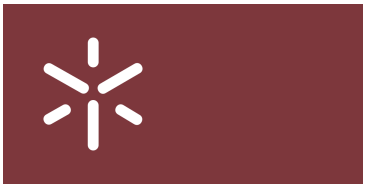

Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Inteligência Artificial

Ano Letivo de 2025/2026

Grupo 06

João Delgado
A106836

Nelson Mendes
A106884

Simão Mendes
A106928

Tomás Machado
A104186

January 13, 2026

IA

Índice

1. Avaliação por Pares

De acordo com as orientações fornecidas no enunciado, o grupo realizou uma avaliação interna do contributo de cada elemento no desenvolvimento do trabalho. Esta análise teve por objetivo **refletir**, de forma justa e transparente, **o empenho, a participação e o impacto de cada membro** na elaboração do instrumento.

Atribuiu-se a cada elemento um delta, que corresponde a uma parcela a somar (ou subtrair) ao valor desta componente avaliativa. Os **deltas podem ser positivos, nulos ou negativos**, devendo, no seu conjunto, totalizar 0.00, garantindo assim um equilíbrio global dentro do grupo.

A tabela abaixo **apresenta os deltas** definidos coletivamente para cada membro, refletindo o contributo relativo de cada um ao longo do desenvolvimento do trabalho.

N.º	Nome	DELTA
A106836	João Delgado	0
A106884	Nelson Mendes	0
A106928	Simão Mendes	0
A104186	Tomás Machado	0

Tabela 1: Avaliação por Pares

2. Descrição do Problema

O **projeto *TaxiGreen*** visa a otimização da gestão de uma **frota heterogénea** (veículos de combustão e elétricos) num ambiente urbano real. O problema central consiste na alocação eficiente de recursos a tarefas dinâmicas, num grafo de grande escala, sob restrições de autonomia e com objetivos conflituosos de minimização de tempo, custos operacionais e impacto ambiental.

De acordo com a metodologia de Resolução de Problemas abordada na Unidade Curricular, o problema é formalmente definido pela quíntupla (S, S_0, A, T, C):

2.1. Representação do Estado (S)

O estado do sistema num instante t é uma representação complexa que agrega as variáveis do ambiente e dos agentes:

- **Estado da Frota:** Para cada veículo v, define-se a sua localização (nó $n \in V$), carga/combustível atual ($E_{\{curr\}}$), estado operacional (`LIVRE`, `OCUPADO` ou `A_CARREGAR`), capacidade de passageiros e **tipo de motorização** (diferenciando custos e necessidades de recarga);
- **Estado do Ambiente:** Inclui a lista de pedidos pendentes, P, onde cada pedido especifica **prioridade**, **prazo** e **preferência ambiental**. Inclui também as condições dinâmicas das arestas (tráfego), que variam consoante a zona (Central ou Periférica).

2.2. Estado Inicial (S_0)

Corresponde à configuração da simulação em $t = 0$, onde a frota é instanciada em nós aleatórios ou predefinidos, com níveis iniciais de energia e sem pedidos ativos. O grafo G encontra-se carregado com os dados estáticos da **API do `OpenStreetMap`.**

2.3. Operadores e Espaço de Ações (A)

As transições entre estados são governadas por operadores que transformam o estado físico e lógico dos agentes:

- **`Move(v, n_dest)`**: Deslocação do veículo para um nó adjacente.
  - **Pré-condição**: O nó $n_{\{dest\}}$ deve ser adjacente ao atual, o veículo v deve ter energia suficiente para o trajeto e não pode estar em estado de carregamento.
  - **Efeito**: Atualização da localização para $n_{\{dest\}}$, redução da energia $E_{\{curr\}}$ e incremento do tempo global t.

- **`Pickup(v, p)`**: Recolha do passageiro p pelo veículo v.
  - **Pré-condição**: Veículo e passageiro devem estar no mesmo nó, o veículo deve estar em estado `LIVRE` e possuir capacidade disponível.
  - **Efeito**: Alteração do estado do veículo para `OCUPADO` e do estado do pedido para `EM_TRANSPORTE`.
- **`Dropoff(v, p)`**: Entrega do passageiro no destino final.
  - **Pré-condição**: O veículo deve estar na localização de destino do pedido p e estar a transportar esse passageiro.
  - **Efeito**: O estado do veículo passa a `LIVRE`, o pedido é removido da lista P e a recompensa é contabilizada na função de custo.
- **`Refuel(v, s)`**: Abastecimento ou carregamento na estação s.
  - **Pré-condição**: O veículo deve estar no nó da estação, esta deve ser compatível com o seu motor e o veículo deve estar `LIVRE`.
  - **Efeito**: O veículo assume o estado `A_CARREGAR` e a energia $E_{\{curr\}}$ é reposta até ao valor máximo.

2.4. Teste de Objetivo (T)

O estado objetivo é alcançado quando o conjunto de pedidos pendentes é vazio ($P = \emptyset$) e todos os passageiros foram transportados com sucesso aos seus destinos dentro dos **horários pretendidos**. Adicionalmente, nenhum veículo deve terminar a operação com autonomia nula em locais não autorizados.

2.5. Função de Custo da Solução (C)

A qualidade da solução é avaliada por uma função multiobjetivo que procura equilibrar os critérios definidos no enunciado:

1. **Tempo de Resposta:** Minimização do tempo de espera e cumprimento de prazos.
2. **Custos Operacionais:** Minimização do consumo energético (eletricidade/combustível).
3. **Sustentabilidade:** Maximização do uso de elétricos e minimização de emissões de `CO2` (via *score* ecológico).

A **função de custo $g(n)$** para a travessia de uma aresta (u, v) pondera estes fatores:

$$\text{Custo}_{\text{total}} = \text{Custo}_{\text{base}} \times M_{\text{clima}} \times M_{\text{trânsito}} \times \text{Fator}_{\text{ecológico}}$$

Para os algoritmos informados **(A* e *Greedy*),** utiliza-se uma **heurística $h(n)$** baseada na **Distância de *Haversine***, garantindo uma estimativa admissível para a navegação no grafo.

Para informações detalhadas sobre a função de custo e heurística, consultar Secção 4.1.

3. Representação da Cidade e os seus Pontos Críticos

A modelação do ambiente operacional concretizou-se através de uma abstração matemática sob a forma de um grafo dirigido $G = (V, E)$, onde V representa o **conjunto de nós** (localizações geográficas) e E o **conjunto de arestas** (segmentos viários). Esta estrutura, definida nas classes `Graph`, `Node` e `Edge`, permite a representação computacional da rede de transportes de todo o **município de Braga**, essencial para a operação dos veículos autónomos.

3.1. Aquisição e Processamento de Dados Geográficos

A construção do grafo, operacionalizada no módulo `Graph_Generator.py`, recorre à biblioteca `osmnx` para extrair a topologia viária do município ("Braga, Portugal") a partir do `OpenStreetMap`. Complementarmente, o sistema integra dados em tempo real sobre postos de abastecimento e estações de carregamento via `Overpass API`, assegurando uma representação fiel da infraestrutura crítica disponível.

3.2. Nós e Classificação Zonal

Os nós, instanciados pela classe `Node`, encapsulam atributos semânticos que transcendem as coordenadas espaciais, fundamentais para a lógica de navegação:

- **Tipologia Funcional:** Os nós possuem classificações como `station` (recolha/entrega), `charge_station` (carregamento elétrico) ou `gas_station` (combustível), permitindo ao sistema identificar os pontos estratégicos para a manutenção da autonomia;
- **Zonamento Dinâmico e Granularidade:** O módulo `ZoneCalc.py` implementa uma lógica de categorização posicional que toma como referência geográfica a **Sé de Braga** (41.5500, −8.4273). O algoritmo define duas macro-áreas com granularidades distintas:
    - **Zona Central (Cidade):** Compreende todos os nós num raio de **3.5 km** a partir da Sé. Devido à maior densidade urbana, esta área é subdividida em **8 setores cardeais** (e.g., `center_northeast`, `center_south`), permitindo um controlo preciso sobre as condições ambientais;
    - **Periferia:** Engloba a área restante do município (além dos **3.5 km**). Por apresentar uma malha viária menos densa, a divisão simplifica-se em **4 setores cardeais** (e.g., `periphery_north`, `periphery_west`).

Esta abordagem híbrida permite aplicar coeficientes de custo (tráfego e clima) mais precisos no centro urbano e mais generalistas nas zonas rurais ou periféricas.

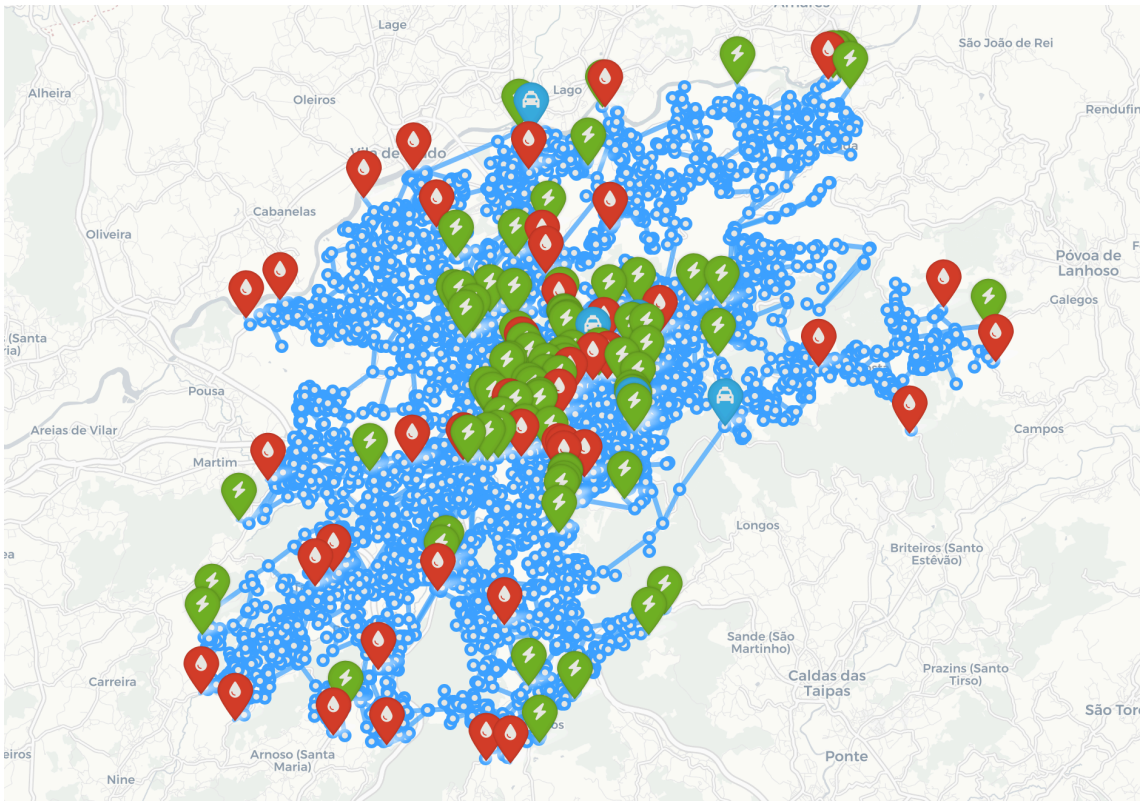

Figura 1: **Representação do grafo do município de Braga**. Apresenta também a localização de todos os veículos (a azul), postos de carregamento (a verde) e postos de combustível (a vermelho).

3.3. Arestas e Custo de Travessia

As arestas, definidas na classe `Edge`, representam os segmentos de via da rede viária de Braga. Cada aresta possui atributos físicos essenciais como: origem, destino, comprimento (em metros) velocidade máxima permitida e sentido de circulação. Uma característica distintiva do sistema ***TaxiGreen*** é o parâmetro `edge.ecology`, um ***score* ecológico** dinamicamente calculado entre 0.0 e 1.0 que quantifica o impacto ambiental de cada segmento viário. Este valor é determinado durante a construção do grafo e considera múltiplos fatores como o tipo de via (e.g., vias residenciais ou pedestres recebem *scores* mais elevados, enquanto autoestradas e vias principais são penalizadas), presença de infraestruturas cicláveis, e limites de velocidade reduzidos. O sistema aplica ainda uma pequena variação aleatória (± 0.05) para introduzir diversidade nos *scores*.

4. Desenvolvimento e Comparação de Diferentes Estratégias de Procura

A navegação eficiente na rede viária do município de Braga, modelada por um grafo de dimensão considerável constituído por **8624 nós** e **19753 arestas**, exige a implementação de algoritmos de procura robustos. O módulo `Graph.py` disponibiliza um conjunto de **estratégias**, categorizadas em **não-informadas** e **informadas**, que permitem ao sistema ponderar variáveis críticas como o tempo de resposta, a distância percorrida e os custos operacionais (tráfego e clima).

4.1. Formalização Matemática da Função de Custo e Heurística

A precisão dos algoritmos de procura depende intrinsecamente da definição da função de avaliação $\boldsymbol{f(n) = c(n) + h(n)}$**.** No contexto deste sistema, o custo de uma aresta não representa apenas tempo ou distância, mas um valor híbrido que integra fatores ambientais e operacionais.

Importante: O valor resultante da função de custo é um valor adimensional que serve apenas como função objetivo para comparação entre rotas. É irrelevante o seu valor absoluto, desde que preserve a ordenação correta das alternativas. O sistema apenas requer que valores menores correspondam a rotas preferíveis.

4.1.1. Função de Custo ($c(n)$)

O custo de travessia entre dois nós u e v, denotado por $c(u, v)$, é calculado dinamicamente no momento da expansão da aresta. A fórmula geral é dada por:

$$c(u,v) = \left[t_{\text{base}}(u,v) * M_{\text{clima}}(u) * M_{\text{trafego}}(u)\right] * \left[1 + \beta * (1 - \eta(u,v))\right]$$

Onde:

- $t_{\text{base}}(u,v) = \frac{d(u,v)}{\frac{v_{\max(u,v)}}{3.6}}$ representa o **tempo mínimo de viagem** (em segundos), calculado pela razão entre o comprimento da aresta e a sua velocidade máxima convertida para m/s;
- M_{clima} e M_{trafego} são **coeficientes dinâmicos** (≥ 1.0) determinados pela zona geográfica do nó de origem u;
- $\beta = 0.2$ é o **peso atribuído à componente ecológica do sistema;**
- $\eta(u,v) \in [0,1]$ é o ***score* ecológico da aresta,** onde valores mais altos indicam rotas mais amigas do ambiente.

Exemplo Numérico: Para uma aresta de 500m com velocidade máxima de 50 km/h, condições de chuva ($M_{\text{clima}} = 1.3$), tráfego normal ($M_{\text{trafego}} = 1.0$) e *score* ecológico 0.7, temos: $t_{\text{base}} = \frac{500}{\frac{50}{3.6}} = 36s$, $c = 36 \times 1.3 \times [1 + 0.2 \times (1 - 0.7)] = 49.6s$.

Nota: O termo $[1 + \beta * (1 - \eta(u, v))]$ atua como multiplicador ecológico, variando entre 1.0 (rota perfeitamente ecológica) e 1.2 (rota não-ecológica).

4.1.2. Função Heurística ($h(n)$)

Para garantir a admissibilidade e a consistência necessárias ao algoritmo A*, a heurística $h(n)$ estima o custo restante até ao nó objetivo (*goal*) baseando-se no cenário mais otimista possível: uma deslocação em linha reta à velocidade máxima permitida na rede.

$$h(u, v) = \frac{D_{\text{Haversine}(u,v)}}{\frac{v_{\text{global\_max}}}{3.6}}$$

Onde:

- $D_{\text{Haversine}(u,v)}$ calcula a **distância (em metros) do grande círculo** (em linha reta sobre a superfície terrestre) entre o nó atual e o destino, em quilómetros;
- $v_{\text{global\_max}}$ corresponde à **maior velocidade registada em qualquer aresta do grafo** (e.g., 120 km/h em autoestrada);
- 3.6 é o **fator de conversão** de km/h para ms/s.

Admissibilidade: A heurística considera apenas o componente temporal mínimo ideal, ignorando os fatores operacionais, ecológicos e ambientais (assumindo condições perfeitas). Como o custo real inclui sempre valores não-negativos para esses componentes multiplicativos, temos garantidamente $\boldsymbol{h(u, v) \leq c(u, v)}$, preservando a admissibilidade da heurística.

4.2. Estratégias de Procura Não-Informada

Esta classe de algoritmos explora o espaço de estados sem recorrer a estimativas sobre a localização do objetivo, baseando a decisão exclusivamente na estrutura topológica e nos custos acumulados das arestas.

- **Busca em Profundidade (DFS):** O algoritmo expande a rota linearmente até atingir um limite de profundidade ou um "beco sem saída". Devido à sua natureza, **raramente encontra a solução ótima** e tende a produzir **caminhos excessivamente longos** e irregulares. A sua utilidade no projeto restringe-se à validação da conectividade entre nós isolados;

- **Busca em Largura (BFS):** Explora o grafo em camadas concêntricas, o que garante a descoberta do caminho com o **menor número de arestas** (saltos). No entanto, **ignora o peso das ligações** (custo temporal ou energético), o que resulta frequentemente na seleção de **rotas mais lentas** que atravessam zonas urbanas densas, em detrimento de vias rápidas periféricas;

- **Custo Uniforme (UCS):** Prioriza a expansão do nó com o menor custo acumulado total ($g(n)$). Esta abordagem **assegura**, matematicamente, a **identificação da rota ótima** (de menor custo ponderado). Contudo, o processo de exploração radial obriga à **análise de um número elevado de nós** em todas as direções, o que penaliza o tempo de computação em trajetos de longa distância.

4.3. Estratégias de Procura Informada

Para otimizar o desempenho computacional, estas estratégias incorporam uma função heurística ($h(n)$) para orientar a procura na direção do objetivo, reduzindo drasticamente o número de nós analisados. A **heurística implementada define-se como o tempo mínimo teórico de viagem** em linha reta até ao destino, calculado com base na velocidade máxima global permitida na rede. Esta abordagem garante a admissibilidade da heurística, condição essencial para a otimalidade do algoritmo A*.

- ***Greedy Best-First*:** Seleciona o nó sucessor que aparenta estar mais próximo do destino, negligenciando o custo do trajeto já efetuado. Embora apresente **tempos de resposta extremamente baixos**, a ausência de ponderação do custo real conduz frequentemente a **soluções sub-ótimas** e a "armadilhas" locais em zonas de custo elevado (e.g., congestionamento);

- **A* (*A-Star*):** Combina o custo real ($g(n)$) com a estimativa heurística ($h(n)$) através da função de avaliação $\boldsymbol{f(n) = g(n) + h(n)}$. Ao utilizar uma heurística admissível (tempo mínimo teórico calculado com a velocidade máxima global da rede), o algoritmo A* garante a **mesma solução ótima do Custo Uniforme**, mas com uma **redução significativa no espaço de procura**, constituindo a escolha padrão para a operação dos veículos;

- ***Weighted* A*:** Uma variante implementada que aplica um fator de ponderação à heurística ($w > 1$), no nosso caso $w = 1.5$. Esta técnica **sacrifica**, ligeiramente, a **garantia estrita de otimalidade** em favor de uma **convergência mais rápida** para o destino, útil em situações de elevada carga no servidor onde o tempo de resposta é prioritário.

5. Implementação do Sistema de Simulação Dinâmica

5.1. Arquitetura e Comunicação do Sistema

O sistema *TaxiGreen* implementa uma arquitetura cliente-servidor baseada em **comunicação `TCP/IP` através de *sockets***. O cliente oferece uma **interface terminal** desenvolvida com a biblioteca *Rich*, apresentando elementos visuais como painéis, tabelas e menus interativos de forma organizada e visualmente apelativa. A comunicação estabelece-se através de uma conexão persistente na porta 8888, utilizando **serialização com *pickle*** para transmitir estruturas de dados complexas entre os componentes. Esta abordagem permite uma **separação clara entre a interface do utilizador e a lógica de simulação**, com um protocolo de comando-resposta bem estruturado.

5.2. Dinâmica de Simulação e Comportamento do Sistema

A simulação destaca-se pela geração aleatória de pedidos de viagem que ocorrem durante a execução, **simulando a chegada imprevisível de clientes** típica dos sistemas reais de transporte. Estes pedidos complementam os criados manualmente pelos utilizadores, criando um **fluxo contínuo de atividade**. A autonomia dos veículos é gerida dinamicamente, ou seja, cada **veículo consome energia proporcionalmente à distância percorrida**, e quando atinge níveis críticos, o sistema, automaticamente, **inicia procedimentos de recarga/ reabastecimento**. Condições ambientais variáveis, como meteorologia e tráfego, afetam os tempos de viagem através de multiplicadores aplicados às rotas, permitindo simular diferentes cenários urbanos.

5.3. Funcionalidades Principais da Interface

A interface organiza-se em oito funcionalidades principais:

1. **"Nova Viagem Manual":** Permite criar pedidos específicos indicando origem, destino, número de passageiros e preferência pelo modo ecológico, com validação automática dos nodos;
2. **"Estado da Frota":** Mostra informações detalhadas sobre todos os veículos, incluindo localização, autonomia, tipo de motor e estado operacional, com codificação visual por cores;
3. **"Estado dos Pedidos":** Apresenta todas as viagens em curso, pendentes ou concluídas, com detalhes sobre atribuição de veículos, distâncias e tempos estimados;

4. **"Mapa de Rotas":** Gera mapas interativos do município de Braga em `HTML` com representação de todos os nós, arestas, viaturas, postos de abastecimento e postos de carregamento representados de forma simples e apelativa;
5. **"Alterar Algoritmo":** Oferece seis opções de *pathfinding* (`DFS`, `BFS`, `UCS` [*Uniform Cost Search*], `Greedy`, `A*` e `Weighted A*`) com diferentes compromissos entre velocidade e otimalidade, alteráveis em tempo real;
6. **"Condições Ambientais":** Configura parâmetros como meteorologia, tráfego e zonas de aplicação para simular diferentes cenários operacionais;
7. **"Gerar Relatórios":** Exporta dados do sistema para ficheiros `CSV` estruturados, facilitando análise posterior e processamento estatístico;
8. **"*Dashboard* do Sistema":** Consolida métricas-chave de desempenho numa vista agregada, mostrando o estado geral da operação.

5.4. Gestão de Concorrência e Estado do Sistema

O servidor opera com uma **arquitetura *multithreaded*** que permite a simulação principal e o atendimento de múltiplos clientes a funcionarem em paralelo. A **simulação corre numa *thread*** dedicada que processa eventos a cada 3 segundos, gerando pedidos aleatórios, processando os pendentes, atribuindo veículos e gerindo a autonomia da frota. Isto garante que a interface responde rapidamente, independentemente da carga de processamento.

Um ***buffer* de pedidos**, baseado em `queue.Queue`, sincroniza as *threads*, **recebendo pedidos manuais e automáticos** que são depois processados na ordem de chegada. Mecanismos de ***locking* protegem o acesso a dados partilhados** durante operações críticas como atribuição de veículos ou atualização de autonomias.

Cada cliente conectado tem uma *thread* dedicado para comunicação, permitindo múltiplos utilizadores simultâneos sem bloquear a simulação principal. O sistema também mantém um histórico automático de rotas e cancela pedidos que excedem tentativas de atribuição, prevenindo acumulação. Esta arquitetura modular resulta num sistema robusto que **suporta operação contínua com vários utilizadores**, mantendo consistência no estado da simulação e oferecendo uma experiência interactiva fluida.

6. Avaliação da Eficiência dos Algoritmos

Esta secção apresenta a **análise quantitativa das estratégias de procura,** transitando de uma avaliação baseada na simulação global para uma metodologia de ***benchmarking* isolado.** Esta abordagem visa extrair métricas de baixo nível (tempo de CPU e memória), eliminando a variabilidade introduzida pela latência de rede ou pela renderização gráfica da interface.

6.1. Metodologia de Teste e Métricas

A avaliação foi realizada recorrendo ao *script* dedicado `benchmark_algoritmos.py`, desenvolvido para testar a classe `Graph` de forma isolada. O procedimento consistiu na execução de 100 iterações de rotas aleatórias, mantendo os pares origem/destino constantes para todos os algoritmos em cada iteração, assegurando a comparabilidade direta dos resultados.

As métricas foram recolhidas utilizando as bibliotecas `time` (relógio monotónico de alta precisão) e `tracemalloc` do `Python`:

- **Tempo de Execução (ms):** Tempo médio de processamento para calcular uma rota;
- **Memória de Pico (KB):** Máximo de memória RAM alocada durante a execução;
- **Custo do Caminho:** Custo acumulado da solução retornada (ponderado por tempo, tráfego e ecologia);
- **Passos (*Hops*):** Número de nós que compõem o caminho final;
- **Eficiência (vs Ótimo):** Desvio percentual do custo encontrado face ao melhor custo possível.

6.2. Análise Comparativa e Otimizações

A análise dos resultados evidencia o impacto crítico das otimizações de engenharia de *software* na materialização da eficiência teórica dos algoritmos.

6.2.1. Desempenho do A* (*A-Star*) vs. Custo Uniforme (UCS)

Teoricamente, o algoritmo **A*** é mais eficiente que o **Custo Uniforme** por expandir um menor número de nós, guiado pela heurística. No entanto, em implementações simples, o custo computacional de calcular a heurística $h(n)$ (neste caso, a fórmula de *Haversine* com múltiplas operações trigonométricas) pode ser superior ao ganho obtido pela redução dos nós, tornando o **A* mais lento** em tempo de relógio.

Neste projeto, este **obstáculo foi superado** através da implementação de uma **Cache de Heurística** ($O(1)$) e de uma versão nativa do algoritmo.

- **Impacto da Otimização:** Ao eliminar o recálculo repetitivo de distâncias geográficas, o custo de processamento por nó do **A*** aproximou-se do custo do **UCS**, sendo normalmente o mesmo;
- **Resultado Final:** Com a penalização computacional da heurística mitigada, a **vantagem algorítmica do A*** (menor espaço de procura) traduziu-se num **ganho efetivo** de desempenho. O **A*** (59.54 ms) superou o **Custo Uniforme** (63.30 ms) em velocidade, mantendo a garantia matemática de otimalidade.

6.2.2. *Weighted* A* vs. *Greedy*

- ***Greedy Best-First:*** Apresentou-se como o **algoritmo mais rápido (3.05 ms)**, dado que ignora o custo acumulado e avança sofregamente para o objetivo. Contudo, a qualidade da solução é **severamente penalizada,** com um custo médio 27.5% superior ao ótimo, tornando-o inviável para rotas que exijam eficiência energética ou temporal;
- ***Weighted* A* ($\varepsilon = 1.5$ e $\varepsilon = 2.0$):** As variantes ponderadas revelaram-se a grande surpresa do *benchmark*, dominando o compromisso entre rapidez e qualidade:
  - **Com $\varepsilon = 1.5$,** o tempo reduziu para metade (29.10 ms) face ao A* padrão;
  - **Com $\varepsilon = 2.0$,** o algoritmo atingiu um tempo de **19.63 ms** (3x mais rápido que o A*), mantendo, surpreendentemente, o **Custo Ótimo** (559.2) em todos os testes.

Análise: Embora teoricamente o ***Weighted* A* sacrifique a otimalidade por velocidade,** a topologia de Braga permitiu que a versão $\varepsilon = 2.0$ convergisse rapidamente para a solução ideal sem desvios, tornando-a a estratégia mais eficiente do sistema para operação em tempo real.

6.2.3. Estratégias de Base (BFS e DFS)

- **BFS (*Breadth-First Search*):** Garante o **caminho com menos arestas,** mas não o mais rápido. O custo temporal das rotas foi **31.8% superior** ao ótimo, demonstrando que o caminho mais "curto" em nós frequentemente atravessa zonas urbanas lentas ao invés de vias rápidas;
- **DFS (*Depth-First Search*):** Confirmou-se **inadequado para a navegação em grafos complexos,** com custos **2056% superiores** ao ótimo e tempos de execução elevados, servindo apenas para validação de conectividade.

6.3. Resultados Obtidos

Os dados recolhidos durante a fase de *benchmark* foram consolidados na Tabela 2 e podem ser visualizados na Figura 2. Estes elementos permitem uma análise multidimensional que contrapõe a precisão matemática dos custos com a realidade da execução computacional.

Algoritmo	Sucesso	Tempo (ms)	Memória (KB)	Custo	Passos	Eficiência
A*	100%	**59.54**	504.5	**559.2**	65.9	**Ótimo**
Uniform Cost	100%	63.30	670.4	559.2	65.9	Ótimo
Weighted A* ($\varepsilon = 1.5$)	100%	29.10	738.3	559.2	65.9	Ótimo
***Weighted* A* ($\varepsilon = 2.0$)**	100%	**19.63**	752.7	559.2	65.9	**Ótimo**
Greedy	100%	**3.05**	**91.8**	712.7	71.6	+27.5%
BFS	100%	48.62	418.2	737.1	**44.1**	+31.8%
DFS	100%	115.41	23,776.8	12,056.7	1314.0	+2056%

Tabela 2: Comparação de Desempenho Algorítmico (Média de 100 iterações).

Para facilitar a interpretação do compromisso (*trade-off*) entre **velocidade** e **otimalidade,** o gráfico de dispersão abaixo posiciona cada algoritmo num plano cartesiano, onde a proximidade à origem (canto inferior esquerdo) representa o desempenho ideal.

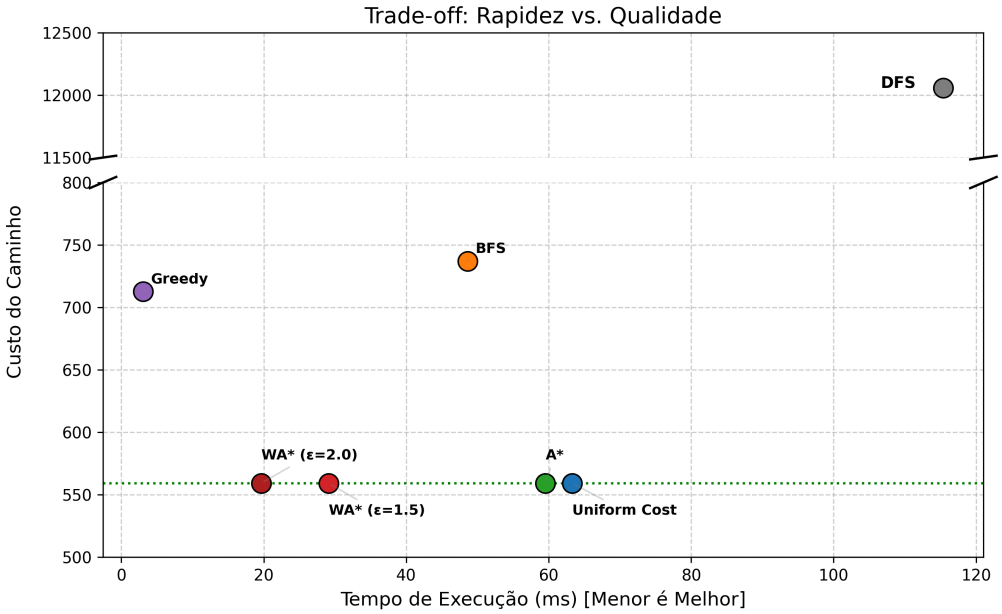

Figura 2: ***Trade-off:*** O ***Weighted* A*** ($\varepsilon = 2.0$) atinge o desempenho ideal, combinando a solução ótima com um tempo de execução 3 vezes inferior ao **A* padrão.**

6.3.1. Interpretação dos Resultados

A disposição dos algoritmos na Figura 2 permite identificar claramente três grupos de desempenho distintos:

- **A Fronteira de Eficiência (*Weighted* A* e A*):** O destaque vai para o ***Weighted* A*** com $\varepsilon = 2.0$ (ponto vermelho escuro), que se revelou a estratégia superior. Alcançou o **custo ótimo** (559.2) num tempo recorde de **19.63 ms**, superando largamente o **A* padrão (59.54 ms)**. Isto prova que, nesta topologia, uma heurística mais agressiva consegue acelerar a convergência sem sacrificar a qualidade da solução;
- **A Garantia Clássica (*Uniform Cost*):** O Custo Uniforme (ponto azul) situa-se **próximo do A*,** mas com um **ligeiro atraso temporal (63.30 ms).** Embora robusto, a sua natureza não-informada obriga-o a explorar uma franja maior de nós para garantir a mesma otimalidade que o A* atinge mais rapidamente;
- **Velocidade vs. Qualidade (*Greedy* e BFS):**
  - **O *Greedy*** (ponto roxo) é o **extremo da rapidez ($\sim$ 3 ms),** mas paga um preço alto pela sua rapidez de julgamento: um **agravamento de $+27.5\%$** no custo da viagem;
  - **O BFS** (ponto laranja), embora **razoavelmente rápido (48 ms),** apresenta resultados **ainda piores ($+31.8\%$ de custo)** por ignorar totalmente os pesos das arestas (tráfego e ecologia), preferindo rotas com menos cruzamentos mas potencialmente mais lentas.
- **Ineficiência Extrema (DFS):** A quebra de escala no eixo vertical é necessária exclusivamente para acomodar **o DFS** (ponto cinzento). Com custos superiores a **12.000 (vs 559.2 do ótimo),** este algoritmo comporta-se como um *outlier* estatístico, percorrendo caminhos erráticos e validando a necessidade de métricas de custo orientadas.

7. Simulação de Condições Dinâmicas

A robustez de um sistema de gestão de frotas não é medida apenas pela sua eficiência em condições ideais, mas pela sua capacidade de adaptação a imprevistos e variações ambientais. O módulo `ambient.py` atua como o **"motor de caos" da simulação**, **introduzindo perturbações aleatórias** que obrigam os agentes a recalcular rotas e a tomar decisões táticas em tempo real. A implementação destas dinâmicas baseia-se em três pilares fundamentais: a **variação de custos de travessia** (trânsito e clima), a **falibilidade da infraestrutura** e a **imprevisibilidade da procura**.

7.1. Modelação de Trânsito e Fenómenos Atmosféricos

Ao contrário de abordagens estáticas onde o custo de uma aresta é constante, o modelo implementa um sistema de **multiplicadores zonais dinâmicos**. Em vez de alterar o custo de cada aresta individualmente, o que seria computacionalmente proibitivo num grafo com milhares de arestas, o sistema utiliza a lógica de zonas definida em `ZoneCalc.py`.

A função de custo $g(n)$ em `Graph.py` foi desenhada para consultar estes multiplicadores em tempo de execução:

$$\text{Custo}_{\text{total}} = \text{Custo}_{\text{base}} \times M_{\text{clima}} \times M_{\text{trânsito}} \times \text{Fator}_{\text{ecológico}}$$

Onde:

- M_{clima} é o **multiplicador dos acontecimentos climáticos**;
- $M_{\text{trânsito}}$ é o **multiplicador dos acontecimentos do tráfego**;
- M_{clima} e $M_{\text{trânsito}}$ são **geridos dinamicamente** por zona (e.g., `center_north`, `periphery_west`);
- As funções `update_traffic` e `update_weather` permitem **alterar estes escalares globalmente para uma zona**, simulando, por exemplo, um congestionamento súbito no centro da cidade ou chuvas fortes na periferia que obrigam a uma redução da velocidade média.

Esta abordagem permite simular cenários onde o caminho geograficamente mais curto (Heurística de *Haversine*) deixa de ser o mais rápido devido a um engarrafamento severo ($M_{\text{trânsito}} > 2.0$), **forçando o algoritmo A^* a explorar rotas alternativas** menos congestionadas.

7.2. Falhas na Infraestrutura de Carregamento

A disponibilidade das estações de carregamento e postos de combustível não é garantida. O **sistema simula falhas técnicas ou ocupação total** através da função `update_station_availability`.

Quando uma estação fica `inutilizável` (sinalizada como `INDISPONÍVEL`):

1. O objeto `FuelStation` correspondente **é marcado como indisponível.**
2. Os algoritmos de procura, **ao avaliarem a ação `Refuel`,** verificam esta *flag*.
3. Se um veículo tinha planeado uma rota para essa estação, o sistema obriga a um re-planeamento para a estação funcional mais próxima, **testando a resiliência da autonomia restante** do veículo.

7.3. Geração Aleatória de Pedidos

Para avaliar a capacidade de resposta da frota, a função `generate_random_request` introduz novos pedidos no sistema de forma assíncrona. Estes pedidos variam em:

- **Localização:** Origem e destino aleatórios dentro dos nós válidos do grafo;
- **Prioridade:** Diferentes níveis de urgência que afetam a função de utilidade da alocação;
- **Preferências:** Inclusão de requisitos de sustentabilidade (preferência por veículos elétricos) e janelas temporais de espera (`max_wait_time`).

Esta injeção contínua de objetivos impede que o sistema atinja um estado estático, exigindo que o algoritmo de escalonamento **resolva continuamente um problema de *Online Bipartite Matching*** entre a frota disponível e a lista de pedidos pendentes.

8. Sumário e Discussão dos Resultados Obtidos

O presente trabalho permitiu desenvolver e avaliar um sistema integrado de simulação dinâmica para gestão inteligente de uma frota de táxis urbanos, com ênfase na sustentabilidade ambiental e na eficiência operacional. A implementação foi estruturada em torno de três pilares fundamentais: a **modelação realista da rede** viária do município de Braga, o **desenvolvimento e comparação de estratégias de procura** aplicadas à navegação veícular, e a **criação de uma plataforma de simulação** interativa e dinâmica.

A representação computacional da cidade sob a forma de um grafo demonstrou ser um **modelo adequado para simular a complexidade de uma rede de transportes** urbana. A classificação zonal híbrida, com granularidade diferenciada entre a **zona central (8 setores) e a periferia (4 setores)**, revelou-se uma abordagem eficaz para aplicar fatores ambientais de forma realista e escalável. A inclusão de um ***score* ecológico** dinâmico, para cada segmento viário, **permitiu integrar critérios de sustentabilidade** diretamente no cálculo das rotas, alinhando o sistema com os objetivos ambientais contemporâneos.

A avaliação comparativa dos algoritmos de procura evidenciou o equilíbrio fundamental entre otimalidade da solução e eficiência computacional. O **algoritmo `A*` destacou-se como a escolha mais equilibrada**, combinando a garantia de otimalidade do Custo Uniforme com uma convergência significativamente mais rápida, graças à utilização de uma **heurística** admissível **baseada na distância geodésica**. Este desempenho foi particularmente relevante no contexto dinâmico da simulação, onde a necessidade de processar múltiplos pedidos simultaneamente exigia algoritmos com tempos de resposta reduzidos. O **`Weighted A*` demonstrou ser uma alternativa viável para cenários com restrições computacionais mais exigentes**, oferecendo uma convergência ainda mais rápida com um compromisso aceitável na otimalidade.

A **arquitetura cliente-servidor baseada em `TCP/IP`** e a **interface gráfica via terminal** desenvolvida com a biblioteca *Rich* proporcionaram uma experiência de utilização intuitiva e profissional, **permitindo aos operadores interagir com o sistema eficazmente**. A capacidade de gerar mapas interativos em `HTML` e relatórios estruturados em `CSV` ofereceu ferramentas valiosas para análise e tomada de decisão. A **implementação *multithreaded*** no servidor **garantiu que a simulação principal e o atendimento a múltiplos clientes pudessem funcionar em paralelo**, resultando num sistema responsivo e robusto.

Os **mecanismos de simulação dinâmica**, incluindo a **geração aleatória de pedidos**, a **gestão proativa da autonomia** dos veículos e a **modelação de condições ambientais** variáveis, conferiram ao sistema um elevado grau de realismo, permitindo simular cenários operacionais complexos e avaliar a resiliência do sistema sob diferentes condições. A **capacidade de alterar dinamicamente os parâmetros da simulação**, incluindo o algoritmo de *pathfinding* e as condições ambientais, **transformou a plataforma numa ferramenta versátil** para análise de diferentes estratégias operacionais.

Em conclusão, o sistema ***TaxiGreen*** demonstrou ser uma **plataforma funcional e realista para a simulação de sistemas de transporte urbano**, integrando conceitos avançados de `Inteligência Artificial`, computação distribuída e sustentabilidade ambiental. O trabalho realizado oferece assim uma base sólida para investigação futura em sistemas inteligentes de mobilidade urbana.

9. Anexos

9.1. Anexo A - Diagrama de Classes

O **sistema *TaxiGreen*** segue uma arquitetura orientada a objetos modular, onde a lógica de simulação é centralizada, mas as entidades mantêm responsabilidades específicas. A figura abaixo ilustra as principais classes e os seus relacionamentos.

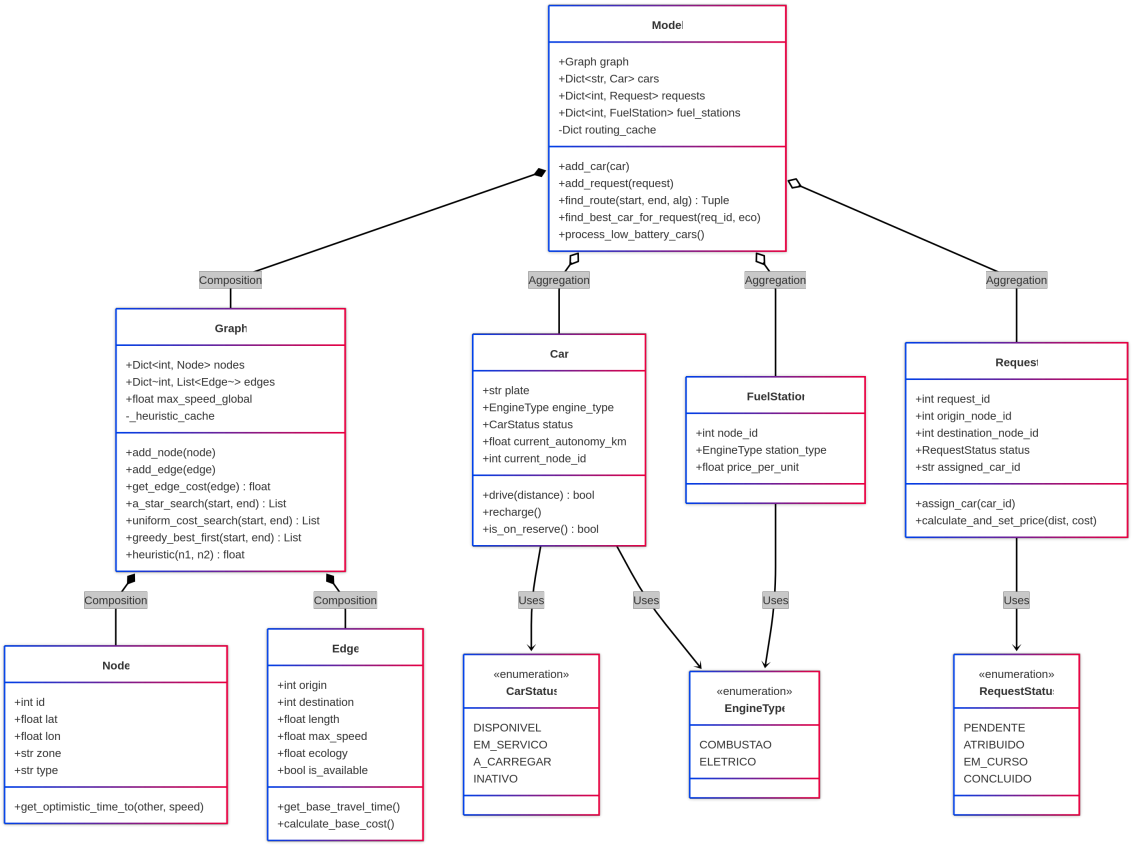

Figura 3: Diagrama de Classes UML do sistema TaxiGreen, evidenciando a relação entre o Modelo central, o Grafo topológico e as entidades móveis (Carros e Pedidos).

Descrição dos Componentes Principais:

- **`Model`:** A classe central que gere o estado da simulação (carros, pedidos, estações) e mantém a `routing_cache` para otimizar o desempenho das procuras;
- **`Graph`:** Encapsula a rede viária (Nós e Arestas) e implementa a lógica de *pathfinding* (e.g., A*, UCS) e o cálculo de custos dinâmicos;
- **`Car`:** Representa os agentes móveis, gerindo a autonomia, o estado operacional (`CarStatus`) e a validação de capacidade para os pedidos;

- **`Request`:** Modela o ciclo de vida do serviço, desde a criação (`PENDENTE`) até à conclusão, incluindo o cálculo automático do preço da viagem;
- **`FuelStation`:** Define os pontos de reabastecimento no grafo, tipificados por motor (Elétrico/Combustão), essenciais para a gestão de energia da frota.

9.2. Anexo B - *Script* de *Benchmarking*

O *script* `benchmark_algoritmos.py` foi desenvolvido para isolar e testar o desempenho da classe `Graph` e dos algoritmos de procura. O teste executa 100 iterações com pares origem-destino aleatórios (mas consistentes entre algoritmos na mesma iteração) e utiliza as bibliotecas `time` (relógio monotónico) e `tracemalloc` para medição precisa de recursos.

```python
import sys
import os
import time
import random
import tracemalloc
import statistics
from rich.console import Console
from rich.table import Table
from rich.progress import track

# Configuração de caminhos para importação dos módulos do projeto
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
if BASE_DIR not in sys.path:
    sys.path.append(BASE_DIR)

try:
    from src.models.Graph import Graph
    from src.models.Model import Model
    from src.simulation.data_loader import load_nodes, load_edges
except ImportError as e:
    print(f"Erro de importação: {e}")
    sys.exit(1)

console = Console()

def run_benchmark(iterations: int = 100):
    """
    Executa testes de performance comparativa nos algoritmos de procura.
    Métricas: Tempo (ms), Memória (KB), Custo, Passos, Taxa de Sucesso.
    """
    # 1. Inicialização do Grafo
    graph = Graph()
```

```
    nodes_path = os.path.join(BASE_DIR, 'data', 'nodes.csv')
    edges_path = os.path.join(BASE_DIR, 'data', 'edges.csv')

    # Carregar dados (nodes e edges)
    load_nodes(graph, nodes_path)
    load_edges(graph, edges_path)

    # Inicializar Modelo
    model = Model(graph)
    node_ids = list(graph.nodes.keys())

    # 2. Definição dos Algoritmos a testar
    algorithms = [
        "bfs", "dfs", "uniform_cost",
        "greedy", "weighted_a_star", "a_star"
    ]

    # Estrutura para guardar resultados
    results = {alg: {'time': [], 'memory': [], 'cost': [],
                     'path_len': [], 'success': 0} for alg in
                     algorithms}

    # 3. Geração de Cenários de Teste (Pares Origem -> Destino)
    test_cases = []
    for _ in range(iterations):
        start = random.choice(node_ids)
        end = random.choice(node_ids)
        while start == end:
            end = random.choice(node_ids)
        test_cases.append((start, end))

    # 4. Execução dos Testes
    for alg in algorithms:
        for start_node, end_node in track(test_cases,
        description=f"Testando {alg.upper()}..."):

            # Limpar cache para forçar cálculo real
            if hasattr(model, '_clear_cache'):
                model._clear_cache()

            # Iniciar medição de recursos
            tracemalloc.start()
            start_time = time.perf_counter() # Relógio de alta precisão

            # Executar Procura
            path, cost, dist = model.find_route(start_node, end_node,
            algorithm=alg)
```

```
77
78              end_time = time.perf_counter()
79              _, peak_mem = tracemalloc.get_traced_memory()
80              tracemalloc.stop()
81
82              # Processamento de Métricas
83              execution_time_ms = (end_time - start_time) * 1000
84              peak_memory_kb = peak_mem / 1024
85
86              if path:
87                  results[alg]['success'] += 1
88                  results[alg]['time'].append(execution_time_ms)
89                  results[alg]['memory'].append(peak_memory_kb)
90                  results[alg]['cost'].append(cost)
91                  results[alg]['path_len'].append(len(path))
92
93      # 5. Apresentação (Output em Tabela Rich omitido para brevidade)
94      # ... (código de geração da tabela de resultados omitido) ...
95
96  if __name__ == "__main__":
97      run_benchmark(iterations=100)
```

9.3. Anexo C - Estudo da Heurística e Otimização

A análise inicial de desempenho revelou um fenómeno inesperado. Embora o algoritmo A* expandisse menos nós que a Procura de Custo Uniforme (UCS), o seu tempo de execução era superior ($\sim$ 84ms vs $\sim$ 60ms). Este anexo detalha o diagnóstico e a solução de otimização implementada.

9.3.1. O Problema: Custo Computacional da Heurística

A **função heurística $h(n)$** utilizada baseia-se na **distância de *Haversine*** (distância geodésica numa esfera), cuja fórmula envolve múltiplas operações trigonométricas pesadas (senos, cossenos, raízes quadradas e arco-tangentes):

$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 \cdot \operatorname{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

$$d = R \cdot c$$

Numa procura típica, esta função é chamada milhares de vezes. O tempo de CPU gasto nestes cálculos trigonométricos **anulava o ganho obtido pela redução do espaço de procura** (menos nós visitados).

9.3.2. A Solução: *Cache* de Heurística ($O(1)$)

Dado que a localização geográfica dos nós é estática durante a simulação, a distância heurística entre qualquer par de nós (u, v) é constante. Implementou-se uma estratégia de ***memorization*** na classe `Graph`:

```python
# Otimização implementada em src/models/Graph.py
def heuristic(self, node1_id, node2_id):
    # Chave única para o par de nós
    cache_key = (node1_id, node2_id)

    # Verificação O(1) antes de calcular
    if cache_key in self._heuristic_cache:
        return self._heuristic_cache[cache_key]

    # ... cálculo pesado (Haversine) ...

    self._heuristic_cache[cache_key] = result
    return result
```

9.3.3. Impacto nos Resultados

A introdução da *cache* eliminou a redundância de cálculos, reduzindo a complexidade prática da heurística para um simples acesso a um dicionário (*Hash Map*).

Métrica	Antes da Otimização	Depois da Otimização	Melhoria
Tempo Médio (A*)	84.00 ms	59.54 ms	**-29.1%**
Comparação vs UCS	Mais Lento (+39%)	**Mais Rápido (-6%)**	*Inversão de Liderança*

Tabela 3: Impacto quantitativo da otimização da heurística no desempenho do algoritmo A*.

Conclusão: Com a otimização, o sistema capitalizou a vantagem teórica do A* (menor número de nós expandidos) sem sofrer a penalização computacional da trigonometria, tornando-o o algoritmo ótimo mais rápido do sistema.

10. Bibliografia

[1] S. Russell e P. Norvig, *Artificial Intelligence: A Modern Approach*, 4.º ed. Pearson, 2020.

[2] G. Boeing, «OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks», *Computers, Environment and Urban Systems*, vol. 65, pp. 126–139, 2017.

[3] Python Software Foundation, «tracemalloc — Trace memory allocation», Python 3.13 Documentation. [Em linha]. Disponível em: https://docs.python.org/3/library/tracemalloc.html

[4] OpenStreetMap Contributors, «Planet dump». [Em linha]. Disponível em: https://planet.osm.org/

[5] W. McGugan, «Rich: A Python library for rich text and beautiful formatting in the terminal». [Em linha]. Disponível em: https://github.com/Textualize/rich