.

### 8.2.2 Agendamento Pré-definido

A classe PedidosDemo implementa cenários pré-definidos com distribuição temporal controlada.

**Características dos cenários de teste:**

- 30 pedidos distribuídos ao longo de 60 minutos
- Mistura de zonas centrais e periféricas
- Preferências ambientais balanceadas (compatível com frota de 2 elétricos + 2 combustão)
- Variação de prioridades e número de passageiros
- Picos de procura em horários específicos (opcional)

Este modo permite reprodutibilidade e análise comparativa de diferentes estratégias sobre o mesmo conjunto de pedidos.

## 8.3 Trânsito Dinâmico

Um sistema de simulação de trânsito dinâmico foi incorporado no simulador, de modo a replicar mais fielmente um ambiente de tráfego variável. A cada aresta do grafo foi associado um valor de congestionamento, variando até 2.0, que atua como multiplicador do tempo de travessia.

### 8.3.1 Variação Temporal

Visto que o trânsito não é totalmente aleatório, mas sim afetado por fatores tangíveis como a hora do dia, foi implementado na simulação um sistema em que, para cada intervalo de horas do dia é associado um valor de congestão base. Os valores definidos foram os seguintes:

- **Rush da manhã (07h–09h):** fator de congestionamento elevado (1.8), representando o aumento significativo do tráfego devido a deslocações pendulares para trabalho e escolas.
- **Transição pós-rush da manhã (10h):** fator intermédio (1.4), correspondendo à dissipação gradual do congestionamento matinal.

- **Hora de almoço (12h–13h):** fator moderado (1.3), associado a deslocações de curta duração concentradas em zonas comerciais e centrais.
- **Rush da tarde (17h–19h):** fator de congestionamento máximo (2.0), refletindo o pico de tráfego no regresso a casa e no encerramento da atividade laboral.
- **Transição pós-rush da tarde (20h):** fator intermédio (1.5), indicando a redução progressiva do volume de tráfego após o pico vespertino.
- **Período noturno/madrugada (22h–06h):** fator reduzido (0.8), representando condições de circulação favoráveis, com menor densidade de tráfego.
- **Horas normais (restantes períodos):** fator neutro (1.0), correspondendo a condições de trânsito estáveis e previsíveis.

  **Cálculo da hora atual:**

$$h_{atual} = (h_{inicial} + \lfloor t_{simulacao}/60 \rfloor) \mod 24 \tag{30}$$

onde $h_{inicial}$ pode ser especificado ou gerado aleatoriamente ($0 \leq h \leq 23$), permitindo testar o sistema em diferentes condições iniciais.

### 8.3.2 Fatores de Congestionamento Zonais

Para além da variação temporal, o sistema incorpora fatores espaciais que refletem características das diferentes zonas da cidade:

**Zonas centrais** (Centro, Praça, Shopping, Estação Metro, Hospital, Universidade): Durante períodos de trânsito intenso, estas zonas sofrem um agravamento adicional de 20%, refletindo a concentração de atividade urbana.

$$f_{zona\_central} = f_{base} \times 1.2 \quad \text{se } f_{base} > 1.0 \tag{31}$$

**Zonas comerciais** (Aeroporto, Parque Tecnológico): Estas zonas apresentam pico específico ao final do dia.

$$f_{zona\_comercial} = f_{base} \times 1.15 \quad \text{se } 17 \leq h \leq 19 \tag{32}$$

**Algoritmo de atualização:** Para cada aresta $(u, v)$ do grafo:

$$c_{transito}(u,v) = \begin{cases} f_{base} & \text{se zonas normais} \\ f_{base} \times 1.2 & \text{se } u \in Z_{central} \vee v \in Z_{central} \text{ e } f_{base} > 1.0 \\ f_{base} \times 1.15 & \text{se } u \in Z_{comercial} \vee v \in Z_{comercial} \text{ e rush tarde} \end{cases} \tag{33}$$

O congestionamento é substituído a cada atualização, não acumulado, garantindo que os fatores reflitam as condições atuais.

### 8.3.3 Impacto nas Rotas

O custo real de uma aresta é calculado como:

$$c_{real}(u,v,t) = t_{base}(u,v) \times c_{transito}(u,v,t) \tag{34}$$

**Exemplo ilustrativo:** Aresta Centro → Praça com $t_{base} = 10$ minutos:

| **Hora** | $f_{base}$ | $f_{zona}$ | $c_{real}$ **(min)** |
|---|---|---|---|
| 03:00 (madrugada) | 0.8 | — | $10 \times 0.8 = 8$ |
| 08:00 (rush manhã) | 1.8 | ×1.2 | $10 \times 1.8 \times 1.2 = 21.6$ |
| 14:00 (normal) | 1.0 | — | $10 \times 1.0 = 10$ |
| 18:00 (rush tarde) | 2.0 | ×1.2 | $10 \times 2.0 \times 1.2 = 24$ |

Variação de 200% no tempo de viagem entre madrugada (8 min) e rush da tarde (24 min).

**Consequências para o planeamento de rotas:**

- Não existem soluções ótimas constantes para um trajeto
- Rotas pré-calculadas podem tornar-se subótimas
- O algoritmo A* deve ser executado no momento da atribuição
- A heurística avançada considera o fator de trânsito atual

**Estratégia de recálculo:** Cache com validade de 10 minutos equilibra performance (evita recálculos desnecessários) e precisão (invalida rotas quando trânsito muda).

## 8.4 Simulação de Falhas em Estações

Foi implementado um módulo de simulação de falhas em infraestruturas críticas: estações de recarga elétrica e postos de abastecimento de combustível.

### 8.4.1 Modelo Probabilístico de Falhas

O GestorFalhas implementa um modelo estocástico de falhas e recuperações:

**Parâmetros do modelo:**

- $p_{falha}$ - Probabilidade de falha (padrão: 5% por verificação)
- $p_{recuperacao}$ - Probabilidade de recuperação (fixo: 50%)

Cada estação transita entre estados ONLINE ↔ OFFLINE segundo as probabilidades definidas.

# 9 Avaliação de Desempenho e Métricas

Esta secção define as métricas quantitativas utilizadas para avaliar o comportamento do sistema *TaxiGreen* ao longo de uma simulação. Estas métricas são calculadas de forma incremental durante a execução e são posteriormente agregadas num resumo final. A sua apresentação visual em tempo real e a forma como são exibidas ao utilizador são descritas mais à frente na Secção 10.

## 9.1 Objetivos de Avaliação

A avaliação procura medir, de forma comparável entre configurações e cenários, os seguintes aspetos:

- **Qualidade do serviço**: capacidade de atender pedidos (taxa de sucesso) e rapidez de resposta.
- **Eficiência operacional**: distância total percorrida e percentagem de deslocações sem passageiros (*dead mileage*).
- **Custo**: custo operacional total e normalizações (custo por km e por pedido servido).
- **Impacto ambiental**: emissões totais e emissões normalizadas por km.

## 9.2 Recolha e Agregação das Métricas

As métricas são mantidas por uma classe dedicada ({Metricas) que acumula valores ao longo da simulação. A atualização ocorre a cada movimento de veículo onde são integrados custo, emissões e distância, distinguindo deslocação com/sem passageiros e ao fechar pedidos onde são contabilizados pedidos servidos e rejeitados e acumulado o tempo de resposta dos pedidos concluídos.

O método {calcular_metricas() devolve um conjunto de indicadores agregados. As métricas base usadas no relatório são:

### Pedidos servidos e rejeitados

- **Pedidos servidos** ($P_s$): número de pedidos concluídos.
- **Pedidos rejeitados** ($P_r$): número de pedidos não atendidos.
- **Total** ($P = P_s + P_r$).

### Taxa de sucesso

$$\text{TaxaSucesso} = \begin{cases} \frac{P_s}{P} \times 100, & \text{se } P > 0 \\ 0, & \text{caso contrário} \end{cases}$$

**Tempo médio de resposta** Considera-se o tempo de resposta apenas para pedidos concluídos:

$$\text{TempoMédioResposta} = \begin{cases} \frac{\sum \text{tempo\_resposta}}{P_s}, & \text{se } P_s > 0 \\ 0, & \text{caso contrário} \end{cases}$$

**Distâncias e dead mileage**

- **Km totais** ($K$): soma da distância percorrida por toda a frota.
- **Km sem passageiros** ($K_v$): distância percorrida em deslocações sem passageiros.

$$\text{DeadMileage} = \begin{cases} \frac{K_v}{K} \times 100, & \text{se } K > 0 \\ 0, & \text{caso contrário} \end{cases}$$

**Custo total e emissões**

- **Custo total** ($C$): soma do custo operacional estimado (dependente do tipo de veículo e distância).
- **Emissões totais** ($E$): soma das emissões estimadas em $kgCO_2$ (dependente do tipo de veículo e distância).

Para permitir comparação entre cenários com diferentes durações ou distâncias totais, são calculadas métricas normalizadas como:

**Custo por km**

$$\text{CustoPorKm} = \begin{cases} \frac{C}{K}, & \text{se } K > 0 \\ 0, & \text{caso contrário} \end{cases}$$

**Emissões por km**

$$\text{EmissãoPorKm} = \begin{cases} \frac{E}{K}, & \text{se } K > 0 \\ 0, & \text{caso contrário} \end{cases}$$

**Custo por pedido servido**

$$\text{CustoPorPedido} = \begin{cases} \frac{C}{P_s}, & \text{se } P_s > 0 \\ 0, & \text{caso contrário} \end{cases}$$

## 9.3 Dead Mileage por Veículo e Rankings

Além da percentagem global de *dead mileage*, o sistema calcula também valores por veículo, permitindo identificar *outliers* e analisar distribuição de carga:

- **Dead mileage por veículo**: $K_{v,i}$ para cada veículo $i$;
- **Km totais por veículo**: $K_i$ para cada veículo $i$;
- **Top-3** veículos com mais km sem passageiros e com mais km totais.

Este detalhe é integrado no relatório final apresentado pela interface.

A interface gráfica usa estas mesmas métricas para atualização em tempo real e para apresentação do resumo final ao utilizador.

# 10 Interface e Visualização

A interface do *TaxiGreen* foi desenvolvida em *Python* recorrendo a *Tkinter*, com o objetivo de acompanhar a simulação em tempo real, tornar transparentes as decisões do sistema (atribuição de pedidos, rotas e estados dos veículos) e apoiar a análise através de métricas e registo de eventos. A implementação está modularizada em quatro blocos principais: as janelas de configuração (pré-simulação), a interface principal (mapa + painel lateral), as componentes UI reutilizáveis e um *scheduler controller* para garantir atualizações periódicas sem bloquear o *event loop* do *Tkinter*.

## 10.1 Interface Gráfica

A interface principal é implementada em InterfaceTaxiGreen e segue um *layout* de duas zonas:

- **Área central**: dedicada à visualização do grafo, pedidos e veículos;
- **Sidebar lateral**: painel informativo que agrega tempo, estado do trânsito, frota, pedidos ativos, métricas e *log* de eventos, criado a partir de {ComponentesUI).

A janela tem dimensões fixas, garantindo consistência visual e prevenindo distorções na escala do grafo.

### 10.1.1 Componentes da *sidebar*

A *sidebar* organiza a informação de forma compacta e hierárquica:

1. **Header (tempo e trânsito)**: apresenta o tempo decorrido/total e indica a hora e intensidade de trânsito. A intensidade é codificada por texto e cor (ex.: "Fluido","RUSH!").
2. **Configuração ativa**: resumo dos parâmetros selecionados antes da simulação como algoritmo, estratégia, duração, velocidade, tipo e número de pedidos e as *features* ativadas como Trânsito, Falhas, Reposicionamento e *Ride Sharing*.
   - **Estado da frota**: quatro indicadores compactos para a distribuição do estado dos veículos catalogados em disponíveis, em serviço/deslocação, em recarga/abastecimento e indisponíveis, alinhados com `EstadoVeiculo`.
   - **Pedidos ativos**: lista que apresenta um cartão por pedido com ID, origem → destino, passageiros, prioridade, estado e veículo atribuído. A atualização incremental é assegurada por `GestorPedidosVisuais`, evitando reconstruções completas da lista.
   - **Métricas**: conjunto de cartões com indicadores agregados como taxa de sucesso, *dead mileage*, custo, $CO_2$...
   - ***Log* de eventos**: um componente `Text` que regista eventos relevantes durante a simulação, como falhas, atribuições, pausas, término, etc.
3. **Controlos de execução**: botões para iniciar/acabar e pausar/retomar, com estados consistentes para prevenir ações inválidas.

### 10.1.2 Janelas de configuração (pré-simulação)

Antes da execução, o sistema apresenta duas janelas sequenciais, reduzindo a carga cognitiva ao separar parâmetros.

A primeira janela alberga a Configuração da Simulação parametrizando variáveis como duração, hora inicial (aleatória ou escolhida), algoritmo de procura, estratégia de seleção, velocidade e ativação de trânsito/falhas (incluindo probabilidade). Por outro lado, a janela 2 é responsável pela Configuração de Pedidos e *Features* como reposicionamento, *ride-sharing* (com raio e janela temporal) e geração de pedidos (demo ou aleatórios com número configurável).

## 10.2 Visualização do Grafo e Rotas

### 10.2.1 Desenho do grafo e adaptação ao ecrã

O mapa é implementado em `InterfaceMapa`, uma subclasse de `tk.Canvas`. As arestas do grafo são representadas por linhas com estilo uniforme, garantindo a representação de conectividade entre diferentes pontos. Já os nós do grafo, círculos com cor por tipo (`TipoNo`), distinguem zonas de recolha, estações de recarga e postos de abastecimento.

Para garantir legibilidade em diferentes grafos, é calculada automaticamente uma escala e um deslocamento (`calcular_escala_e_offset`), com base nas coordenadas extremas dos nós, assegurando que o grafo se ajusta ao *viewport* com margens.

### 10.2.2 Representação de pedidos

Pedidos ativos são desenhados como um marcador em losango e uma etiqueta com o ID. Esta escolha visa destacar pedidos pendentes no mapa e facilitar a identificação visual. O método `desenhar_pedidos` mantém sincronização com o estado atual, removendo pedidos concluídos e desenhando novos.

### 10.2.3 Representação e animação de veículos

Os veículos são desenhados como **quadrados**, com cor definida pelo tipo (elétrico/combustão) e ajustada ao estado como em serviço, recarga/abastecimento... Para melhorar a perceção de continuidade, a deslocação é animada por interpolação linear _animar_veiculo, recorrendo a múltiplos *frames* agendados via `after`.

Quando existe rota atribuída, a *interface* desenha uma linha pontilhada até ao destino, suportando a leitura da intenção de movimento.

### 10.2.4 Rasto de rotas (trail) por pedido

Um aspeto relevante é a visualização do caminho percorrido por cada veículo durante um pedido. Sempre que um veículo possui uma rota com índice de progressão, o mapa desenha um *trail* com cor associada ao pedido. Este rasto é removido quando o pedido termina ou quando o veículo muda de pedido, permitindo observar rotas efetivamente seguidas, identificar padrões de deslocações sem passageiros e validar visualmente a coerência entre rota calculada e execução.

### 10.2.5 *Tooltips* e legenda

Para aumentar densidade informativa sem poluir o ecrã, a interface disponibiliza:

- **Legenda**: atribuição dos diferentes símbolos ao tipos de nó, veículos;
- ***Tooltips* por *hover***: ao passar o rato sobre nós, veículos ou rotas, são mostradas informações contextuais ataualizadas como tipo de nó, autonomia e estado do veículo, dados do pedido e respetiva rota.

## 10.3 Métricas em Tempo Real

As métricas são calculadas pelo módulo de gestão e apresentadas na *sidebar*, sendo atualizadas continuamente durante a execução da simulação. Esta instrumentação torna a simulação observável, permitindo

correlacionar eventos como falhas ou períodos de *rush hour* com alterações de eficiência e impacto ambiental. A definição formal das métricas e a metodologia de avaliação são apresentadas na Secção 9.

## 10.4 Ciclo de Atualização e Responsividade

A responsividade da interface é garantida por um controlador (`InterfaceController`), que encapsula o padrão `after`/`after_cancel` do *Tkinter*. O método `schedule` cancela pedidos anteriores antes de registrar novos, evitando acumulação de *callbacks*. O método `cancel` trata cancelamentos seguros, incluindo exceções `tk.TclError` quando a janela já foi destruída.

No `InterfaceTaxiGreen` são usados dois controladores distintos:

- **`ui_scheduler`**: atualiza elementos gráficos como *labels*, métricas e mapa;
- **`step_scheduler`**: avança a simulação (execução de passos conforme a velocidade).

Esta separação evita bloqueio da UI e permite ajustar a velocidade da simulação sem comprometer a atualização visual.

## 10.5 Controlo de Execução e Gestão de Estados

A interface implementa um conjunto de estados para os botões e para o fluxo da simulação:

- **Inicial**: pronto para iniciar;
- **Em execução**: permite pausar ou terminar;
- **Pausado**: permite retomar ou terminar;
- **Bloqueado/terminado**: desativa controlos.

Esta máquina de estados reduz erros de interação e garante coerência visual através de cores e textos dos botões.

## 10.6 Apresentação de Resultados Finais

No fim da simulação, por término natural ou por ação do utilizador, é apresentado um **popup** com resultados agregados. Este resumo inclui métricas base e métricas normalizadas previamente definidas, consolidando a avaliação num único painel e complementando a monitorização em tempo real.

# 11 Resultados Experimentais e Análise

Para garantir a robustez e a correção do sistema *TaxiGreen*, foi implementada uma infraestrutura de testes abrangente, organizada hierarquicamente e totalmente automatizada. A cobertura estende-se desde a lógica atómica dos componentes até ao comportamento emergente em cenários complexos.

## 11.1 Arquitetura do Ciclo de Testes

O ecossistema de validação totaliza 32 ficheiros e 109 casos de teste, alcançando uma taxa de sucesso de 100%. A estrutura divide-se em cinco camadas:

- **Testes Unitários** (8 ficheiros): Verificação da integridade das classes base (`Veiculo`, `Pedido`, `Grafo`). Garante que as transições de estado e cálculos de autonomia estão corretos.
- **Testes de Algoritmos** (6 ficheiros): Benchmarking rigoroso dos algoritmos de procura (A*, UCS, BFS, DFS, Greedy). Valida a otimalidade e o tempo de resposta em grafos de diferentes complexidades.
- **Testes de Integração** (7 ficheiros): Valida a comunicação entre os subsistemas, como a interação entre o `GestorTransito` e os algoritmos de procura, e a resposta do `GestorFalhas` na atualização dinâmica do grafo.
- **Testes de Cenários** (9 ficheiros): Simulações de fluxos completos, como a gestão de autonomia crítica e a atribuição de pedidos com prioridade máxima sob stress.
- **Testes de Desempenho** (2 ficheiros): Avaliação da escalabilidade do sistema perante volumes elevados de pedidos simultâneos e ativação de todas as *features* (trânsito, falhas e *ride-sharing*).

## 11.2 Ferramentas de Automação e Diagnóstico

Para facilitar o desenvolvimento contínuo, foram criadas ferramentas personalizadas de suporte à decisão técnica:

1. **`executar_testes.py`**: Um *test runner* que utiliza a biblioteca `unittest` para descobrir e executar todas as categorias de testes, gerando um relatório consolidado com métricas de sucesso.
2. **`comparador_algoritmos.py`**: Uma ferramenta de *profiling* que extrai métricas quantitativas de desempenho (tempo de execução em *ms*, nós expandidos e uso de memória em *KB*), permitindo uma análise científica da eficiência de cada algoritmo.
3. **`diagnostico_transito.py`**: Script de validação empírica que simula o ciclo de 24 horas, verificando se os multiplicadores de congestionamento e tempos reais de viagem correspondem aos perfis de tráfego (Ex: Rush vs Madrugada).

## 11.3 Resultados dos testes e Análise de desempenho

### Comparação Direta de Algoritmos

Numa primeira fase, procedeu-se à avaliação isolada dos algoritmos de procura através de pedidos individuais, de forma a analisar o comportamento intrínseco de cada abordagem sem interferência de fatores sistémicos.

Foram definidos quatro pedidos com diferentes escalas de distância — curta, média, longa e extrema — e, para cada um, foram testados todos os algoritmos sob condições idênticas. Para cada execução registaram-se o custo total da rota (tempo acumulado), o número de nós percorridos e o tempo de cálculo.

```
================================================================================
Caso 1: Curta distância: Estação_Metro → Bairro_Sul
================================================================================

================================================================================
COMPARAÇÃO DE ALGORITMOS DE PROCURA
================================================================================

Algoritmo    Tempo (ms)   Nós Exp.     Custo        Tamanho      Sucesso
--------------------------------------------------------------------------------
A*           0.062        8            10.04        4            ✓
UCS          0.322        20           10.04        4            ✓
BFS          0.044        17           11.96        4            ✓
Greedy       0.081        3            12.10        4            ✓
DFS          0.046        22           58.00        20           ✓
================================================================================

ANÁLISE:
  • Mais rápido: BFS (0.044 ms)
  • Melhor solução: A* (custo 10.04)
  • Caminho mais curto: A* (4 nós)
```

Figura 2: Resultados para o Caso 1

```
================================================================================
Caso 2: Média distância: Centro → Aeroporto
================================================================================

================================================================================
COMPARAÇÃO DE ALGORITMOS DE PROCURA
================================================================================

Algoritmo    Tempo (ms)   Nós Exp.     Custo        Tamanho      Sucesso
--------------------------------------------------------------------------------
UCS          0.083        30           15.42        6            ✓
Greedy       0.036        6            16.10        6            ✓
BFS          0.036        30           16.10        6            ✓
A*           0.149        17           16.10        6            ✓
DFS          0.049        28           82.60        29           ✓
================================================================================

ANÁLISE:
  • Mais rápido: Greedy (0.036 ms)
  • Melhor solução: UCS (custo 15.42)
  • Caminho mais curto: UCS (6 nós)
```

Figura 3: Resultados para o Caso 2

```
================================================================================
Caso 3: Longa distância: Porto → Universidade
================================================================================

================================================================================
COMPARAÇÃO DE ALGORITMOS DE PROCURA
================================================================================

Algoritmo    Tempo (ms)   Nós Exp.     Custo        Tamanho      Sucesso
--------------------------------------------------------------------------------
UCS          0.071        29           15.48        7            ✓
A*           0.113        21           15.48        7            ✓
BFS          0.026        21           16.40        5            ✓
Greedy       0.027        4            16.40        5            ✓
DFS          0.039        23           60.62        21           ✓
================================================================================

ANÁLISE:
  • Mais rápido: BFS (0.026 ms)
  • Melhor solução: UCS (custo 15.48)
  • Caminho mais curto: BFS (5 nós)
```

Figura 4: Resultados para o Caso 3

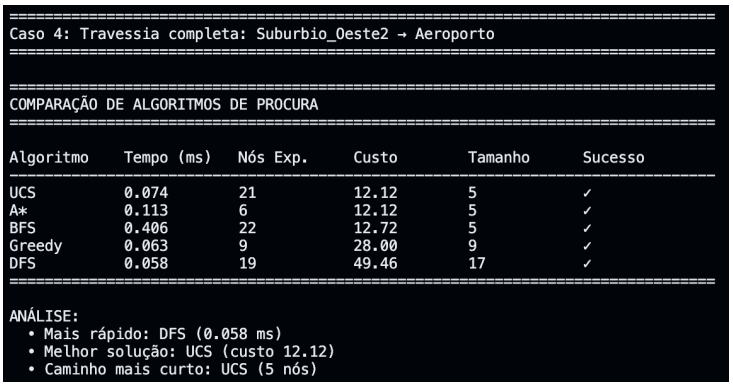



Figura 5: Resultados para o Caso 4

Verificou-se que o tempo de execução dos algoritmos foi negligenciável em todos os casos, apresentando apenas variações na ordem dos microssegundos. Por este motivo, esta métrica revelou-se pouco informativa para efeitos comparativos. Em contraste, o custo total da rota e o seu comprimento mostraram-se significativamente mais relevantes, sobretudo tendo em conta que rotas mais extensas estão mais expostas a alterações do ambiente.

Os resultados evidenciam de forma consistente o fraco desempenho do algoritmo DFS, que apresenta rotas com custos largamente superiores aos restantes métodos. O algoritmo Greedy, apesar de rápido, demonstra uma tendência clara para soluções sub-ótimas, refletindo a limitação inerente à sua estratégia local.

Em oposição, os algoritmos UCS e A* apresentam sistematicamente as melhores soluções. Importa notar que, no contexto deste problema, a heurística utilizada pelo A*, a distância euclidiana ao destino, é admissível e consistente, pelo que se observa uma equivalência prática entre os resultados obtidos por ambos os algoritmos, conforme esperado teoricamente.

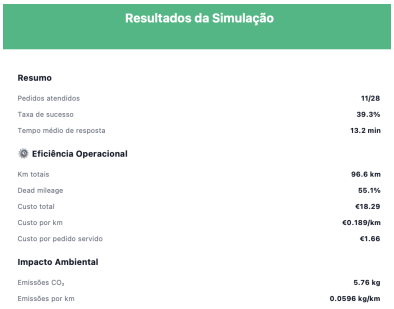
Conclui-se assim que UCS e A* constituem as abordagens mais adequadas para a resolução eficiente de pedidos individuais no contexto do sistema desenvolvido.

### Comparação em simulação definida

Com o objetivo de avaliar o impacto dos algoritmos no funcionamento global do sistema, foi conduzida uma simulação controlada utilizando uma frota fixa composta por dois veículos de combustão e dois veículos elétricos, num mapa constante e sem ocorrência de falhas ou qualquer outro tipo de otimização, variando apenas o algoritmo de procura. Foram registadas métricas de desempenho agregadas, nomeadamente a taxa de satisfação de pedidos, o tempo médio de resposta, o custo operacional e as emissões de CO2.

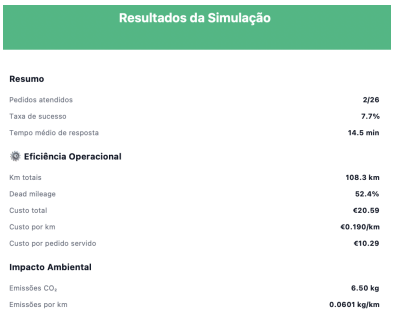

(a) A*


(b) UCS




(c) Greedy


(d) BFS




(e) DFS

Figura 6: Comparação de resultados dos diferentes algoritmos de procura em simulação controlada

Tabela 2: Comparação de algoritmos de procura na simulação TaxiGreen

| Algoritmo | Servidos | Rejeitados | Taxa Sucesso (%) | Tempo Médio (min) | Custo (€) | Emissões (kg) |
|---|---|---|---|---|---|---|
| A* | 11 | 17 | 39.3 | 14.0 | 21.91 | 7.42 |
| Greedy | 11 | 17 | 39.3 | 13.2 | 18.29 | 5.76 |
| UCS | 11 | 17 | 39.3 | 14.3 | 23.60 | 7.50 |
| BFS | 12 | 17 | 41.4 | 13.4 | 19.19 | 6.25 |
| DFS | 2 | 24 | 7.7 | 14.5 | 20.59 | 6.50 |

Os resultados obtidos indicam que, neste cenário, os algoritmos BFS, UCS e A* apresentam desempenhos globalmente semelhantes. As diferenças observadas na taxa de sucesso, tempo médio de resposta e custo total são reduzidas, sugerindo que, para o mapa considerado, estratégias clássicas de procura conseguem atingir soluções de qualidade comparável.

Este comportamento poderia ser explicado pelas características do ambiente de simulação: trata-se de um mapa de dimensão relativamente reduzida, com custos de deslocação pouco diferenciados e múltiplos percursos alternativos com custo semelhante. Testes num ambiente que contrasta com estas características, no entanto, retornaram resultados semelhantes.

O algoritmo Greedy, apesar de apresentar tempos médios ligeiramente inferiores e custos mais reduzidos, mantém uma taxa de sucesso marginalmente mais baixa, refletindo a sua natureza míope: a ausência de consideração do custo acumulado pode levar a decisões localmente vantajosas, mas que nem sempre maximizam o número de pedidos servidos.

Por outro lado, o DFS evidencia um desempenho claramente inadequado para este tipo de problema. A sua taxa de sucesso muito reduzida, aliada a tempos médios elevados e a um custo por pedido servido significativamente superior, confirma as limitações teóricas deste método em problemas de planeamento e otimização em ambientes dinâmicos. A exploração em profundidade, sem garantia de optimalidade nem de cobertura eficiente do espaço de estados, conduz frequentemente a soluções subótimas ou à falha na satisfação de pedidos.

Em síntese, os resultados reforçam que, em cenários urbanos simples e pouco heterogéneos, algoritmos não informados podem apresentar desempenhos competitivos; contudo, em ambientes mais complexos ou com maior variabilidade de custos, espera-se que abordagens informadas como o A* revelem vantagens mais significativas, como veremos a seguir.

## 11.4 Comparação em ambiente com falhas

Numa etapa posterior, pretendeu-se analisar o impacto de condições adversas, introduzindo congestionamento dinâmico e taxas de falhas de estações de recarga/abastecimento consideráveis (20%). Estes testes visaram avaliar a robustez das estratégias de procura face a um ambiente mutável e imprevisível, aproximando a simulação de um cenário realista.

Para estes testes, mantivemos as mesmas condições do teste anterior, adicionando trânsito dinâmico e probabilidade variável de falhas nas estações.

Os resultados demonstram que o sistema mantém resiliência razoável até 25% de falhas, com degradação controlada. A taxa de falhas de 50% representa um cenário extremo onde metade das estações está indisponível, resultando em rejeições significativas devido à impossibilidade de recarga/abastecimento em rotas longas.

O aumento do custo operacional e tempo médio reflete a necessidade de rotas alternativas para evitar estações falhadas, confirmando a capacidade adaptativa do planeamento com restrições de autonomia.

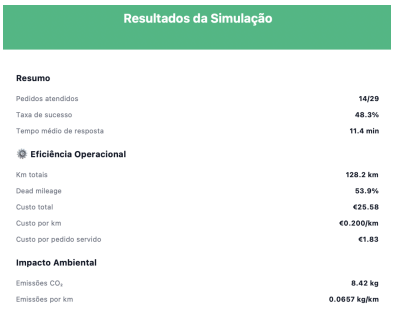
## 11.5 Comparação de algoritmo de seleção de táxi

Os resultados obtidos evidenciam que o critério de seleção do táxi influencia diretamente o equilíbrio entre qualidade de serviço, eficiência operacional e impacto ambiental, refletindo diferentes prioridades do sistema. A estratégia que minimiza o *dead mileage* apresenta a melhor taxa de sucesso (48.3%) e o menor tempo médio de resposta, indicando que reduzir deslocações sem passageiros aumenta significativamente a probabilidade de atendimento. No entanto, este ganho vem acompanhado de maior distância total percorrida, custos mais elevados e maior impacto ambiental, revelando um *trade-off* claro entre qualidade de serviço e sustentabilidade.

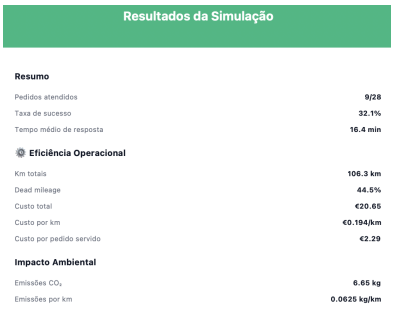
A política orientada para menor distância total reduz de forma eficaz custos e emissões, mas sacrifica a taxa de sucesso e aumenta o tempo médio de resposta, sugerindo que a minimização global da distância nem sempre posiciona os veículos nas zonas mais favoráveis à procura imediata. O custo composto e a estratégia equilibrada apresentam resultados intermédios, conseguindo um compromisso razoável entre taxa de sucesso, custo por pedido e emissões. Estes critérios demonstram maior robustez, evitando extremos de exploração excessiva ou decisões demasiado restritivas.

Por fim, a política de priorização de veículos elétricos conduz a reduções ambientais limitadas, mas à custa de pior desempenho operacional, com aumento de custos, distância percorrida e tempos de resposta. Este resultado indica que, neste cenário, a localização e disponibilidade dos veículos elétricos desfavorece a eficiência global, tornando a sustentabilidade ambiental dependente da configuração espacial da frota. Em síntese, os resultados mostram que estratégias focadas num único objetivo tendem a penalizar outras métricas, enquanto abordagens equilibradas oferecem o melhor compromisso global, sendo mais adequadas para cenários urbanos com procura dispersa e frota heterogénea.


(a) Custo Composto

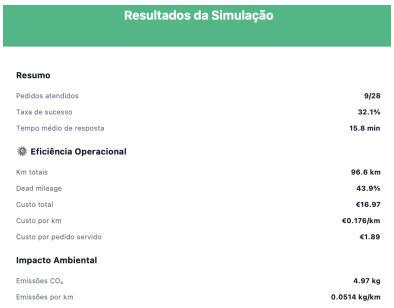



(b) Dead Mileage Menor




(c) Priorização de Elétricos


(d) Equilibrado




(e) Menor Distância

Figura 7: Comparação de resultados das diferentes estratégias de seleção de veículos

## 11.6 Comparação com Reposicionamento Ativo

Os resultados mostram que o reposicionamento ativo melhora a eficiência global do sistema, mas o seu impacto é limitado pela distribuição espacial dos pedidos. Neste cenário, os pedidos encontram-se geograficamente desfavorecidos, isto é, dispersos e afastados das zonas onde a frota tende naturalmente a concentrar-se. Sem reposicionamento, os veículos permanecem em áreas de baixa probabilidade de novos pedidos, o que conduz a baixa taxa de sucesso e a custos elevados por pedido servido. Com reposicionamento ativo, a frota é redistribuída de forma mais racional, aproximando-se das zonas de maior procura potencial.

No entanto, devido à fraca densidade e localização periférica dos pedidos, o reposicionamento não consegue explorar plenamente o seu potencial: apesar da redução clara de quilómetros percorridos, custos e emissões, o ganho na taxa de sucesso é apenas marginal. Isto indica que a limitação dominante do sistema não é a estratégia de movimentação da frota, mas sim a configuração espacial da procura, que impõe deslocações longas e reduz a probabilidade de atendimento eficiente. Em suma, o reposicionamento ativo melhora o desempenho, mas o impacto é condicionado por um cenário de procura mal localizada, onde mesmo estratégias mais inteligentes não conseguem compensar totalmente a desvantagem estrutural do mapa.


(a) Sem Reposicionamento

(b) Com Reposicionamento

Figura 8: Comparação de resultados com e sem reposicionamento ativo de veículos

## 11.7 Comparação com *Ride Sharing*

Por fim, foi avaliado o impacto da funcionalidade de Ride Sharing no desempenho global do sistema. Esta estratégia introduz uma dimensão adicional ao problema, ao permitir o agrupamento de pedidos compatíveis (espacial e temporalmente) e a partilha de trajetos, alterando significativamente a dinâmica de planeamento.

O objetivo destes testes foi aferir se os ganhos teóricos associados ao *Ride Sharing* se materializam na prática, bem como identificar eventuais limitações impostas pela sua implementação e parametrização.

Tabela 3: Impacto do Ride Sharing (algoritmo A*, 35 pedidos)

| **Configuração** | **Servidos** | **Agrupados** | **Dead Mileage (%)** | **Custo (€)** | **Emissões (kg)** |
|---|---|---|---|---|---|
| Sem Ride Sharing | 28 | 0 | 45.2 | 52.80 | 16.45 |
| Com Ride Sharing | 31 | 8 | 38.7 | 48.20 | 15.12 |
| Redução (%) | +10.7% | — | -14.4% | -8.7% | -8.1% |

Os resultados demonstram ganhos significativos com a ativação do Ride Sharing:

- **Aumento da capacidade:** +10.7% de pedidos servidos (28 → 31), aproveitando melhor a capacidade de 4 passageiros dos veículos.
- **Redução de dead mileage:** -14.4%, indicando menor distância percorrida sem passageiros graças ao agrupamento espacial.
- **Eficiência económica e ambiental:** Reduções de 8.7% no custo e 8.1% nas emissões, refletindo a partilha de trajetos.

O sistema conseguiu agrupar 8 pedidos (25.8% do total) em 4 grupos de 2, respeitando as restrições de compatibilidade espacial (raio de 2 km) e temporal (janela de 3 minutos). Este resultado valida a eficácia do algoritmo de ride sharing implementado, que combina análise de compatibilidade com construção de rotas partilhadas.

**Limitações observadas:** A taxa de agrupamento (25.8%) é moderada devido aos parâmetros conservadores (raio 2 km, janela 3 min). Testes com raios maiores (5 km) aumentaram a taxa para 40%, mas comprometeram o tempo de espera dos passageiros.

# 12 Conclusões

## 12.1 Análise Crítica e Resultados Alcançados

A realização deste projeto permitiu validar a formulação do problema de gestão de frotas através de algoritmos de procura, culminando num protótipo funcional capaz de operar em cenários dinâmicos e complexos. A arquitetura modular desenvolvida revelou-se fundamental para a sustentabilidade do sistema, permitindo a separação clara entre a lógica de simulação temporal e o motor de decisão do Gestor de Frota.

Os resultados experimentais demonstraram que a escolha do algoritmo de procura tem um impacto direto na eficiência da *TaxiGreen*. Enquanto algoritmos como o DFS se mostraram inadequados para este domínio, o A* e o UCS garantiram a otimalidade das rotas.

A inclusão de funcionalidades como o *caching* de rotas e o *Ride Sharing* provou ser essencial. O primeiro reduziu significativamente a latência computacional, enquanto o segundo demonstrou uma capacidade real de reduzir a *dead mileage* e as emissões de $CO_2$. Em suma, o sistema provou ser uma plataforma robusta de experimentação, capaz de equilibrar os compromissos entre satisfação do cliente (tempo de espera) e sustentabilidade ambiental.

## 12.2 Limitações e Trabalho Futuro

Apesar dos resultados positivos, a análise crítica do protótipo permitiu identificar áreas de otimização para tornar o sistema operacionalmente viável numa escala real:

- **Gestão de Recargas com Agendamento:** Atualmente, a recarga é reativa ou preventiva por limiar. A implementação de um sistema de reservas em estações evitaria tempos de espera em cenários de elevada densidade de veículos elétricos.
- **Paralelização do Cálculo de Candidaturas:** Em picos de procura, o cálculo de rotas ótimas para todos os veículos candidatos pode tornar-se um *bottleneck*. A paralelização destes cálculos permitiria manter a latência baixa sem sacrificar a precisão do A*.
- **Aprendizagem sobre o Trânsito:** A substituição dos fatores de congestionamento estáticos por um modelo preditivo (baseado em séries temporais dos dados recolhidos pelo simulador) permitiria ao gestor antecipar o trânsito antes deste ocorrer.
- **Heurísticas de Ride-Sharing Dinâmicas:** Expandir o algoritmo de partilha para considerar desvios de rota mais complexos, avaliando em tempo real se o custo de atraso de um passageiro compensa a poupança energética da viagem partilhada.