
Análise de Desempenho de um Sistema de Roteamento Geoespacial com Integração ao OpenStreetMap

Guilherme Coutinho Sousa Ferreira



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Uberlândia
2026

Guilherme Coutinho Sousa Ferreira

**Análise de Desempenho de um Sistema de
Roteamento Geoespacial com Integração ao
OpenStreetMap**

Trabalho de Conclusão de Curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito exigido parcial à obtenção do Título
de Bacharel em Ciência da Computação

Área de concentração: Ciência da Computação

Orientador: Prof. Humberto Luiz Razente

Uberlândia

2026

Guilherme Coutinho Sousa Ferreira

**Análise de Desempenho de um Sistema de
Roteamento Geoespacial com Integração ao
OpenStreetMap**

Trabalho de Conclusão de Curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito exigido parcial à obtenção do Título
de Bacharel em Ciência da Computação

Trabalho aprovado. Uberlândia, 12 de agosto de 2026.

Prof. Humberto Luiz Razente
Orientador

Prof.^a Maria Adriana Vidigal de Lima

Prof. Rafael Dias Araújo

Uberlândia
2026

Resumo

O presente estudo investiga o desempenho de um sistema de roteamento baseado em Sistemas de Informação Geográfica (SIG), que identifica a menor rota possível para o deslocamento entre dois pontos ao longo de uma rede de ruas ou estradas. Esse sistema de roteamento utiliza dados cartográficos de código aberto do OpenStreetMap para cobrir todo o território do Brasil. Ele foi implementado utilizando PostgreSQL, juntamente com suas extensões associadas PostGIS e pgRouting, para gerenciar e analisar a rede viária. O backend do sistema de roteamento foi implementado utilizando FastAPI, e seu frontend foi implementado utilizando React e Leaflet. O sistema completo foi containerizado utilizando Docker.

O principal objetivo deste estudo foi avaliar três algoritmos conhecidos de menor caminho: Dijkstra, AStar e AStar Bidirecional. Aproximadamente 673 mil pares de arestas e 671 mil vértices estavam contidos no conjunto de dados da rede viária brasileira, que foi importado para o sistema de roteamento utilizando o osm2pgsql.

Os testes foram conduzidos em seis rotas diferentes, com comprimentos variados. Os resultados indicam que o Dijkstra apresentou os menores tempos de execução para cada rota testada e também forneceu, de forma consistente, vantagens de tempo quase constantes em relação tanto ao AStar quanto ao AStar Bidirecional. Esse resultado é contrário a estudos anteriores, que indicaram que o AStar geralmente supera o Dijkstra na determinação da menor rota.

Palavras-chave: Roteamento geoespacial, pgRouting, OpenStreetMap, Docker, FastAPI.

Abstract

The present study investigates the performance of a geographic information systems (GIS) based routing system which identifies the shortest route possible for travel between two points along a street or road network. This routing system utilizes open-source mapping data from OpenStreetMap to cover the entirety of Brazil. It was implemented using PostgreSQL with its associated extensions postgis and pgRouting to manage and analyze the road network. The routing system's backend was implemented using fastapi and its frontend was implemented using react and leaflet. The complete system has been containerized within docker.

The primary objective of this study was to evaluate three well known shortest-route algorithms: dijkstra, astar and bidirectional. Approximately 673 thousand edge pairs and 671 thousand vertexes were contained in the brazilian road network dataset, which was imported into the routing system utilizing osm2pgsql.

Tests were conducted on six different routes of varying lengths. Results indicate that dijkstra produced the fastest execution times for each route tested; also consistently provided near-constant time advantages relative to both astar and bidirectional astar. This result is contrary to previous studies, which have indicated that astar generally outperforms dijkstra when determining the shortest route.

Keywords: geospatial routing, osm2pgsql, pgRouting, OpenStreetMap, Docker, FastAPI, Leaflet, shortest path.

Lista de ilustrações

Figura 2.1 – Modelos vetorial e matricial	15
Figura 2.2 – Correspondência entre a malha viária e o grafo de roteamento	16
Figura 4.1 – Arquitetura em três camadas do sistema e fluxo de uma requisição de rota.	34
Figura 4.2 – Formulário de entrada para o cálculo de rota.	45
Figura 4.3 – Interface do sistema exibindo o painel de resultados no modo de com- paração	47
Figura 5.1 – Rotas calculadas pelos três algoritmos, sobrepostas, para um mesmo par origem–destino em Monte Alegre de Minas	57
Figura 5.2 – Mesmo percurso apresentado pelo Google Maps, como referência qua- litativa externa	57
Figura 5.3 – Tempo médio de resposta dos três algoritmos em função da distância da rota (escala logarítmica)	59

Lista de tabelas

Tabela 2.1 – Exemplo de representação de uma geometria <code>LineString</code> em GeoJSON	24
Tabela 5.1 – Tempos médios de resposta dos algoritmos por rota	58
Tabela 5.2 – Comparação entre o sistema de Oliveira (2021) e a presente proposta .	60

Lista de abreviaturas e siglas

API	Application Programming Interface
GIS	Geographic Information System
GeoJSON	Geographic JavaScript Object Notation
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
ODbL	Open Database License
OSM	OpenStreetMap
PBF	Protocolbuffer Binary Format
REST	Representational State Transfer
SGBD	Sistema Gerenciador de Banco de Dados
SGBDG	Sistema de Gerenciamento de Banco de Dados Geográfico
SGBDR	Sistema de Gerenciamento de Banco de Dados Relacional
SIG	Sistema de Informação Geográfica
SQL	Structured Query Language
SRID	Spatial Reference System Identifier
WGS	World Geodetic System

Sumário

1	INTRODUÇÃO	11
1.1	Objetivos	12
1.2	Metodologia	13
1.3	Organização do Trabalho	13
2	CONCEITOS FUNDAMENTAIS	14
2.1	Sistemas de Informação Geográfica	14
2.2	Bancos de Dados Geográficos	15
2.2.1	Estrutura dos Dados Espaciais	16
2.2.2	PostgreSQL e PostGIS	16
2.2.3	Tipos Espaciais e SRID	17
2.3	OpenStreetMap	17
2.4	Importação de Dados OSM para o PostgreSQL	19
2.4.1	osm2pgsql	19
2.4.2	Construção da Topologia	20
2.5	pgRouting	20
2.6	Algoritmos de Caminho Mínimo	21
2.6.1	Algoritmo de Dijkstra	21
2.6.2	Algoritmo A*	22
2.6.3	A* Bidirecional	22
2.6.4	Comparação	22
2.7	Tecnologias de Desenvolvimento	23
2.7.1	React	23
2.7.2	Leaflet	23
2.7.3	GeoJSON	24
2.7.4	FastAPI	25
2.7.5	Nominatim	25
2.7.6	Docker e Docker Compose	25

2.8	Escalabilidade	26
3	TRABALHOS CORRELATOS	28
3.1	Sistemas de Roteamento sobre OpenStreetMap e pgRouting . .	28
3.2	Comparação de Algoritmos de Caminho Mínimo em Redes Vi- árias	29
3.3	Roteamento em Redes Viárias de Grande Escala	30
3.4	Posicionamento em Relação aos Trabalhos Correlatos	31
4	DESENVOLVIMENTO	33
4.1	Visão Geral da Arquitetura	33
4.2	Tecnologias Utilizadas	34
4.3	Base de Dados	34
4.3.1	Obtenção dos Dados do Brasil	34
4.3.2	Importação com osm2pgsql	35
4.3.3	Estrutura das Tabelas	36
4.3.4	Extensões PostGIS e pgRouting	37
4.4	Back-end com FastAPI	39
4.4.1	Estrutura do Projeto	39
4.4.2	Endpoints Implementados	40
4.4.3	Consultas pgRouting	40
4.4.4	Geocodificação e Busca do Nó Mais Próximo	43
4.5	Front-end com React e Leaflet	44
4.5.1	Estrutura de Componentes	44
4.5.2	Formulário de Entrada	45
4.5.3	Renderização da Rota com Leaflet	45
4.5.4	Exibição dos Resultados	46
4.6	Containerização com Docker	48
4.6.1	Estrutura do Docker Compose	48
4.6.2	Contêiner do Banco de Dados	50
4.6.3	Contêiner do Back-end	51
4.6.4	Contêiner do Front-end	51
4.6.5	Execução da Aplicação	52
4.7	Malha Urbana Detalhada sob Demanda	53
5	TESTES E RESULTADOS	55
5.1	Metodologia dos Testes	55
5.2	Teste de Menor Caminho — Comparação Visual	56
5.3	Teste de Desempenho — Comparação dos Algoritmos	58
5.4	Teste de Escalabilidade	59

5.5	Comparação com o Sistema de (Oliveira, 2021)	60
5.6	Discussão dos Resultados	61
6	CONCLUSÃO	62
6.1	Considerações Finais	62
6.2	Limitações	63
6.3	Trabalhos Futuros	63
	Referências	64

APÊNDICES 69

APÊNDICE A	–	COMANDOS DE INSTALAÇÃO E CONFIGURAÇÃO	70
APÊNDICE B	–	COMANDOS DA TENTATIVA DE IMPORTAÇÃO COM OSM2PGROUTING	72

Introdução

Após a aprovação da liberação do sinal de GPS (*Global Positioning System*) para a população civil nos anos 2000 ([The Aerospace Corporation, 2021](#)), qualquer pessoa que tenha acesso a um sistema moderno consegue, facilmente, definir um trajeto a ser percorrido mesmo que não haja conhecimento prévio do percurso. No entanto, muitos usuários não percebem a quantidade de operações que são feitas para que essa informação chegue de maneira mais simples e com possibilidades de diferentes tipos de locomoção, vias, caminhos mais rápidos ou curtos e diferentes paradas até o destino final.

Por definição, um SIG (*Sistema de Informação Geográfica*) é “um sistema automatizado para a captura, armazenamento, recuperação, análise e exibição de dados espaciais” ([Clarke, 1995](#)). Esse sistema difere dos convencionais por justamente solucionar problemas que dependem de dados geográficos, para as tomadas de decisões em atividades científicas, comerciais e governamentais ([Graham, 2012](#)). A base desse trabalho está em utilizar sistemas SIG de código aberto para processar o roteamento entre localizações indicadas.

Com uma crescente popularidade dos dados geoespaciais, houve uma maior disponibilidade de serviços e APIs que facilitam integrar as informações geográficas com outros sistemas. No entanto, esses serviços podem ter custos elevados ou restrições de uso ([Wei et al., 2018](#)). Como alternativa, o OpenStreetMap (OSM) surgiu como um projeto colaborativo de um mapa global, onde qualquer pessoa tem a oportunidade de utilizar gratuitamente os dados, além de poder contribuir alterando ou adicionando novas informações ([Antoniou, 2017](#)), diminuindo custos operacionais de projetos que utilizam bases geográficas.

Este trabalho tem como objetivo mostrar a possibilidade de projetar um sistema de roteamento geoespacial com um volume de dados de um país com vasta extensão territorial, como o Brasil ([Zhu et al., 2025](#)), utilizando como fonte primária dessas informações o OpenStreetMap.

São apresentadas neste trabalho algumas características, entre elas:

- escalabilidade, com avaliação do desempenho com um grande volume de dados

([Bondi, 2000](#));

- reprodutibilidade, onde há a possibilidade de executar um sistema completo da mesma forma em qualquer máquina ([Chamberlain; Schommer, 2014](#)); e
- interatividade, onde o usuário pode ter uma boa experiência de usabilidade ([Kivijarvi; Parnanen, 2021](#)) de um sistema web, sem um conhecimento de dados geoespaciais.

Foram utilizados os algoritmos de caminhos mais curtos, como Dijkstra, A* e A* bidirecional ([Rachmawati; Gustin, 2020](#)) como opção para o usuário e como comparativo de rotas para estudo.

1.1 Objetivos

O *objetivo* deste trabalho é projetar um sistema web de roteamento geoespacial com softwares de código aberto, escalável e reproduzível sobre um grande volume de dados geográficos, utilizando o Brasil como escopo, com uma interface interativa para o usuário. E como *objetivos específicos*:

1. Importar para o banco de dados PostgreSQL, utilizando a ferramenta osm2pgsql, os dados geoespaciais do Brasil.
2. Configurar o ambiente do banco de dados para que as extensões, PostGis e pgRouting, sejam utilizadas para o cálculo do caminho mais curto.
3. Implementar o back-end utilizando a linguagem Python e o framework web FastAPI disponibilizando endpoints para o cálculo das rotas.
4. Desenvolver a camada de integração para utilizar o Nominatim, serviço open source de geocodificação, para não depender de serviços pagos.
5. Implementar o front-end, para que o usuário tenha a visão do mapa e rotas a partir das entradas de endereços, utilizando React e Leaflet
6. Configurar o arquivo do docker compose para as aplicações serem empacotadas, para que todas as dependências estejam em seu devido contêiner.
7. Comparar desempenhos e execução em distintos cenários de rotas do Brasil utilizando os algoritmos Dijkstra, A* e A* bidirecional já implementadas pela extensão pgRouting do PostgreSQL.

1.2 Metodologia

Na primeira etapa do trabalho, foram feitas pesquisas bibliográficas para refinamento técnico e melhor compreensão dos conceitos e ferramentas a serem utilizados. Diante da compreensão de todo o contexto do trabalho, foram escolhidas as tecnologias de código aberto com comunidade ativa, documentação consolidada e com uso estabelecido em produção, garantindo baixo custo, reprodutibilidade e maturidade suficiente para operar sobre dados em escala nacional.

Com a facilidade que o gerenciador de contêineres Docker oferece no início de qualquer projeto para configuração e testes, foi iniciada a configuração do banco de dados, do back-end e do front-end em seus distintos contêineres. Após os ajustes no contêiner do PostgreSQL para a importação dos dados geoespaciais, os dados do Brasil foram obtidos no Geofabrik ([Geofabrik GmbH, 2024](#)), já que o OpenStreetMap limita a área para exportação e download. Os dados geográficos do país utilizados foram importados utilizando a ferramenta chamada osm2pgsql para a análise do tempo de execução.

Diante da disponibilidade dos dados, foram desenvolvidos com FastAPI os endpoints de roteamento para buscar as informações solicitadas. O pgRouting tem os algoritmos Dijkstra, A* e A* bidirecional já implementados, facilitando a integração com a API no cálculo das rotas. Para uma melhor visualização das rotas em um mapa e usabilidade, foram utilizados React e Leaflet como tecnologias para o front-end, conectando via requisições HTTP no back-end próprio e na API Nominatim da OpenStreetMap.

Os testes foram executados comparando os três algoritmos em diversas situações de rotas para o caminho fornecido entre eles e com o serviço do Google Maps que é um serviço de roteamento geoespacial popular porém pago. Na análise do tempo, foi analisada a criação de índice espacial no PostGIS para um melhor estudo.

1.3 Organização do Trabalho

O documento do projeto está estruturado em 6 capítulos e 4 apêndices. O capítulo 2 está fundamentado nos conceitos teóricos para um entendimento de todo o projeto. O capítulo 3 apresenta pesquisas correlacionadas com o tema deste trabalho. A forma como foi desenvolvido o sistema está descrito no capítulo 4. Já no capítulo 5 estão os testes realizados e os resultados comparativos entre os algoritmos propostos na pesquisa. Fechando o trabalho, o capítulo 6 traz as considerações finais, os problemas identificados e melhorias que podem ser feitas no projeto.

Os 2 apêndices estão estruturados como: instalação e configuração do Docker e configurações dos contêineres do banco de dados, back-end e front-end.

Conceitos Fundamentais

Este capítulo apresenta os fundamentos teóricos e tecnológicos que sustentam o desenvolvimento do sistema de roteamento geoespacial proposto. A compreensão do trabalho exige o domínio de conceitos que se estendem desde a representação de dados geográficos até os algoritmos de caminho mínimo e as ferramentas empregadas em sua implementação.

Inicialmente, abordam-se os Sistemas de Informação Geográfica e os bancos de dados geográficos, que estabelecem as bases para o armazenamento e a manipulação de dados espaciais. Em seguida, examina-se o OpenStreetMap como fonte da malha viária utilizada, bem como o processo de importação e de construção da topologia da rede, etapa necessária para que os dados possam ser tratados como um grafo navegável. Na sequência, descreve-se o pgRouting, extensão responsável pelo processamento das rotas, e os algoritmos de caminho mínimo comparados neste trabalho: Dijkstra, A* e A* bidirecional. Por fim, apresentam-se as tecnologias adotadas na construção da aplicação, que abrangem as camadas de dados, de serviços e de interface.

2.1 Sistemas de Informação Geográfica

Um Sistema de Informação Geográfica(SIG) é um sistema computacional que proporciona suporte às fases de inserção dos dados georreferenciados, análise dos dados de diversas maneiras, como também, produção de apresentação de mapas e outras formas de demonstração a partir desses dados. Um dado georreferenciado, sinônimo de dado espacial ou geoespacial, tem atributos de coordenadas de um espaço geográfico onde pode se determinar, com precisão, o evento ou objeto a ser localizado (By *et al.*, 2001).

Os dados SIG podem ser representados basicamente através de dois modelos estruturais, o vetorial e o matricial, para demonstrar o mundo real. O modelo vetorial é caracterizado por ter coordenadas geográficas (x, y) e separadas em três categorias: pontos que representa apenas a característica da localização, linhas que é uma sequência de pontos e polígonos que é uma representação por linhas fechadas sem interseções possuindo localização, área e perímetro. Já o modelo matricial é ideal para representar espaços geográficos

como elevações e erosão do solo ou a temperatura, precipitações de chuva e entre outros. Para o estudo dirigido neste projeto foi adotado somente o modelo vetorial, pois os pontos e linhas formam grafos para se obter o caminho mais curto entre dois pontos (By *et al.*, 2001).

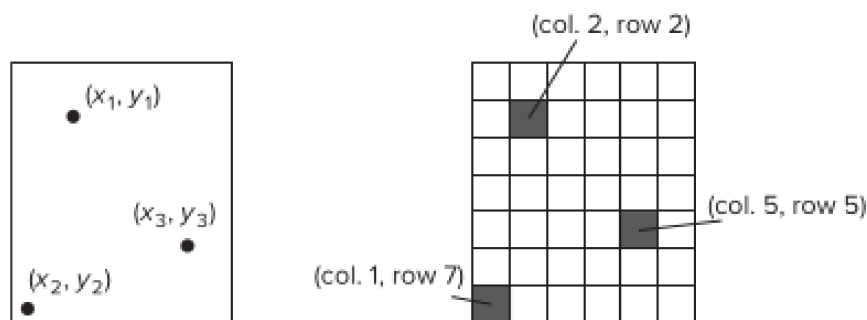


Figura 2.1 – Modelos vetorial e matricial

Fonte: Adaptado de (Chang, 2019).

O desempenho das consultas espaciais em SIG depende do modo que são indexados e da quantidade dos dados armazenados, pois se torna computacionalmente custoso a medida que o volume da base de dados cresce. Utilizando em uma rede viária de escala nacional, como o Brasil, uma busca linear sobre as tabelas do banco de dados pode inviabilizar o sistema por conta do tempo de resposta. Para diminuir esse gargalo, utiliza-se a indexação espacial por meio da estrutura de dados R-Tree (Casanova *et al.*, 2005). Esse tipo de indexação organiza os dados em uma árvore hierárquica, onde permite ter uma otimização na busca dos dados espaciais para que sejam processados por um algoritmo de caminho mínimo, como o Dijkstra, A* e A* bidirecional.

Para a implementação desse projeto utiliza-se o PostgreSQL como o Sistema de Gerenciamento de Banco de Dados (SGBD), a extensão PostGIS que analisa os dados geoespaciais (POSTGIS, 2026) e a extensão pgRouting que calcula o caminho mais curto (PGROUTING, 2026) na malha viária brasileira da OpenStreetMap. O conjunto dessas ferramentas nos possibilita ter um Sistema de Informação Geográfica (SIG) de código aberto, escalável, reproduzível e interativo para o usuário final (Casanova *et al.*, 2005).

2.2 Bancos de Dados Geográficos

Um banco de dados geográfico consiste em um SGBD com extensões para armazenar, indexar, manipular e consultar dados espaciais, preservando as estruturas geométricas e os atributos alfanuméricos no mesmo banco de dados. Na área do roteamento, a estrutura integrada das extensões facilita a representação do grafo de uma malha viária por meio dos seus vértices, arestas e custos de um ponto ao outro (Casanova *et al.*, 2005).

2.2.1 Estrutura dos Dados Espaciais

No modelo vetorial, a malha viária é formada pela interconexão de segmentos de vias, que são representados pelas linhas, e os pontos de interseção são representados pelos nós correspondendo diretamente a uma estrutura de grafo. Essa mesma estrutura está em conformidade com a teoria dos grafos, onde os nós de interseção equivalem aos vertices e os trechos das vias correspondem às arestas (Casanova *et al.*, 2005).

Em cada trecho da via existem atributos associados para representá-lo como o nome, tipo, velocidade máxima, sentido, entre outros. Essas informações facilitam o cálculo do custo do percurso da aresta, valor esse utilizado no algoritmo do caminho mais curto entre dois pontos, devido à correspondência direta entre a estrutura de dados vetorial e a topologia do grafo (Casanova *et al.*, 2005).

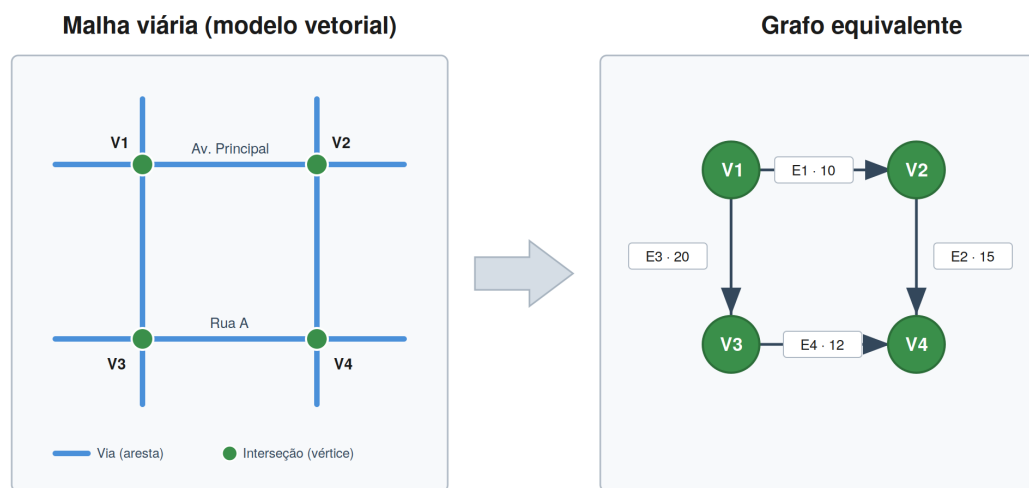


Figura 2.2 – Correspondência entre a malha viária e o grafo de roteamento

Fonte: Autor.

A Figura 2.2 ilustra essa correspondência entre a malha viária e o grafo resultante.

2.2.2 PostgreSQL e PostGIS

O PostgreSQL é um sistema gerenciador de banco de dados objeto-relacional (SGBD-OR) de código aberto e gratuito, que está em total conformidade com as normas da Linguagem de Consulta Estruturada (*Structured Query Language* —SQL). Uma particularidade que faz esse banco de dados ter uma vantagem é o potencial de agregar novas extensões, permitindo novos tipos de dados, funções e operadores ligados ao sistema. Esse mecanismo possibilita a extensão do SGBD-OR sem modificar a sua estrutura, oferecendo a infraestrutura para integrar extensões geográficas.

Utilizado como uma extensão do PostgreSQL, o PostGIS permite que adicione suporte a objetos geográficos, transformando-o em um banco de dados com dados espaciais em conformidade com as especificações do Open Geospatial Consortium (OGC). A extensão disponibiliza tipos de dados geométricos, como pontos e linhas e funções para análises de medição de distância e topológicas, além de fornecer a possibilidade de indexação espacial por meio do GiST (*Generalized Search Tree*). As consultas são eficientes pois a malha viária e o grafo do roteamento pode ser armazenadas e consultados diretamente no banco de dados ([Casanova et al., 2005](#)).

2.2.3 Tipos Espaciais e SRID

No Sistema de Gerenciamento de Banco de Dados espacial, as estruturas geométricas são representadas por meio de tipos vetoriais, como o ponto para uma localização única, o *LINESTRING* para aspecto unidimensional, o polígono para delimitações de áreas e suas versões semelhantes como MULTIPOLYGON. No roteamento viário, a *LINESTRING* é utilizada para a representação de cada trecho da via, determinando toda a estrutura do grafo de roteamento ([Casanova et al., 2005](#)).

Além disso, toda instancia geométrica tem um identificador dentro do Sistema de Referência Espacial (SRID), que definem como as coordenadas se associam com as posições geográficas na Terra ([Casanova et al., 2005](#)). O sistema de referência utilizado é o WGS 84, podendo ser identificado pelo código SRID 4326, funciona como referência global para o GPS e para a base de dados do OpenStreetMap, no qual utiliza-se como padrão neste trabalho ([OpenStreetMap Wiki, 2026](#)).

2.3 OpenStreetMap

O OpenStreetMap (OSM) trata-se de um projeto em que há participação coletiva dos usuários do software para a formulação de um mapeamento do globo terrestre, sendo sua criação advinda do empreendedor britânico Steve Coast no ano de 2004, objetivando a oferta de informações de cunho geográfico com acesso livre e passível de edição por qualquer usuário ([Elias et al., 2020](#); [Chang, 2019](#)). A base cartográfica utilizada neste programa é formulada a partir da ideia de Informação Geográfica Voluntária (VGI), onde os dados são gerados através da colaboração pública das pessoas, através da sua inserção por uso de redes de internet ([Elias et al., 2020](#); [Chang, 2019](#)). Esse formato permite que os usuários também trabalhem na alimentação da plataforma, de modo a participarem como sensores voluntários, que transformam os dados que coletaram em informação pública, editando e disponibilizando suas descobertas, sem cobranças de direitos autorais que costumam ocorrer em bases proprietárias ([Elias et al., 2020](#); [Li et al., 2016](#)).

O formato do banco de dados do OpenStreetMap ocorre através de uma divisão das informações em 3 vertentes distintas, sendo compostas pelos elementos básicos que repre-

sentam a visualização da terra, sendo os elementos os nós (*nodes*), os caminhos (*ways*) e as relações (*relations*), que juntos formam uma interação que posteriormente será representada por um etiqueta (*flag*) que é distinta para cada descrição criada com o cruzamento das informações no ponto em questão (Chang, 2019).

De modo mais específico, os nós simbolizam lugares específicos da superfície terrestre, possuindo latitude, longitude e um identificador único. Já os caminhos são listas ordenadas em que a representação visual ocorre de maneira linear com traços ou formas, permitindo verificar traçados de caminhos ou áreas de locais específicos. Por último as relações agrupam de maneira racional vários elementos, para que a informação não seja apenas um elemento básico, mas o contexto completo, como por exemplo de rotas de transporte (Chang, 2019).

Quando se fala sobre a malha viária deste software, tem-se que a mesma é formada por um banco de dados próprio para sistemas de roteamento e navegação (Li *et al.*, 2016). Sendo um modelo representado visualmente por vetores com características topológicas, há uma ligação entre os caminhos de vias e arcos, com junções formadas por nós de interseção, que formam a rede completa disponível no OpenStreetMap (Chang, 2019). Com essa construção é possível atribuir propriedades de operação de fluxo para os arcos, como por exemplo restrições de sentido de tráfego e dados financeiros de percurso, conforme o peso das arestas (Casanova *et al.*, 2005). Com essa modelagem de relações do grafo de vias, os algoritmos de formação de caminhos mínimos podem processar o trajeto ótimo existente entre origem e destino escolhidos previamente (Casanova *et al.*, 2005; Chang, 2019).

Apesar de análises em escala planetária concluírem que há alta completude da malha viária no OpenStreetMap, contabilizada em números superiores a 80% (Barrington-Leigh; Millard-Ball, 2017), a qualidade das informações apresenta grande heterogeneidade do território (Elias *et al.*, 2020). A nível nacional, as metrópoles do país como Salvador (BA), são verificadas em estudos experimentais com completude do arruamento em mais de 80% de áreas verificadas, apesar de possuir lacunas relevantes, além de apresentar quase 60% dos eixos viários desprovidos de toponímia (Elias *et al.*, 2020). A completude dessa malha também pode posteriormente ser ampliada por importações em massa de informações governamentais ou oficiais (Barrington-Leigh; Millard-Ball, 2017); esses trabalhos consolidam e aceleram a cobertura das vias, inserindo mais dados no sistema, porém, depende de procedimentos de controle e consistência que assegure a integridade da base geográfica (Elias *et al.*, 2020).

2.4 Importação de Dados OSM para o PostgreSQL

2.4.1 osm2pgsql

O `osm2pgsql` é uma ferramenta de linha de comando utilizada para adição de dados advindos do OSM, onde sua distribuição é transportada do formato Protocolbuffer Binary Format (PBF), para um banco de dados PostgreSQL estendido com o PostGIS, o que permite gerar o produto em forma de tabelas espaciais correspondentes ([Osm2pgsql developers, 2026](#)). Dentre as formas de saída, a maneira seriada e eficiente da `pgsql` foi criada justamente para criar renderizações e apresentações cartográficas com as informações dos geradores de mapas, como o Mapnik ([Osm2pgsql developers, 2026](#)). Assim, utilizar essa saída produz um conjunto de quatro tabelas espaciais, compostas pelos elementos *point*, *line*, *roads* e *polygon*, em que é nomeado sempre pelo prefixo padrão do sistema `planet_osm` ([Osm2pgsql developers, 2026](#)).

Optar então pelo `osm2pgsql`, frente ao `osm2pgrouting`, ocorre justamente por sua escala nacional presente no conjunto de dados. Enquanto o `osm2pgrouting` atua com a leitura integral dos dados de entrada no processamento o `osm2pgsql` tem a opção de atuar no modo *slim* (opção `--slim`), com dados temporários ([PGROUTING, 2026](#)). Assim, o primeiro precisa de informações de menor porte para sua execução, enquanto o segundo permite armazenagem temporárias de dados em seu próprio banco de dados, permitindo grandes bases de informações mesmo que a memória do equipamento seja limitada ([Osm2pgsql developers, 2026](#)). De qualquer modo, ambos os processos são exigentes quanto à memória, sendo regra prática ter sempre memória disponível correspondente ao tamanho do arquivo PBF que será importado ([Osm2pgsql developers, 2026](#)).

A inviabilidade do `osm2pgrouting` para o extrato nacional foi verificada empiricamente durante o desenvolvimento. Como a ferramenta processa arquivos no formato OSM XML, o extrato em PBF precisou ser previamente convertido com o utilitário `osmium` ([Topf, 2026](#)), resultando em um arquivo de aproximadamente 50 GB. Mesmo com o limite de memória do contêiner elevado para 80 GB, o processamento foi interrompido pelo sistema operacional durante a leitura dos dados, e a alternativa de fracionar a importação por unidade federativa mostrou-se operacionalmente custosa. Esse resultado motivou a adoção do `osm2pgsql` em modo *slim*, que realiza a leitura direta do formato PBF com apoio de tabelas temporárias no próprio banco de dados.

No presente trabalho, a importação partiu de um arquivo PBF de aproximadamente 2 GB, correspondente à malha viária completa do Brasil, e foi executada em modo *slim*. Desse processo resultou a tabela `planet_osm_roads`, com 661.095 registros, que constitui o conjunto de segmentos viários empregado como base para a construção do grafo.

2.4.2 Construção da Topologia

A partir do momento em que os dados importados são de cunho geométrico, sem informações explícitas sobre a conectividade entre os segmentos ([Osm2pgsql developers, 2026](#)), a topologia foi gerada por uma rede viária, que é etapa obrigatória para que possa realizar o processamento das rotas ([PGROUTING, 2026](#)). Já para a formação estrutural, utilizou-se a função `pgr_createTopology`, disponibilizada pelo `pgRouting` ([PGROUTING, 2026](#)).

Com uma tabela de arestas então, junto a um limite de tolerância de junção (*snapping tolerance*), a função criada identifica a proximidade entre extremidades dos segmentos de linha e acaba por preencher as colunas dos nós de origem (*source*) e destino (*target*) dentro da tabela de arestas, criando uma tabela de vértices que é nomeada com o sufixo `_vertices_pgr` ([PGROUTING, 2026](#)). No trabalho é verificado então que essa etapa gerou como produto a tabela `planet_osm_roads_vertices_pgr`. Com a realização desse procedimento, os segmentos compõem um grafo formado por interseções (vértices) e vias (arestas) ([PGROUTING, 2026](#)). A unidade para a tolerância é subjetiva e sua interpretação depende do sistema de referência espacial (SRID) relacionado aos dados ([PGROUTING, 2026](#)). Ainda para a base do projeto aqui realizado, foi adotada a tolerância de 1,0, em unidades da projeção Web Mercator (SRID 3857), cuja unidade linear é o metro.

2.5 pgRouting

Nomeado como `pgRouting`, a extensão por meio de código aberto do sistema PostgreSQL é integrada à extensão espacial PostGIS, para que sejam inseridas funcionalidades de roteamento e análise de redes de banco de dados ([PGROUTING, 2026](#)). Por meio de sua operação diretamente sobre o grafo armazenado, essa extensão acaba por unificar no mesmo local o armazenamento de dados espaciais e o processamento das verificações sobre caminho mínimo, sem que haja a necessidade de fazer a exportação da rede para ferramentas externas ([PGROUTING, 2026](#)).

O processamento das rotas pode ser realizado a partir de uma rede viária que esteja representada por tabela de arestas com estrutura específica. No caso, cada uma das arestas precisa ser minimamente caracterizada por colunas, *source*, *target*, *id* e *cost* ([PGROUTING, 2026](#)), sendo que as duas primeiras, respectivamente, correspondem aos vértices do início e fim do trecho, referenciando identificadores da tabela de vértices, já a coluna *cost* é responsável pelo armazenamento do peso da aresta ([PGROUTING, 2026](#)). Ainda é possível haver uma coluna extra opcional chamada *reverse_cost*, para conter o custo no sentido inverso, para casos em que o grafo é tratado como direcionado ([PGROUTING, 2026](#)). No presente trabalho, foi escolhido o grafo não direcionado, onde cada trecho viário é transitável para ambos os sentidos; consequentemente, as restrições de tráfego por vias de mão única não são modeladas, criando uma limitação já assumida neste trabalho.

Para realizar o cálculo de rotas, a extensão escolhida disponibiliza diversas funções de busca (PGROUTING, 2026). Tem-se a função `pgr_dijkstra`, que implanta o algoritmo de Dijkstra para verificar o caminho de custo mínimo entre vértices (PGROUTING, 2026). Já a função `pgr_aStar` implementa o algoritmo A^* , que utiliza a estimativa heurística para encaminhar a verificação dos vértices em direção ao destino (PGROUTING, 2026). Por último a função `pgr_bdAstar` trabalha com a implementação da versão bidirecional do A^* , que atua realizando duas buscas heurísticas simultâneas, a partir da origem e do destino (PGROUTING, 2026). De maneira igual, as três permitem trabalhar sobre grafos direcionados ou não, através do parâmetro *directed* (PGROUTING, 2026).

A integração com o PostGIS aumenta as possibilidades de aplicação dessas funções (PGROUTING, 2026). Através da função espacial `ST_Distance`, são relacionadas as coordenadas de origem e destino previamente informadas pelo usuário ao vértice que se encontra mais próximo à rede. Com o uso da função `ST_Length`, é calculado o comprimento de extensão de cada aresta conforme sua geometria, armazenando os dados na coluna *way*, e assim, completando as informações da coluna de custo do grafo. Ao finalizar a busca, a união entre o produto retornado pela função de roteamento e a tabela de arestas forma o traçado completo da rota, permitindo então sua reprodução sobre o mapa.

2.6 Algoritmos de Caminho Mínimo

Dificultando o caminho mínimo em grafos ponderados, o problema encontrado nessa execução encontra-se em determinar o trajeto onde o somatório dos pesos das arestas percorridas seja o mínimo entre os vértices de origem e destino. A modelagem da rede viária é feita pelo grafo $G = (V, E)$ onde intersecções correspondem aos vértices (v) e trechos de via correspondem às arestas (e), com seus respectivos pesos de grandezas a serem minimizadas, sendo elas a distância e tempo de percurso (Casanova *et al.*, 2005). Dessa maneira, as seções a seguir deste trabalho descrevem os três algoritmos empregados, sendo todos disponibilizados pelo pgRouting.

2.6.1 Algoritmo de Dijkstra

O algoritmo de Dijkstra é o método encontrado para resolver o problema do caminho mínimo, sendo iniciado com uma origem única em grafos com pesos que não sejam negativos, assim, o algoritmo preserva para cada vértice uma expectativa de distância contada a partir da origem relacionado-a a um conjunto de vértices com caminhos mínimos anteriormente determinados (Dijkstra, 1959; Cormen *et al.*, 2012). Assim, os vértices que ainda não foram processados ficam organizados em ordem de prioridade mínima, onde são retirados os de menor estimativa, com arestas adjacentes sendo então relaxadas (Cormen *et al.*, 2012). Por fim, no final do trajeto, o caminho ótimo é reconstruído com a utilização

da lista de precedentes registrada neste relaxamento, sendo restrita a admissão de pesos negativos nas arestas (Cormen *et al.*, 2012).

2.6.2 Algoritmo A*

Este algoritmo é uma metodologia de procura de origem heurística onde há generalização do algoritmo de Dijkstra através da função avaliativa $f(x) = g(x) + h(x)$, onde $g(x)$ significa o custo acumulado da origem ao vértice corrente e $h(x)$ trata-se de uma expectativa de custo que falta para o destino (Rios; Chaimowicz, 2011). Diferente do Dijkstra que aumenta a busca uniformemente em todas as direções, o A* orienta a busca diretamente ao seu destino, reduzindo o número de vértices da expansão (Rios; Chaimowicz, 2011). Ainda para garantir a solução ótima, essa função precisa ser admissível, nunca aumentando o custo real restante (Rios; Chaimowicz, 2011). No caso das redes viárias, é adotada uma distância euclidiana em linha reta para formar um limite inferior que não consegue superestimar a distância do trajeto (Rios; Chaimowicz, 2011). A eficácia do algoritmo então depende da qualidade da estimativa diretamente, ou seja, quando mais precisa é a heurística, menor o número de vértices expandidos, formando uma relação de resultados inversamente proporcionais.

2.6.3 A* Bidirecional

Por último quando utiliza-se o A* bidirecional, há uma combinação entre os dois métodos anteriores, onde as duas ocorrem simultaneamente, sendo uma progressiva iniciada na origem, e a outra regressiva iniciada no destino (Rios; Chaimowicz, 2011). A justificativa e motivação para seu uso encontra-se na possibilidade de redução do número de vértices expandidos, contando com porções menores do grafo exploradas em cada execução até o encontro de ambas no intermédio do percurso (Rios; Chaimowicz, 2011). Ressalta-se que essa busca não conseguirá superar a unidirecional todas as vezes, verificado que as primeiras pesquisas do tema não resultaram em ganhos significativos e a eficácia é atrelada a fatores como critério de parada e qualidade da heurística (Rios; Chaimowicz, 2011). Ainda assim, estudos levantam essa metodologia como alternativa eficaz para grandes redes viárias reais (Rios; Chaimowicz, 2011).

2.6.4 Comparação

Os três algoritmos visam oferecer um caminho ótimo, porém, diferem em relação ao uso de heurística e as maneiras como ocorrem os esforços de exploração do grafo em cada caso. Para o algoritmo de Dijkstra é utilizada a busca uniforme, sem necessidade de estimativas, já o A* e o A* bidirecional dependem da função heurística, pois atuam em buscas direcionadas ao destino, visando a redução da expansão de vértices, principalmente em casos de redes de grande porte (Rios; Chaimowicz, 2011; Rachmawati; Gustin,

2020). Já em relação à complexidade de tempo, é relatado no manual do pgRouting que as buscas por origem e destinos únicos, a ordem $O((E + V) \log V)$ para as funções `pgr_aStar` e `pgr_bdAstar`, e a ordem equivalente na função `pgr_dijkstra` (PGROUTING, 2026). Ainda é preciso lembrar que a função `pgr_dijkstra` é descrita como complexidade $O(|start_vids| \cdot (V \log V + E))$, onde o termo $|start_vids|$ é referente ao número de vértices na origem, o que, para esta pesquisa, que possui origem e destinos únicos, faz com que se torne um termo unitário, com expressão reduzida para $O(V \log V + E)$ (PGROUTING, 2026). Diferenciando os métodos pelo desempenho, é possível verificar que atua com maior diferenciação no número de vértices expandidos na busca e menos em relação e ordem assintótica de pior caso (Rachmawati; Gustin, 2020).

2.7 Tecnologias de Desenvolvimento

Esta seção descreve as principais tecnologias empregadas na construção da interface e da camada de serviços do sistema.

2.7.1 React

Definido como biblioteca JavaScript de código aberto, o React é uma ferramenta voltada para a formulação de interfaces de usuários, sendo seu uso público oferecido no ano de 2013 pela Facebook, e mantida até os dias atuais pela Meta, juntamente aos contribuintes (Meta Platforms, 2013). O modelo de evolução de seu programa é baseado em componentes, unidades reutilizáveis que formam uma estrutura encapsulada, com visualização e comportamento de partes da interface (Meta Platforms, 2013). Cada um desses componentes acaba realizando o gerenciamento de seu próprio estado interno e, quando o mesmo sofre algum tipo de mudança, o React trabalha renovando os trechos afetados na interface, atualizando estes trechos conforme as alterações ocorridas (Meta Platforms, 2013). Assim, no trabalho, a ferramenta atua como construtora de interfaces web através de usuários que apresentam origem e destino que desejam buscar, para visualizar a rota desejada.

2.7.2 Leaflet

Já o Leaflet é uma biblioteca que trabalha como banco de dados do JavaScript de código aberto, tornando possível a construção de mapas interativos. Essa biblioteca teve sua criação em 2011, por Volodymyr Agafonkin e é descrita como biblioteca de tamanho reduzido, tendo apenas aproximadamente 42 KB de código JavaScript. Ademais, ela é responsável por reunir recursos de mapeamento que possam ser necessários para grande parte das execuções (Agafonkin, 2026). A biblioteca é uma forma de suporte às exibições de camadas de *tiles*, também auxilia na inserção de marcadores e polilinhas e no carrega-

mento de dados geográficos que estejam no formato GeoJSON ([Agafonkin, 2026](#)). Para o presente trabalho, o Leaflet é utilizado como meio de renderização do mapa e também pela visualização das rotas que retornam ao servidor solicitante, sem que dependa de serviços cartográficos prioritários.

2.7.3 GeoJSON

A tecnologia GeoJSON trata-se de uma plataforma pública de intercâmbio de dados geoespaciais, que executa programas baseada na notação JavaScript Object Notation (JSON) e padronizado pela IETF na RFC 7946 ([Butler *et al.*, 2016](#)). Sua forma de apresentação define os tipos de objetos que serão apresentados nas feições geográficas, nas propriedades e extensões espaciais, podendo ser utilizados para esse fim pontos, linhas e polígonos ([Butler *et al.*, 2016](#)). Devido à sua especificação, as coordenadas da plataforma adotam o sistema de georreferenciamento WGS 84 (EPSG:4326), que possui longitude e latitude descritas com graus decimais ([Butler *et al.*, 2016](#)).

A listagem da Tabela 2.1 a seguir mostra um exemplo de como é a representação de uma geometria do tipo `LineString` em GeoJSON, semelhante ao traço de uma rota, onde para qual par de valores há uma coordenada com longitude e latitude correspondente na tabela.

Tabela 2.1 – Exemplo de representação de uma geometria `LineString` em GeoJSON

```
{
  "type": "LineString",
  "coordinates": [
    [-48.2772, -18.9186],
    [-48.2765, -18.9179],
    [-48.2751, -18.9168]
  ]
}
```

Fonte: Elaborado pelo autor.

No trabalho realizado, o GeoJSON foi utilizado como forma de intercâmbio entre a camada de dados e a interface. Depois de calcular a rota, os segmentos que formam a mesma foram unidos para converter os dados para o GeoJSON por meio da função `ST_AsGeoJSON` do PostGIS, por meio da camada de serviços implementada em FastAPI. Devido a diferentes de projeção, sendo o roteamento operado na projeção Web Mercator (SRID 3857) e o GeoJSON atuante no WGS 84, é realizada a conversão com uso da função `ST_Transform` na geometria inicial da rota, para que a mesma se encontre no modelo SRID 4326. Após essa similaridade de projeções, o resultado pode então ser encaminhado para à interface, onde o Leaflet interpreta o GeoJSON e faz a renderização da rota mostrando seu traçado no mapa.

2.7.4 FastAPI

O FastAPI é um framework Python de código aberto utilizado para construir interfaces de programação de aplicações (APIs), sendo sua criação atribuída a Sebastián Ramírez ([Ramírez, 2018](#)). O FastAPI é implementado sobre o Starlette, que gerencia a camada web assíncrona, e também é implementado sobre o Pydantic, que valida as informações a partir de dados de tipo da linguagem ([Ramírez, 2018](#)). Como detalhamento de seu sistema, encontram-se o suporte para execuções assíncronas, validações automáticas de informações de entrada e criação de produtos automáticos de documentação interativa que segue o padrão OpenAPI ([Ramírez, 2018](#)). Conforme sua descrição oficial, seu desempenho pode ser comparado ao de outros frameworks como o Node.js e o Go ([Ramírez, 2018](#)). Para este trabalho, a ferramenta implementou a camada de serviços responsável por receber requisições de usuários da interface, acionando o roteamento no banco de dados e gerando as rotas calculadas como produto final.

2.7.5 Nominatim

O Nominatim é um serviço de geocodificação de código aberto desenvolvido sobre os dados do OpenStreetMap, na função de converter endereços e nomes de lugares em coordenadas geográficas ([OpenStreetMap Foundation, 2026](#)). Sustentado pela comunidade do OpenStreetMap, disponibiliza para consultas públicas por meio de chamadas em uma API HTTP, que retorna, para um termo de busca, uma lista de resultados com as respectivas coordenadas e informações de endereço. É utilizado nesse projeto como camada de busca textual, convertendo os endereços de origem e destino informados pelo usuário nas coordenadas em seguida relacionadas ao grafo viário.

2.7.6 Docker e Docker Compose

Como opção ao problema chamado de *dependency hell*, onde implantações modernas com diversos componentes entram em conflitos pelos diferentes requisitos que são incompatíveis entre si, surge a containerização ([Merkel, 2014](#)). Quando utilizada essa containerização viabiliza a aglutinação implementações junto à suas próprias dependências em uma unidade padronizada e isolada ([Merkel, 2014](#)). Esse modelo de isolamento garante que o sistema trabalhe de maneira eficaz em ambientes diversificados de desenvolvimento, teste e produção, mesmo havendo diferenças de distribuição ou de infraestruturas subjacentes ([Merkel, 2014](#)).

Dentro da plataforma Docker, há uma diferenciação sobre os conceitos de imagem e contêiner, sendo o primeiro um modelo estático, que não aceita mudanças e preserva sua estrutura em forma de camadas sobrepostas, para montar um sistema de arquivos de base prontos para uso ([Merkel, 2014](#)). Já a segundo é composto pela parte ativa gerada a partir de uma imagem, onde trabalha em espaço isolado limitado em nível operacional

do sistema, com uso de recursos do núcleo do Linux, como *namespaces* e *control groups* (*cgroups*) (Merkel, 2014). Esse modelo de trabalho garante então a possibilidade de portabilidade, permitindo a variação entre os sistemas por sua compatibilidade (Merkel, 2014). Evidencia-se que na definição original do modelo, a implementação nativa dos contêineres era limitada a distribuições Linux, que acabavam por exigir a aplicação de ferramentas de virtualização leves que pudessem operar sem problemas em sistemas operacionais como Windows e OS X (Merkel, 2014).

Quando se trata do gerenciamento dessas aplicações de serviços variados, o Docker Compose acaba sendo a ferramenta ideal que possibilita a definição e execução de contêineres variados através de um único arquivo de configuração no formato YAML (Docker, Inc., 2026). É então considerado uma ferramenta declarativa, onde o arquivo faz a descrição dos serviços, redes e volumes da aplicação, para que assim o conjunto tenha como ser iniciado após um único comando (Docker, Inc., 2026). Ainda dentro do escopo do projeto que visa o roteamento geoespacial, é escolhida uma infraestrutura em que serão organizados os principais serviços do sistema — o banco de dados geográfico, o servidor de aplicação e a interface com o usuário — dentro de contêineres isolados, gerenciado de forma unificada pelo Docker Compose. Trata-se de uma interação de acoplamento flexível que se apoia no princípio de responsabilidade única, conforme verificado que cada contêiner terá responsabilidade sobre apenas um serviço específico da arquitetura (Merkel, 2014).

2.8 Escalabilidade

A escalabilidade mede o quão bem um sistema consegue lidar com mais tarefas ou volumes maiores de dados antes que o seu desempenho seja afetado (Bondi, 2000). Isso pode ser feito adicionando mais recursos (como mais hardware) ou utilizando arquiteturas eficientes.

A escalabilidade pode acontecer como “escalabilidade de carga”, quando o sistema consegue processar um número maior de requisições ao mesmo tempo; como “escalabilidade espacial”, quando há volumes maiores de dados para armazenar e/ou processar; e como “escalabilidade estrutural”, que ocorre quando a estrutura do sistema não precisa ser alterada só porque o volume de dados ou de usuários aumentou (Bondi, 2000).

Este trabalho foca principalmente na escalabilidade espacial. A dimensão da malha viária brasileira torna o volume de dados muito maior do que o usado no estudo de caso de uma única cidade. Existem muito mais trechos de vias para importar e armazenar no banco de dados, além de mais segmentos que precisam ser analisados para encontrar o caminho mais curto entre dois pontos.

Várias decisões de arquitetura foram tomadas para lidar com esse problema. Por exemplo, na Seção 2.4.1, onde o `osm2pgsql` é discutido em relação à importação de dados para

o PostgreSQL com PostGIS, explica-se que, ao rodar o `osm2pgsql` no modo *slim*, ele salva os dados temporários no disco rígido em vez de manter tudo na memória RAM.

Além disso, como o `pgRouting` realiza as operações de roteamento inteiramente dentro do próprio banco de dados, nenhum dado precisa sair dali para ser enviado e processado por outra ferramenta externa.

A viabilidade dessas escolhas de projeto foi testada no Capítulo 5, medindo tanto os tempos de importação quanto os tempos de resposta com a base de dados nacional.

Trabalhos Correlatos

Neste capítulo são apresentados três tópicos: sistemas de roteamento baseados em dados do OpenStreetMap e na extensão pgRouting; estudos comparativos sobre o desempenho de algoritmos de caminho mínimo em redes viárias; e técnicas de roteamento para grandes redes viárias. O projeto em desenvolvimento nesse trabalho permite comparar com os estudos citados pois o sistema de roteamento é baseado na Web para redes viárias no Brasil, compara vários algoritmos e usa uma arquitetura baseada em contêineres.

3.1 Sistemas de Roteamento sobre OpenStreetMap e pgRouting

A extensão pgRouting adiciona ao PostgreSQL/PostGIS um conjunto de várias funções para o cálculo de rotas sobre grafos que estão armazenados no banco de dados, e isso a torna uma referência de uso em sistemas de roteamento. Nesta seção, há trabalhos que utilizam o pgRouting, de maneira singular ou comparando com outras tecnologias, em diferentes cenários de navegação e análise de redes viárias.

[Chadha e Garg \(2019\)](#) desenvolveram um sistema de roteamento sobre a malha viária da Índia utilizando o pgRouting integrado ao QGIS. O estudo compara três funções de caminho mínimo disponíveis na extensão, Dijkstra, Dijkstra bidirecional e A*, avaliando o tempo de resposta e o custo das rotas obtidas. Por empregar o pgRouting em escala nacional e contrastar diferentes algoritmos, é o trabalho mais próximo da presente proposta. A diferença central está na arquitetura e no conjunto de algoritmos: enquanto [Chadha e Garg \(2019\)](#) operam a partir de um *plugin desktop* no QGIS e recorrem ao Dijkstra bidirecional, uma busca bidirecional sem heurística, este trabalho oferece um sistema web independente e inclui o A* bidirecional (`pgr_bdAstar`), combinação que associa busca bidirecional e heurística e que não é avaliada pelos autores. Acrescenta-se ainda a distribuição containerizada, ausente naquele estudo, que favorece a reprodutibilidade dos experimentos.

[Nik Yusoff et al. \(2014\)](#) apresenta um sistema *web-GIS* utilizando o pgRouting para cálculo de rotas com desvio de adversidades, no qual alguns caminhos estão temporariamente intransitáveis são removidos do grafo antes da busca. O roteamento é feito com o algoritmo de Dijkstra. A proposta do trabalho é a entrega de um sistema web usando o pgRouting, mas há a restrição de se utilizar somente um único algoritmo e o foco no tratamento de obstáculos, diferente do apresentado neste trabalho, que tem um das suas discussões a comparação de três algoritmos em escala nacional, a escalabilidade e a portabilidade da solução por meio de contêineres.

[Zhang e He \(2012\)](#) empregam o pgRouting em um mecanismo de busca de rotas (*Route Search*) que explora a hierarquia da malha viária para acelerar consultas em escala provincial. O trabalho enfatiza a organização hierárquica das vias como estratégia de desempenho. Diferentemente, este trabalho não modela a hierarquia viária de forma explícita e adota escala nacional, tendo como eixo a comparação entre Dijkstra, A* e A* bidirecional e a distribuição containerizada do sistema completo.

[Miller, Medak e Odobašić \(2014\)](#) realizam uma comparação entre o pgRouting, apoiado em banco relacional, e o Neo4j, um banco de dados orientado a grafos, aplicando o algoritmo de Dijkstra sobre a malha viária da Áustria. O objetivo é avaliar qual tecnologia de armazenamento responde melhor ao problema de caminho mínimo. Enquanto o estudo se volta à comparação entre tecnologias de banco de dados com um único algoritmo, este trabalho mantém-se no pgRouting e desloca a comparação para o plano dos algoritmos, três deles, incluindo o A* bidirecional, entregando ainda um sistema web containerizado de escala nacional.

3.2 Comparação de Algoritmos de Caminho Mínimo em Redes Viárias

Nesta parte, são comparados três algoritmos de caminho mínimo, adaptados especificamente para um ambiente de rede viária. Além disso, é revisada uma pesquisa de outra área relevante para o uso da busca bidirecional com heurísticas, avaliando assim um dos algoritmos utilizados neste projeto que ainda não foi amplamente estudado em redes viárias. [Rachmawati e Gustin \(2020\)](#) avaliaram o A* e o Dijkstra tendo a cidade de Medan como caso de aplicação. Ambos foram analisados comparando a qualidade das rotas encontradas e o tempo que cada um levou para executar. Confirmou-se que ambos se comportam como o esperado em uma rede urbana de pequeno porte. O caminho seguido por [Rachmawati e Gustin \(2020\)](#) é expandido neste trabalho de duas maneiras. Primeiro, avalio os algoritmos em escala nacional, utilizando a malha viária do Brasil. Segundo, introduzo um terceiro algoritmo, o A* Bidirecional. Para o A* Bidirecional, desenvolvi o código para rodar em um sistema web containerizado.

Rios e Chaimowicz (2011) detalham uma versão paralela e bidirecional do A*. Essa versão paralela foi projetada para resolver quebra-cabeças e labirintos, sendo muito relevante por descrever em detalhes o processo de execução de uma busca bidirecional com heurística. Como mencionado antes, embora Rios e Chaimowicz (2011) não resolvam problemas em redes viárias, o estudo deles dá uma base sólida para criar soluções de busca bidirecional com heurística aplicadas a esse tipo de rede. Para realizar essa tarefa, o `pgr_bdAstar` do `pgRouting` é utilizado para analisar e comparar esses três métodos. Embora os estudos anteriores apresentem técnicas bem conhecidas para busca bidirecional com heurística em sistemas baseados em grafos, quase não existem pesquisas investigando a aplicação ou análise desses métodos em malhas viárias nacionais de grande escala. Portanto, é justamente essa lacuna no conhecimento que este projeto busca preencher.

3.3 Roteamento em Redes Viárias de Grande Escala

O roteamento em redes viárias de grande escala, em nível nacional e continental, exige um desempenho muito alto. Por causa disso, vários pesquisadores desenvolveram métodos específicos. Esta seção analisa algumas das principais pesquisas nessa área para mostrar onde a nossa proposta se encaixa nesse contexto.

Dibbelt, Strasser e Wagner (2015) descrevem uma situação em que o custo do trajeto muda a cada consulta, o que impede o uso de várias técnicas comuns de pré-processamento. Por isso, eles sugerem melhorias usando apenas a estrutura da rede (a topologia), incluindo o *TopoCore*, que remove pontos de conexão com poucas ligações. Em redes viárias reais e muito grandes, eles conseguiram responder a consultas de longa distância em milissegundos, superando resultados anteriores. Esse estudo resolve um problema bem mais difícil e diferente do nosso: enquanto a nossa proposta trabalha com custos fixos e usa funções padrão do `pgRouting`, o trabalho de Dibbelt, Strasser e Wagner (2015) lida com custos variáveis usando um código próprio em C++, sem interface para o usuário. Assim, o objetivo deste trabalho não é ser mais rápido do que o deles em tempo de execução, mas sim oferecer um sistema web completo e fácil de rodar com contêineres, usando três algoritmos prontos do `pgRouting` e sem exigir conhecimentos avançados de engenharia de algoritmos.

Luxen e Vetter (2011) apresentam o OSRM (*Open Source Routing Machine*), um motor de roteamento de código aberto baseado em dados do OSM que usa *Contraction Hierarchies* para responder rápido a consultas em escala continental. O OSRM é uma grande referência em desempenho, com tempos de resposta muito menores do que buscas sem pré-processamento. No entanto, ele é um sistema focado apenas no processamento de rotas, sem interface visual, que usa uma única técnica (busca bidirecional sem heurística) e exige compilação e pré-processamento em C++. A proposta deste trabalho não tenta competir com o OSRM em velocidade pura. Em vez disso, entregamos uma aplicação web

completa e pronta em contêineres, que compara três algoritmos diferentes (incluindo o A* bidirecional), priorizando a facilidade de reprodução e a integração completa do sistema.

Tian *et al.* (2025) apresentaram um método de roteamento para veículos especiais na rede viária da China, usando um modelo de rede em duas camadas, um índice espacial em grade e uma versão paralela do Dijkstra bidirecional. Em um banco de dados nacional com cerca de 24 milhões de trechos de vias, eles mostraram que a busca bidirecional paralela é cerca de 3,5 vezes mais rápida que o Dijkstra comum. Em uma rota específica, a busca bidirecional foi 41,2% mais rápida que o A*, o que os autores explicam pela dificuldade do A* em calcular distâncias em redes muito complexas. Embora esse estudo seja parecido com o nosso por focar na escala nacional, existem diferenças importantes: a rede chinesa é muito maior que a brasileira usada aqui; o sistema deles foi feito sob medida em C++ em vez de usar o pgRouting; a busca bidirecional deles é feita sem heurística, enquanto nós avaliamos justamente o impacto da heurística na busca bidirecional (usando o `pgr_bdAstar`); e eles incluem regras específicas para veículos, como limite de peso em pontes e altura em túneis. O fato de Tian *et al.* (2025) mostrarem que a busca bidirecional supera o A* em escala nacional reforça a importância de investigar o comportamento da busca bidirecional com heurística nessa mesma escala, como é feito neste estudo.

3.4 Posicionamento em Relação aos Trabalhos Correlatos

O Quadro 3.1 sintetiza os trabalhos discutidos segundo os aspectos que caracterizam a presente proposta.

Quadro 3.1: Comparativo dos trabalhos correlatos segundo os aspectos que caracterizam a presente proposta.

Trabalho	Ferramenta / motor	Bidirecional com heurística	Escala	Sistema web	Docker
Chadha e Garg (2019)	pgRouting + QGIS	Não	Nacional (Índia)	Não	Não
Yusoff et al. (2014)	pgRouting (web-GIS)	Não	Regional	Sim	Não
Zhang e He (2012)	pgRouting	Não	Provincial	—	Não
Miler et al. (2014)	pgRouting; Neo4j	Não	Nacional (Áustria)	Não	Não
Rachmawati e Gustin (2020)	Implementação própria	Não	Urbana	—	Não
Rios e Chaimowicz (2011)	Impl. própria (PNBA*)	Sim (fora de rede viária)	— (quebra-cabeças)	Não	Não
Dibbelt et al. (2015)	Impl. própria (C++)	Não	Continental	Não	Não
Luxen e Vetter (2011)	OSRM (C++, CH)	Não	Continental	Não	Não
Tian et al. (2025)	Impl. própria (C++)	Não	Nacional (China)	Não	Não
Este trabalho	pgRouting	Sim (pgr_bdAstar)	Nacional (Brasil)	Sim	Sim

Fonte: Elaborado pelo autor.

Muitos estudos tratam de apenas alguns desses pontos, como o **pgRouting**, a busca bidirecional, a escala nacional, a interface web e os contêineres, mas nenhum combina todos eles juntos como este trabalho faz: **pgRouting**, busca bidirecional com heurística (**pgr_bdAstar**), escala nacional, sistema web completo e uso de contêineres. Tudo isso é combinado, com exceção da rede de ruas não direcionada (**directed := FALSE**), já que a direção da via não faz diferença se considerarmos apenas ruas de sentido único.

Desenvolvimento

Este capítulo apresenta o sistema desenvolvido para o cálculo e a visualização de rotas sobre a malha viária brasileira, detalhando as decisões de implementação que sustentam a comparação entre os três algoritmos de caminho mínimo propostos. A exposição parte da visão geral da arquitetura e das tecnologias empregadas, avança pela construção da base de dados geográfica a partir dos dados do OpenStreetMap e descreve, em seguida, os três componentes que constituem a aplicação: o serviço de banco de dados, responsável pelo armazenamento do grafo viário e pela execução das consultas de roteamento; o *back-end*, que expõe os algoritmos por meio de uma interface de programação; e o *front-end*, encarregado da interação com o usuário e da renderização das rotas sobre o mapa. Por fim, descreve-se a estratégia de containerização que torna a aplicação reproduzível de forma integral por meio de um único comando. O código-fonte do projeto está disponível no GitHub no repositório <https://github.com/coutinhosf/roteamento-geoespacial>.

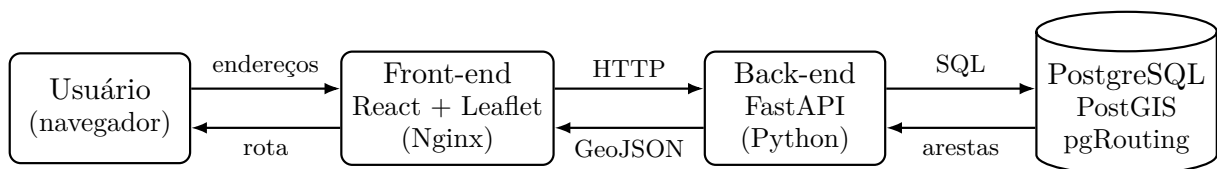
4.1 Visão Geral da Arquitetura

A arquitetura do sistema foi criada de maneira composta por três camadas, em que cada uma é isolada dentro de seu próprio contêiner e gerenciada através do Docker Compose, conforme ilustra a Figura 4.1. Para a primeira camada tem-se o serviço do banco de dados, que utiliza como base para sua construção o PostgreSQL estendido pelo PostGIS e pelo pgRouting, onde está o grafo da malha viária do país e onde são implementadas as buscas por rotas de caminho mínimo. Já na segunda camada encontra-se o *back-end*, que é implementado em Python com o arcabouço FastAPI, deixando disponíveis os algoritmos de busca de rotas através de uma *API* REST e realizando um intermédio com o acesso ao banco de dados. Na última camada é encontrado o *front-end*, que se desenvolve em React servido por um servidor Nginx, e é onde há a responsabilidade pela interface com o usuário, além de realizar a visualização cartográfica do roteamento utilizando-se a biblioteca Leaflet.

O fluxo natural de uma solicitação acaba por percorrer as três camadas em sequência.

Primeiramente, na interface, o usuário coloca como dados de entrada os endereços de origem e destino e o algoritmo de sua escolha. No *front-end* ocorre então o encaminhamento das informações até o *back-end*, onde serão transformadas em novas informações, sendo os endereços interpretados nesta camada como coordenadas geográficas, que encontram o vértice do grafo mais próximo e então enviam para o banco de dados a busca pgRouting referente ao algoritmo selecionado. O resultado é então enviado pelo caminho contrário para o *back-end* em formato GeoJSON, voltando até o *front-end*, onde é convertido em uma polilinha desenhada sobre o mapa, contendo as informações de distância total e tempo de processamento do algoritmo. A separação entre as camadas independentes contribui com a manutenção, possibilita a substituição isolada dos componentes e ainda facilita a reprodução do ambiente descrito na Seção 4.6.

Figura 4.1 – Arquitetura em três camadas do sistema e fluxo de uma requisição de rota.



Fonte: Elaborado pelo autor.

4.2 Tecnologias Utilizadas

As ferramentas tecnológicas utilizadas para criar esse sistema foram selecionadas com embasamento em sua maturidade, com características de código aberto e aderência ao propósito de manipulação de dados geográficos. O Quadro 4.1 traz a relação entre cada uma das tecnologias com sua versão correspondente e a finalidade que desenvolve no projeto.

4.3 Base de Dados

4.3.1 Obtenção dos Dados do Brasil

As informações geográficas responsáveis pela composição da base do sistema foram coletadas do projeto OpenStreetMap através de um repositório gerenciado pela Geofabrik, que atua fornecendo extratos regionais atualizados periodicamente. O uso desse extrato corresponde ao território completo do Brasil, em formato binário `.osm.pbf` (*Protocolbuffer Binary Format*), que foi selecionado por se tratar de um modelo mais compacto e com leitura mais eficaz que o formato XML, que seria o original utilizado no OpenStreetMap, fazendo com que não fosse preciso realizar mais conversões em formato de importação. O arquivo `brazil.osm.pbf`, com aproximadamente 1,9 GB, obtido em 6 de agosto de 2026

Quadro 4.1: Tecnologias empregadas no desenvolvimento do sistema.

Tecnologia	Versão	Finalidade no projeto
PostgreSQL	16	Sistema gerenciador de banco de dados que hospeda o grafo viário e executa as consultas de roteamento.
PostGIS	3.4	Extensão espacial do PostgreSQL, responsável pelos tipos geométricos e pelas operações geográficas.
pgRouting	3.8	Extensão de roteamento sobre o grafo; fornece as funções <code>pgr_dijkstra</code> , <code>pgr_aStar</code> e <code>pgr_bdAstar</code> .
hstore	—	Extensão para armazenamento das etiquetas (<i>tags</i>) do OpenStreetMap em pares chave-valor.
osm2pgsql	1.4.1	Importação dos dados do OpenStreetMap para o banco de dados.
Python	3.11	Linguagem de programação do <i>back-end</i> .
FastAPI	0.83	Arcabouço web para a construção da <i>API REST</i> .
React	18.3	Biblioteca de construção da interface do <i>front-end</i> .
Leaflet	1.9.4	Biblioteca de mapas interativos para a renderização das rotas.
Nginx	1.31	Servidor do <i>front-end</i> e <i>proxy</i> reverso para o <i>back-end</i> .
Docker / Compose	29.2 / 5.0	Containerização e orquestração dos serviços da aplicação.

Fonte: Elaborado pelo autor.

(ver Seção 4.3.2 para a data do extrato e a referência), foi por fim transportado para o contêiner do banco de dados, onde ocorreu a etapa de importação a ser descrita a seguir.

4.3.2 Importação com osm2pgsql

A importação dos dados do OpenStreetMap para o banco de dados foi realizada com a ferramenta `osm2pgsql`. Cogitou-se, inicialmente, o uso do `osm2pgrouting`, que gera diretamente as tabelas topológicas exigidas pelo `pgRouting`; essa alternativa, contudo, processa todo o conjunto de dados em memória, o que a torna impraticável para um extrato de dimensão nacional. O `osm2pgsql`, por sua vez, opera em modo *slim*, apoiando-se em tabelas intermediárias persistidas no próprio banco, e por isso viabiliza a importação de arquivos de grande porte como o do Brasil.

A opção por uma ou outra ferramenta configura, assim, um compromisso entre conveniência e escalabilidade: o `osm2pgrouting` favorece a conveniência, ao entregar um grafo imediatamente roteável, mas não escala para grandes volumes de dados; o `osm2pgsql` favorece a escalabilidade, ao custo de exigir a construção posterior da topologia. Diante de um objeto de estudo em escala nacional, a conveniência do `osm2pgrouting` não se mostrou aplicável, e o passo adicional de construção do grafo constituiu a condição necessária à importação dos dados.

A impossibilidade foi observada de modo empírico durante as etapas de desenvolvimento. Tendo em consideração que o `osm2pgrouting` realiza o processamento de arquivos que se encontram na extensão OSM XML, foi preciso que o extrato em PBF fosse previamente transformado por meio do `osmium`, que é um utilitário de linha de comando que tem como serventia o tratamento e a transformação de extensões dos arquivos do

OpenStreetMap ([Topf, 2026](#)), obtendo como resultado um arquivo de aproximadamente 50 GB. Apesar de o limite de memória ter sido aumentado para 80 GB, o processamento acabou por ser interrompido automaticamente pelo sistema operacional enquanto realizava a leitura das informações, e a outra opção, distribuir a importação dos dados em partes menores por unidades federativas, acabou aumentando os custos da operação. Para essa etapa, os códigos utilizados são apresentados no Apêndice B.

As informações foram obtidas no extrato nacional público da Geofabrik, através do arquivo `brazil-latest.osm.pbf`, que possui em torno de 1,9 GB ([Geofabrik GmbH, 2026](#)). O arquivo foi salvo no dia 6 de agosto de 2026, sendo que o seu extrato possui os dados consolidados até 5 de agosto de 2026, ou seja, uma diferença mínima entre a coleta e a execução. Como o arquivo `latest` permanece sendo atualizado dia após dia, esse registro recente é ideal para a reprodução, sendo que o mesmo recorte selecionado se encontra passível de busca e armazenagem dentro do diretório de arquivos datados do servidor, com a nomeação de `brazil-260805.osm.pbf`.

A importação foi executada no interior do contêiner do banco de dados, tomando como entrada o arquivo `brazil.osm.pbf` previamente transferido. O comando empregado assumiu, em linhas gerais, a seguinte forma:

```
osm2pgsql --create --slim --hstore \  
          --cache 24000 --number-processes 8 \  
          --database mydatabase --username postgres \  
          brazil.osm.pbf
```

No final do processo, foram criadas tabelas com geometria `planet_osm_point`, `planet_osm_line`, `planet_osm_polygon` e `planet_osm_roads`, junto às tabelas intermediárias `planet_osm_nodes`, `planet_osm_ways` e `planet_osm_rels`, próprias do modo *slim*.

A inserção total dos dados foi finalizada em um tempo de 10 minutos e 5 segundos, utilizando um cache de nós de 24 GB em memória. Com esse tamanho é importante observar que, em relação à escala nacional, o cache abrange todos os 331 milhões de nós do extrato, mantendo sua taxa de acerto em 100%, fazendo com que sejam dispensadas as releituras em disco. A base concluída acabou por utilizar um espaço de quase 59 GB em disco, e a tabela `planet_osm_roads` armazenou 673.282 registros de segmentos de malha viária, que seriam utilizados como base para a construção do grafo.

4.3.3 Estrutura das Tabelas

Entre as tabelas que foram produzidas durante a importação, ressalta-se a importância para a busca da `planet_osm_roads`, que atua unindo os segmentos viários com maior hierarquia, ou seja, as vias arteriais e rodovias da malha, que foram extraídas do banco de dados. É justamente sobre essa tabela que é formada a topologia de rede executada pelo pgRouting. As outras tabelas `planet_osm_point` e `planet_osm_polygon`, ficam então armazenando, de maneira correspondente, as feições pontuais e poligonais que não

se encontram no cálculo das rotas, sendo que as tabelas intermediárias armazenam a representação bruta dos elementos do OpenStreetMap.

Ainda a tabela `planet_osm_roads` possui, dentre outras, as colunas que gerenciam a identificação de cada via, além de permitir a descrição de sua geometria, assim como o identificador originado no OpenStreetMap, que detalha a classe da via, nome, indicador de sentido único e a própria geometria. Quando analisadas as quase 70 colunas criadas, grande parte delas está ligada de maneira direta às *tags* do OpenStreetMap (como `surface`, `maxspeed` e `bridge`). Para o roteamento, apenas o subconjunto observado no Quadro 4.2 é relevante, acompanhado de um valor de exemplo retirado de um trecho existente na Avenida João Naves de Ávila, em Uberlândia.

Quadro 4.2: Principais colunas da tabela `planet_osm_roads` utilizadas no roteamento

Coluna	Tipo	Descrição	Exemplo
<code>osm_id</code>	<code>bigint</code>	Identificador único do elemento no OpenStreetMap.	1171661478
<code>highway</code>	<code>text</code>	Classificação da via (<i>motorway</i> , <i>residential</i> etc.); empregada para filtrar os segmentos roteáveis.	<code>primary</code>
<code>oneway</code>	<code>text</code>	Indicação de sentido único da via; não modelada neste trabalho, por adotar-se grafo não direcionado.	<code>yes</code>
<code>name</code>	<code>text</code>	Nome da via, utilizado na exibição da rota.	Avenida João Naves de Ávila
<code>z_order</code>	<code>integer</code>	Valor de ordenação derivado da hierarquia viária.	37
<code>way</code>	<code>geometry</code>	Geometria do tipo <i>LineString</i> na projeção Web Mercator (SRID 3857); base para o cálculo do custo das arestas.	<code>LINESTRING(...)</code> ¹

¹ Geometria convertida para SRID 4326 e exibida parcialmente: `LINESTRING(-48.244869 -18.928955, -48.245592 -18.928817, -48.246259 -18.928688, ...)`, num total de 31 vértices.

Fonte: Elaborado pelo autor.

A geometria de cada segmento permanece na coluna `way`, sobre a qual foram criados índices espaciais, sendo eles o GiST e o B-tree para o `osm_id`, com função de acelerar as consultas de proximidade e de roteamento. sobre a malha viária nacional. Na fase seguinte, com a topologia construída, conforme descrito na Seção 4.3.4, ainda é inserido na tabela atual as colunas `source` e `target`, que relacionam cada um dos segmentos aos vértices de sua delimitação. Também como produto é criada a tabela de vértices `planet_osm_roads_vertices_pgr`, que possui como principais colunas as representadas no Quadro 4.3.

Resulta-se um grafo que engloba 673.282 arestas e 671.093 vértices, sobre o qual são executadas as funções de roteamento descritas na Seção 2.5.

4.3.4 Extensões PostGIS e pgRouting

As informações do banco de dados foram obtidas por meio das extensões necessárias para o armazenamento espacial e também para o roteamento. A extensão PostGIS consegue fornecer modelos geométricos e funções espaciais, já a extensão pgRouting adiciona

Quadro 4.3: Principais colunas da tabela `planet_osm_roads_vertices_pgr`

Coluna	Tipo	Descrição	Exemplo
<code>id</code>	<code>bigint</code>	Identificador único do vértice, referenciado pelas colunas <code>source</code> e <code>target</code> das arestas.	240024
<code>cnt</code>	<code>integer</code>	Número de arestas incidentes ao vértice (grau), preenchido pela função <code>pgr_analyzeGraph</code> .	3
<code>ein</code>	<code>integer</code>	Arestas de entrada; não preenchida, por adotar-se grafo não direcionado.	—
<code>eout</code>	<code>integer</code>	Arestas de saída; não preenchida, por adotar-se grafo não direcionado.	—
<code>the_geom</code>	<code>geometry</code>	Geometria do tipo <i>Point</i> na projeção Web Mercator (SRID 3857), localizando o vértice na rede.	<code>POINT(-46.5627 -23.5615)</code>

Fonte: Elaborado pelo autor.

algoritmos de caminho mínimo e a extensão `hstore` consegue guardar as etiquetas (*tags*) do OpenStreetMap em formato de pares chave-valor. Assim, as extensões foram habilitadas para ser executadas com os seguintes comandos:

```
CREATE EXTENSION IF NOT EXISTS postgis;
CREATE EXTENSION IF NOT EXISTS pgrouting;
CREATE EXTENSION IF NOT EXISTS hstore;
```

Já a fase em que realmente há conversão de dados geográficos em um grafo que pode ser roteado é a criação da topologia de rede, ou seja, a atribuição dada a cada segmento pelos identificadores de vértice de origem e destino. Essa execução foi realizada através da função `pgr_createTopology`, aplicada à tabela `planet_osm_roads`. Conforme a versão que foi escolhida do `pgRouting`, não havia a possibilidade de criação automática das colunas de conectividade, então houve a necessidade de inserir no início do desenvolvimento as mesmas à tabela de arestas do `pgRouting`, o que permitiu que a topologia construída tivesse uma tolerância de junção de 1,0 metro, na projeção Web Mercator (SRID 3857):

```
ALTER TABLE planet_osm_roads
  ADD COLUMN source integer,
  ADD COLUMN target integer;

SELECT pgr_createTopology('planet_osm_roads', 1.0, 'way', 'osm_id');
```

Como resultados obtidos, foram construídas colunas `source` e `target` na tabela de vias e também foi formada a tabela de vértices `planet_osm_roads_vertices_pgr`, que, atuando em conjunto, podem descrever o grafo onde os algoritmos atuam.. Ainda sobre a base do território nacional, o tempo de processo foi finalizado em 2 minutos e 36 segundos, criando um grafo com 673.282 arestas e 671.093 vértices, onde não foi observada nenhuma aresta sem conectividade.

Finalizada a criação, a qualidade da topologia gerada da rede foi analisada com o uso da função `pgr_analyzeGraph`, que completa as colunas `cnt`, `ein` e `eout` da tabela de vértices e cria um diagnóstico da conectividade encontrada na malha. Como resultados após

observação e conclusão dos dados, pode-se verificar a Seção 5.4, que contém o contexto da avaliação de escalabilidade.

```
SELECT pgr_analyzeGraph('planet_osm_roads', 1.0, 'way', 'osm_id');
```

Os resultados que podem ser visualizados no Quadro 4.4, mostram que há características comuns de malha viária colaborativa no resultado, como por exemplo a observação de segmentos que não possuem saída geralmente são formados por vias que realmente não possuem continuidade, por outro lado os segmentos isolados e vãos potenciais são gerado por causa de imprecisões de digitalização e da falta de preenchimento das informações de cobertura do OpenStreetMap, que cria lacunas de dados, conforme verificado na Seção 2.3. Esses fatores acabam por gerar restrições de roteamento, sendo que os trajetos dependem de segmentos isolados nem sempre poderão ser encontrados na busca.

Quadro 4.4: Diagnóstico topológico da rede viária, obtido com `pgr_analyzeGraph`

Característica	Quantidade
Segmentos isolados	54.373
Segmentos sem saída (<i>dead ends</i>)	194.493
Vãos potenciais próximos a extremidades	160.947
Interseções detectadas	652.836
Geometrias em anel	38.969

Fonte: Elaborado pelo autor.

4.4 Back-end com FastAPI

A camada formada para atendimento dos serviços do sistema foi desenvolvida em Python utilizando o *framework* FastAPI, que gerencia as chamadas de requisições da interface, aciona as buscas de roteamento no banco de dados e oferta como produto final as rotas calculadas em formato GeoJSON. As subseções deste capítulo irão detalhar a organização do projeto, os *endpoints* que foram disponibilizados e as consultas de busca de rotas com o tratamento das coordenadas informadas pelos usuários.

4.4.1 Estrutura do Projeto

A escrita do código do *back-end* foi organizada internamente em blocos com responsabilidades distintas: a definição das rotas HTTP, as funções que estruturam e executam as buscas de roteamento e as funções auxiliares de acesso ao banco de dados e de tratamento das geometrias. Por causa dessa separação é possível alterar a lógica da busca sem que afete negativamente a camada HTTP, tornando mais fácil a manutenção do projeto. A aplicação é composta pelo módulo principal `main.py`, que concentra a lógica de serviço, e pelo arquivo `requirements.txt`, que declara as dependências. Os principais blocos do módulo são apresentados a seguir:

- ❑ **Helpers** — funções de apoio para formatação de distâncias, cálculo de direções e montagem das instruções de navegação;
- ❑ **SQL** — construção dinâmica das consultas de roteamento e busca do vértice mais próximo;
- ❑ **Endpoints** — definição das rotas HTTP expostas pela *API*.

4.4.2 Endpoints Implementados

A *API* é responsável por expor *endpoints* REST para que seja possível calcular as rotas e também atua como apoio da interface. O *endpoint* principal gerencia as coordenadas de origem e destino após recebê-las, executa a busca solicitada no pgRouting e devolve o traçado obtido da rota em GeoJSON, junto com os dados de distância total e tempo de processamento. Os *endpoints* implementados, com seu método HTTP, rota, descrição e parâmetros, são relacionados no Quadro 4.5.

Quadro 4.5: Endpoints da *API* REST do *back-end*

Método	Rota	Descrição	Parâmetros
POST	/rota	Calcula a rota de caminho mínimo entre origem e destino com o algoritmo selecionado.	origem_lat, origem_lng, destino_lat, destino_lng, algoritmo
GET	/geocode	Converte um endereço textual em coordenadas geográficas, via Nominatim.	q (endereço)
POST	/cidade/preparar	Inicia a construção da malha urbana detalhada para a cidade informada.	nome (cidade)
GET	/cidade/status	Consulta o progresso da preparação de uma cidade.	slug (cidade)
GET	/cidades	Lista as cidades já preparadas.	—
GET	/health	Verifica a disponibilidade da <i>API</i> .	—

Fonte: Elaborado pelo autor.

4.4.3 Consultas pgRouting

Para calcular as rotas foram utilizadas as funções `pgr_dijkstra`, `pgr_aStar` e `pgr_bdAstar`, aplicadas sobre uma consulta interna de arestas composta pelas colunas `id`, `source`, `target` e `cost`, na qual o custo corresponde ao comprimento do trecho, calculado pela função `ST_Length` (PGROUTING, 2026). As funções `pgr_aStar` e `pgr_bdAstar` dependem de uma função heurística e, por isso, exigem que a tabela de arestas inclua as coordenadas das extremidades de cada aresta (`x1`, `y1`, `x2`, `y2`), utilizadas para estimar, por meio da distância euclidiana, o custo restante até o destino (PGROUTING, 2026). A função

`pgr_dijkstra`, que realiza uma busca uniforme, não requer tais coordenadas (PGROUTING, 2026).

Como as três funções compartilham a mesma base de arestas, a única variação entre as execuções é a estratégia de busca, o que garante que a comparação entre os algoritmos ocorra em condições idênticas. Para todas as consultas foi utilizado o parâmetro `directed := false`, que trata o grafo como não direcionado, conforme a limitação assumida no Capítulo 3, segundo a qual as restrições de sentido único das vias não são modeladas. Em todas elas, o trecho `SELECT ...` externo, omitido por concisão, une os segmentos retornados pela função de roteamento à tabela de arestas, agrega as geometrias em um único traçado com `ST_Union` e o converte para GeoJSON com `ST_AsGeoJSON`.

O código-fonte apresentado na Listagem 4.1 demonstra a consulta que utiliza a função `pgr_dijkstra`, na qual o algoritmo é aplicado sobre a base de arestas com as colunas mínimas requisitadas pelo pgRouting.

Código-fonte 4.1 – Consulta de roteamento com `pgr_dijkstra`

```
SELECT ... FROM planet_osm_roads r
JOIN pgr_dijkstra(
  'SELECT osm_id AS id, source, target,
        ST_Length(way) AS cost
    FROM planet_osm_roads',
  origem_id, destino_id, false
) d ON r.osm_id = d.edge;
```

Fonte: Elaborado pelo autor.

O código-fonte apresentado na Listagem 4.2 demonstra uma busca que utiliza a função `pgr_aStar`, na qual são adicionadas à base de arestas as coordenadas das extremidades, essenciais para o cálculo da heurística que orienta a busca em direção ao destino.

Código-fonte 4.2 – Consulta de roteamento com `pgr_aStar`

```
SELECT ... FROM planet_osm_roads r
JOIN pgr_aStar(
  'SELECT osm_id AS id, source, target,
        ST_Length(way) AS cost,
        ST_X(ST_StartPoint(way)) AS x1,
        ST_Y(ST_StartPoint(way)) AS y1,
        ST_X(ST_EndPoint(way)) AS x2,
        ST_Y(ST_EndPoint(way)) AS y2
    FROM planet_osm_roads',
  origem_id, destino_id, directed := false
) d ON r.osm_id = d.edge;
```

Fonte: Elaborado pelo autor.

Por fim, o código-fonte apresentado na Listagem 4.3 demonstra uma busca que utiliza a função `pgr_bdAstar`, na qual se emprega a mesma base de arestas do A*. Trata-se da

versão bidirecional do algoritmo, que realiza buscas heurísticas simultâneas a partir da origem e do destino.

Código-fonte 4.3 – Consulta de roteamento com `pgr_bdAstar`

```
SELECT ... FROM planet_osm_roads r
JOIN pgr_bdAstar(
  'SELECT osm_id AS id, source, target,
        ST_Length(way) AS cost,
        ST_X(ST_StartPoint(way)) AS x1,
        ST_Y(ST_StartPoint(way)) AS y1,
        ST_X(ST_EndPoint(way)) AS x2,
        ST_Y(ST_EndPoint(way)) AS y2
  FROM planet_osm_roads',
  origem_id, destino_id, directed := false
) d ON r.osm_id = d.edge;
```

Fonte: Elaborado pelo autor.

Em cada uma delas, o trecho descrito em `SELECT ...` externo, que é omitido por concisão, consegue unir na tabela de arestas aqueles segmentos que foram recebidos como produto final após a função de roteamento. Conforme o caminho é formado por diferentes segmentos, todos independentes, como por exemplo caminhos que geram 29 segmentos, a função `ST_Union` acaba por ligar todas essas geometrias em uma geometria única do tipo *MultiLineString*, que é composta por todas as partes dessa rota, porém apresentadas em uma só estrutura. Essa geometria é posteriormente transformada para o formato GeoJSON com `ST_AsGeoJSON` e enviada à interface, na qual ocorre a renderização do traçado completo da rota (Seção 2.7.3).

Conforme as funções de roteamento criam a rota fragmentada em diversos segmentos, em que cada um deles corresponde a uma aresta do trajeto, a função `ST_Union` relaciona essas geometrias em um único objeto do tipo *MultiLineString*, que reúne todas as partes da rota em uma só estrutura, que ao final do processo será convertida para o formato GeoJSON com `ST_AsGeoJSON`. Por fim, o Código-fonte 4.4 ilustra esse comportamento: três segmentos individuais da rota (à esquerda) são agregados em uma única geometria composta (à direita).

Código-fonte 4.4 – Efeito da função `ST_Union` sobre os segmentos de uma rota

```
-- Antes: tres segmentos separados
LINESTRING(-46.563318 -23.561524, -46.563160 -23.561599)
LINESTRING(-46.563160 -23.561599, -46.563002 -23.561676)
LINESTRING(-46.563002 -23.561676, ..., -46.562731 -23.561524)

-- Depois: um unico MultiLineString
MULTILINESTRING(
  (-46.563318 -23.561524, -46.563160 -23.561599),
  (-46.563160 -23.561599, -46.563002 -23.561676),
  (-46.563002 -23.561676, ..., -46.562731 -23.561524)
)
```

Fonte: Elaborado pelo autor.

4.4.4 Geocodificação e Busca do Nó Mais Próximo

As localizações de origem e destino são posições informadas pelo usuário como dados de entrada e precisam ser relacionadas a vértices do grafo antes que haja a execução da busca. Para tal ação, as informações recebidas pela API são relacionadas ao vértice mais próximo da rede através de uma consulta espacial que se baseou na função `ST_Distance`, com apoio do operador de índice `<->` do PostGIS, que possibilita a recuperação do vizinho mais próximo sem precisar da varredura completa da tabela dos vértices. Nos casos em que a entrada ocorre por formato de texto, a conversão prévia para coordenadas geográficas é feita por meio de um trabalho de geocodificação, que é feito por meio do serviço Nominatim (Seção 2.7.5), no qual é acrescentado por meio do endpoint `/geocode` da API, que trabalha encaminhando o termo utilizado na procura até a interface pública do Nominatim, o que acaba por limitar os resultados para dentro do território brasileiro. A requisição também assume a forma a seguir:

Código-fonte 4.5 – Requisição ao serviço Nominatim.

```
https://nominatim.openstreetmap.org/search
?q=Universidade Federal de Uberlandia
&format=json&limit=5&countrycodes=br
```

Como resposta, o trabalho realizado gerou como produto final um documento JSON composto por uma lista de resultados, no qual cada um contém itens como as coordenadas (`lat`, `lon`) e o endereço completo (`display_name`). Já o Código-fonte 4.6 demonstra, sucintamente, um resultado que é retornado para a busca por meio do termo referente à Universidade Federal de Uberlândia.

Código-fonte 4.6 – Resposta resumida do Nominatim para uma consulta de geocodificação

```
{
  "osm_type": "way",
  "osm_id": 804592441,
  "lat": "-18.9185163",
  "lon": "-48.2576139",
  "class": "amenity",
  "type": "university",
  "name": "Universidade Federal de Uberlandia",
  "display_name": "Universidade Federal de Uberlandia,
    Rua Isaac de Oliveira, Santa Monica,
    Uberlandia, Minas Gerais, 38408-102, Brasil"
}
```

Fonte: Elaborado pelo autor, a partir de retorno do Nominatim.

Ainda sobre as informações geradas, o sistema utiliza as coordenadas `lat` e `lon` obtidas no primeiro resultado para servir de referência como ponto de origem ou destino. Em seguida, cada uma das coordenadas é associada ao seu vértice mais próximo no grafo por meio de outra busca, conforme a descrição do Código-fonte 4.7, no qual o operador `<->` organiza os vértices pela distância até a coordenada informada. A procura de rota é

limitada ao maior componente conectado do grafo, de modo que segmentos isolados não possam ser escolhidos como origem ou destino.

Código-fonte 4.7 – Consulta de localização do vértice mais próximo

```
SELECT v.id
FROM planet_osm_roads_vertices_pgr v
JOIN (
    SELECT node FROM pgr_connectedComponents(
        'SELECT osm_id AS id, source, target,
            ST_Length(way) AS cost
        FROM planet_osm_roads'
    ) WHERE component = 7
) c ON v.id = c.node
ORDER BY v.the_geom <-> ST_Transform(
    ST_SetSRID(ST_MakePoint(:lng, :lat), 4326), 3857
)
LIMIT 1;
```

Fonte: Elaborado pelo autor.

4.5 Front-end com React e Leaflet

4.5.1 Estrutura de Componentes

A interface é dividida em componentes específicos com funções características. Mesmo que se encontrem unidos dentro de um único arquivo, tais componentes possuem responsabilidades distintas uns dos outros, onde o principal realiza as tarefas de gerenciamento do estado de implementação e organiza a ordenação da barra lateral e do mapa. Já o outro componente de entrada de informações, se torna responsável por realizar a procura do endereço solicitado utilizando o texto de entrada. Outro componente é o de instrução, o qual é representado pela divisão de etapas do trabalho, e por fim, tem-se um componente auxiliar, que cuida dos ajustes de enquadramento do mapa gerado. Cada um desses componentes organiza o próprio estado e mantém a comunicação com os demais através de propriedades e estados compartilhados que possuem na aplicação. Os componentes implementados são relacionados no Quadro 4.6.

Quadro 4.6: Componentes React implementados na interface

Componente	Responsabilidade
App	Componente principal; gerencia o estado global e monta a barra lateral e o mapa.
AddressInput	Campo de busca de endereço com sugestões automáticas, obtidas por geocodificação.
InstructionItem	Item individual da lista de instruções de navegação da rota.
FitBounds	Ajusta automaticamente o enquadramento do mapa à extensão da rota calculada.

Fonte: Elaborado pelo autor.

4.5.2 Formulário de Entrada

O formulário de entrada é responsável por concentrar os parâmetros necessários para calcular a rota, sendo eles a origem, o destino e o algoritmo de busca a ser executado. Os campos de inserção de origem e destino contam com sugestões automáticas: à medida que o usuário digita, a interface consulta o serviço de geocodificação e gera uma lista de endereços correspondentes ao termo solicitado, dentre os quais um é escolhido para fixar as coordenadas. O algoritmo é selecionado entre as três opções disponíveis: Dijkstra, A* e A* bidirecional. Quando o formulário é preenchido e submetido, a requisição é enviada ao *back-end* e a interface aguarda o retorno da rota calculada. A Figura 4.2 apresenta o formulário implementado.

A imagem mostra a interface de usuário de um aplicativo de roteamento. No topo, há o título 'Roteamento' em verde, com o subtítulo 'pgRouting + OpenStreetMap Brasil' abaixo dele. Abaixo do título, há uma barra de navegação com três abas: 'Rota' (destacada em verde), 'Instruções' e 'Cidades'. O formulário principal contém os seguintes elementos: 1. Um campo de entrada rotulado 'ORIGEM' com um ícone de localização e o placeholder 'Digite origem...'. 2. Um campo de entrada rotulado 'DESTINO' com um ícone de localização e o placeholder 'Digite destino...'. 3. Uma seção 'MODO' com dois botões: 'Único' (destacado em verde) e 'Comparar'. 4. Uma seção 'ALGORITMO' com três botões: 'Dijkstra' (destacado em verde), 'A*' e 'Bi-A*'. 5. Um texto descritivo 'Clássico, sempre ótimo' abaixo dos botões de algoritmo. 6. Um botão grande e verde no rodapé rotulado '→ Calcular Rota'.

Figura 4.2 – Formulário de entrada para o cálculo de rota.
Fonte: Elaborado pelo autor.

4.5.3 Renderização da Rota com Leaflet

O GeoJSON retornado pelo *back-end* é interpretado pelo Leaflet e renderizado sobre as camadas de *tiles* do OpenStreetMap. De modo diverso à construção manual de uma polilinha, a renderização é criada de modo que o componente `GeoJSON` da biblioteca *react-leaflet*, recebe diretamente o objeto GeoJSON, e o mesmo o desenha sobre o mapa. Como a geometria já se encontra no sistema WGS 84 (EPSG:4326) quando chega em resposta, conforme descrito no Capítulo 2, nenhuma outra transformação é necessária na

interface, e o traçado é apresentado de maneira direta no mapa, com ajuste automático do enquadramento à extensão da rota. O trecho responsável pela renderização é apresentado no Código-fonte 4.8.

Código-fonte 4.8 – Renderização da rota em GeoJSON com *react-leaflet*

```
<MapContainer center={[-15.7, -47.9]} zoom={5}>
  <TileLayer
    url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
    attribution="(c) OpenStreetMap" />

  {resultado?.geojson && (
    <>
      <GeoJSON data={resultado.geojson}
        style={{ color: cor, weight: 4, opacity: 0.85 }} />
      <FitBounds geojson={resultado.geojson} />
    </>
  )}
</MapContainer>
```

Fonte: Elaborado pelo autor.

4.5.4 Exibição dos Resultados

Ainda além da visualização do traço apresentada sobre o mapa, a interface mostra também ao usuário os dados que são obtidos no cálculo, como a distância total do percurso e o tempo de processamento do algoritmo selecionado, o que permite a comparação entre as execuções. Para identificar melhor a exibição, a Figura 4.3 apresenta o painel de resultados.

OSM ROUTING

Roteamento

pgRouting + OpenStreetMap Brasil

Rota

Instruções (3)

Cidades (2)

ORIGEM

Universidade Federal de Uberlândia, Rua Vita

DESTINO

Biblioteca Setorial Glória - UFU, Rodovia Chic

MODO

Único

Comparar

REPETIÇÕES

5

Cada algoritmo roda 5x. A 1ª execução é descartada (cache frio); média ± desvio padrão sobre as 4 restantes.

ALGORITMO

Dijkstra

A*

Bi-A*

Os três algoritmos rodam em sequência, um de cada vez

⇌ Comparar (5x cada)

COMPARAÇÃO

5 execuções · 1ª descartada

◇ Malha nacional

Dijkstra DIST. 6,54 km TEMPO MÉDIO 1041 ms ± 17,8 n=4 INSTR. 7 MAIS RÁPIDO	A* DIST. 6,54 km TEMPO MÉDIO 1345 ms ± 3,3 n=4 INSTR. 7	Bi-A* DIST. 6,54 km TEMPO MÉDIO 1350 ms ± 6,5 n=4 INSTR. 7
---	---	--

! Salvar comparação (PNG)

Ver instruções →

Figura 4.3 – Interface do sistema exibindo o painel de resultados no modo de comparação

Fonte: Elaborado pelo autor.

4.6 Containerização com Docker

4.6.1 Estrutura do Docker Compose

O arquivo `docker-compose.yml` detalha, por meio de declarações, os quatro serviços da implementação — banco de dados, importador da base, *back-end* e *front-end* —, além de descrever a rede que os conecta e o volume de persistência dos dados. A integridade do banco de dados, verificada pelo *healthcheck* por meio de testes simples de conexão, consultas rápidas e verificação de tabelas corrompidas, garante a inicialização correta do serviço do banco de dados, para que os demais serviços entrem em execução somente após essas verificações. Através dessa definição, todo o conjunto pode ser iniciado por meio de um único comando, sem a necessidade de instalação manual de dependências. O arquivo utilizado é apresentado no Código-fonte 4.9.

Código-fonte 4.9 – Arquivo `docker-compose.yml`

```
services:
  db:
    build: ./db
    image: routing-db
    environment:
      POSTGRES_DB: mydatabase
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: "****"
    ports:
      - "5433:5432"
    volumes:
      - db_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres -d mydatabase"]
      interval: 5s
      timeout: 5s
      retries: 10
    networks:
      - routing-net
  importer:
    build: ./db
    image: routing-db
    environment:
      PGHOST: db
      PGPORT: "5432"
      PGDATABASE: mydatabase
      PGUSER: postgres
      PGPASSWORD: "****"
      OSM2PGSQL_CACHE: ${OSM2PGSQL_CACHE:-2048}
      PBF_URL: ${PBF_URL:-https://download.geofabrik.de/south-america/
        brazil-latest.osm.pbf}
```

```
volumes:
  - ./data:/data
  - ./db/import/importar_brasil.sh:/importar_brasil.sh:ro
entrypoint: ["bash", "/importar_brasil.sh"]
restart: "no"
depends_on:
  db:
    condition: service_healthy
networks:
  - routing-net
backend:
  build: ./backend
  environment:
    DB_HOST: db
    DB_PORT: "5432"
    DB_NAME: mydatabase
    DB_USER: postgres
    DB_PASSWORD: "***"
  depends_on:
    db:
      condition: service_healthy
    importer:
      condition: service_completed_successfully
  ports:
    - "8000:8000"
  networks:
    - routing-net
frontend:
  build: ./frontend
  depends_on:
    - backend
  ports:
    - "80:80"
  networks:
    - routing-net
volumes:
  db_data:
networks:
  routing-net:
    driver: bridge
```

Fonte: Elaborado pelo autor.

No arquivo, cada serviço cumpre um papel específico. O serviço `db` constrói a imagem do banco de dados a partir do diretório `./db` e persiste os dados no volume `db_data`, garantindo que a base importada não seja perdida entre execuções; sua diretiva `healthcheck` verifica periodicamente, com o utilitário `pg_isready`, se o banco está apto a receber conexões. O serviço `importer`, executado a partir da mesma imagem do banco, é responsável

pela preparação automática da base: na primeira execução, baixa o extrato do Brasil da Geofabrik para a pasta `./data` (caso o arquivo ainda não exista), realiza a importação com o `osm2pgsql`, constrói a topologia do grafo e registra o maior componente conectado da malha; nas execuções seguintes, detecta que a base já está pronta e encerra em segundos, por ser um processo idempotente. O serviço `backend` constrói a API a partir do diretório `./backend` e recebe, por variáveis de ambiente, os dados de conexão ao banco — note-se que o `host` passa a ser `db`, o nome do serviço na rede interna, e não `localhost`; sua cláusula `depends_on` assegura que a API só inicie após o banco estar pronto (`service_healthy`) e a importação concluída (`service_completed_successfully`). O serviço `frontend` constrói a interface a partir do diretório `./frontend` e depende do `back-end`. Os serviços comunicam-se pela rede `routing-net`, do tipo `bridge`, que os isola em um domínio próprio de resolução de nomes.

4.6.2 Contêiner do Banco de Dados

O trabalho do banco de dados se utiliza da imagem oficial `postgis/postgis`, que possibilita o uso do PostgreSQL com a extensão espacial PostGIS anteriormente instalada. A extensão `pgRouting` não vem adicionada por padrão à imagem; o comando para sua instalação está no arquivo `Dockerfile`, que utiliza o repositório PGDG e faz as cópias dos scripts de inicialização. A mesma imagem é reutilizada pelo serviço `importer`, motivo pelo qual o `Dockerfile` também instala o `osm2pgsql` e o `wget`, utilizados na importação e no download do extrato. Os dados persistem em volume, onde a base importada da malha viária nacional não precisa ser reconstruída a cada execução dos contêineres.

Código-fonte 4.10 – Dockerfile do serviço de banco de dados

```
FROM postgis/postgis:16-3.4
RUN apt-get update \
    && apt-get install -y --no-install-recommends \
        postgresql-16-pgrouting osm2pgsql wget ca-certificates \
    && rm -rf /var/lib/apt/lists/*
COPY init/ /docker-entrypoint-initdb.d/
```

Fonte: Elaborado pelo autor.

A instrução `FROM` usa a imagem `postgis/postgis:16-3.4` como base. A instrução `RUN` atualiza a lista de pacotes e instala o `postgresql-16-pgrouting`, o `osm2pgsql` e o `wget` e, logo em seguida, exclui os arquivos de cache para reduzir o tamanho da imagem. Por último, a instrução `COPY` copia os scripts do diretório `init` para `/docker-entrypoint-initdb.d`, diretório reconhecido pela imagem para execução automática na primeira inicialização do banco. Estes scripts habilitam as três extensões necessárias, conforme o Código-fonte 4.11.

Código-fonte 4.11 – Script de habilitação das extensões

```
CREATE EXTENSION IF NOT EXISTS postgis;
CREATE EXTENSION IF NOT EXISTS hstore;
```

```
CREATE EXTENSION IF NOT EXISTS pgrouting;
```

Fonte: Elaborado pelo autor.

4.6.3 Contêiner do Back-end

O contêiner da API é formado por meio de uma imagem Python, onde se instalam as dependências do projeto e onde é copiado o código utilizado para a aplicação, através do servidor FastAPI que é exposto na rede interna do Compose. O `Dockerfile` utilizado é apresentado no Código-fonte 4.12.

Código-fonte 4.12 – `Dockerfile` do serviço de *back-end*

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 8000
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Fonte: Elaborado pelo autor.

O comando `FROM` usa o `python:3.11-slim`, que é uma versão leve do Python. O `WORKDIR` define que o diretório de trabalho vai ser o `/app`. Depois disso, o `requirements.txt` é copiado para a pasta. Em seguida, as dependências são instaladas usando `pip --no-cache-dir` para aproveitar o sistema de cache de camadas do Docker para instalações só do código alterado para agilizar, ao invés de ter que reinstalar todos os pacotes de dependência de novo a cada atualização de código. O `EXPOSE` indica que a porta 8000 está exposta. No ultimo passo, o `CMD` vai rodar o `uvicorn` para iniciar o servidor `uvicorn` rodando o FastAPI escutando em todas as interfaces (compondo a rede interna do Compose).

4.6.4 Contêiner do Front-end

O contêiner pertencente à interface faz a compilação da aplicação React em uma etapa de *build* e fornece os arquivos estáticos resultantes através do servidor Nginx, que realiza o encaminhamento das solicitações feitas à API. Para isso, utiliza-se um processo de compilação em múltiplos estágios (*multi-stage build*), apresentada no Código-fonte 4.13.

Código-fonte 4.13 – `Dockerfile` do serviço de *front-end*

```
FROM node:20-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build
```

```
FROM nginx:alpine
COPY nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=build /app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Fonte: Elaborado pelo autor.

A primeira fase desse processo é construir a aplicação React a partir do código-fonte usando a imagem `node:20-alpine`, nesta etapa, todas as dependências são instaladas com o `npm ci` e depois compiladas em arquivos estáticos (HTML/CSS/JS) através do `npm run build`. Esses arquivos vão para a pasta `dist`.

A segunda fase copia apenas esses arquivos compilados para a nova imagem (imagem `nginx:alpine`). Com isso, o tamanho é reduzido e a imagem final fica bem mais leve, já que não há necessidade de manter a imagem salva no ambiente de compilação depois que o aplicativo foi compilado.

O Nginx é configurado para realizar duas tarefas principais: servir a sua *Single Page Application* (por padrão, ele entrega o `index.html` para URLs desconhecidas) e atuar como um *Proxy Reverso*, encaminhando todas as chamadas de API (`/api/`) para a porta 8000, onde fica o serviço de *backend*.

Código-fonte 4.14 – Configuração do Nginx (`nginx.conf`)

```
server {
    listen 80;
    server_name _;
    root /usr/share/nginx/html;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    location /api/ {
        proxy_pass http://backend:8000/;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Fonte: Elaborado pelo autor.

4.6.5 Execução da Aplicação

A aplicação é iniciada por meio de um único comando, executado na raiz do projeto:

```
docker compose up --build
```

Esse comando constrói as imagens dos serviços — banco de dados, importador, *back-end* e *front-end* — e os coloca em execução na ordem definida pelas dependências descritas na Seção 4.6.1. Na primeira execução, o banco inicia vazio e o serviço **importer** realiza automaticamente o download do extrato, a importação dos dados e a construção da topologia, conforme detalhado no Apêndice A; o *back-end* só entra em execução após a conclusão dessas etapas. Nas execuções seguintes, como os dados são persistidos em volume, o importador detecta que a base já está disponível e encerra em segundos, e a interface torna-se acessível no navegador.

O sistema foi validado nos sistemas operacionais Linux (Ubuntu 26.04 LTS) e Windows 11, nos quais a aplicação foi executada a partir do mesmo comando, sem alterações na configuração dos contêineres.

4.7 Malha Urbana Detalhada sob Demanda

Os dados importados usando a base nacional como entrada no programa **osm2pgsql** incluem apenas as vias principais contidas na tabela **planet_osm_roads**. Funciona bem para viagens rurais ou de longa distância, mas é insuficiente para trajetos urbanos, já que vias locais e residenciais são excluídas da tabela. Por isso, ao criar rotas urbanas baseadas na malha nacional, elas podem seguir exclusivamente por rodovias e criar caminhos mais longos do que os que existiriam na vida real.

Assim, o sistema desenvolveu uma opção para gerar um mapa bem detalhado de uma cidade escolhida a cada pedido de rota. Esse novo mapa detalhado é gerado por um processo de 4 etapas. Primeiro, o *bounding box* associado à cidade selecionada pelo viajante é pego do Nominatim (veja a Seção 2.7.5). Segundo, as linhas que ficam dentro do *bounding box* buscado acima são então tiradas da tabela **planet_osm_line**. As linhas são salvas nessa tabela como todos os tipos de vias. Terceiro, essas linhas podem ser divididas em vários segmentos, dependendo de quantas formas existem de dividir a linha. O número total de segmentos criados é maior do que o número original de linhas. Esses segmentos agora podem ser processados no formato **pgr_nodeNetwork** para usar o **pgr_createTopology**. Por exemplo, o processamento da cidade de Monte Alegre de Minas resultou em 7.523 vias convertidas em 17.479 segmentos. O maior componente conectado representou cerca de 99,38% de toda a rede. Assim, quase a rede inteira ficou conectada. O processo também cria um registro na tabela de controle para que a cidade possa ser consultada de novo sem precisar passar pelo processo de preparação outra vez.

Na hora de escolher uma rota, o sistema de roteamento vai determinar automaticamente se deve usar a malha detalhada da cidade preparada anteriormente ou a malha detalhada nacional. Se os pontos de início e fim da rota estiverem dentro do *bounding box* de uma cidade já preparada, essa malha detalhada da cidade será usada. Caso contrário,

a malha detalhada nacional será utilizada. Essa escolha é direta e não exige nenhuma interação do usuário.

Testes e Resultados

Neste capítulo serão apresentados os testes que foram realizados no sistema desenvolvido e a análise dos resultados obtidos. A avaliação foi dividida em três frentes: a verificação da correção das rotas calculadas, feita por meio de comparação visual entre os algoritmos; a medição do desempenho de cada uma das três funções de roteamento sobre a malha viária nacional; e, por último, a análise da escalabilidade da solução considerando o volume de dados do Brasil. Ao final, os resultados foram confrontados com o sistema desenvolvido em (Oliveira, 2021), que constitui o trabalho estendido por esta monografia, e discutidos à luz das limitações assumidas.

5.1 Metodologia dos Testes

Os testes foram executados em uma máquina com processador Intel Core Ultra 9 275HX (24 núcleos, frequência máxima de 5,4 GHz), 122 GB de memória RAM e armazenamento em SSD NVMe Kingston de 1,9 TB (modelo SFYR2S2T0), no qual foram hospedados o sistema operacional Ubuntu 26.04 LTS e os contêineres da aplicação. O gerenciamento dos contêineres utilizou o Docker na versão 29.2.1. Todos os serviços foram executados nos contêineres detalhados no Capítulo 4, de modo que os tempos medidos refletem o ambiente reproduzível proposto pelo trabalho.

Para a comparação de maneira visual, foram escolhidos pares com origem e destino em diferentes faixas de distância, sendo elas percursos curtos, no interior de uma mesma cidade; percursos médios, entre municípios; e percursos longos, encontrados entre Uberlândia e capitais de estados distintos. Essa distinção permitiu observar o comportamento dos algoritmos em diferentes profundidades de busca sobre o grafo.

Contando ainda com a avaliação de desempenho, cada um dos caminhos teve seu cálculo realizado a partir dos três algoritmos iniciais, com uma mesma busca de arestas, ofertando meios similares para a sua aplicação. Cada uma das procuras foi refeita por cinco vezes, com a permanência do tempo médio para a execução e também mantendo o desvio padrão de cada uma dessas execuções, sendo que o primeiro resultado da medição

foi descartado, para assim reduzir o efeito de *cache* frio. Os trajetos dentro da área urbana descritos tiveram uma análise sobre a malha viária que foi formulada pela demanda (Seção 4.7), e os demais trajetos foram analisados sobre a malha nacional.

5.2 Teste de Menor Caminho — Comparação Visual

Entendendo que os três algoritmos avaliados seguem métodos exatos, e que a heurística euclidiana adotada é admissível, conforme discutido no Capítulo 2, a execução foi realizada com a expectativa de que as rotas retornadas para um mesmo par origem-destino apresentassem custo idêntico, ainda que pudessem existir empates entre caminhos de comprimento igual. A comparação visual tem por objetivo confirmar esse comportamento sobre a malha real.

As Figuras 5.1 e 5.2 demonstram os caminhos calculados para os pares escolhidos, apresentados na interface do sistema com cada um dos três algoritmos. É possível perceber que em todos os pares testados, os três algoritmos de escolha trouxeram como produto final rotas de traçado idêntico, permitindo a confirmação da correção sobre o implementado: se tratando de metodologias precisas que atuam sobre um único grafo, o trajeto de custo mínimo resultante é exatamente o mesmo, sendo modificada apenas o seu método de procura e, desse modo, o tempo de processamento. Diferenças de margens sobre a distância total, com ordem de poucos metros são visualizadas nos caminhos de maior extensão, decorridas por empates entre caminhos que apresentam custo equivalente.

Como referência qualitativa externa, cada traçado foi também confrontado com o serviço Google Maps, verificando-se que há uma similaridade geral nas rotas que foram calculadas para a malha resultante, frente a outras rotas para a malha nacional que concluem roteiros de maior distância, que tem como justificativa a falta de vias com hierarquia menor e também por meio da adoção de grafo que seja não direcionado e com custos fixos, conforme o que foi discutido na Seção 4.7.

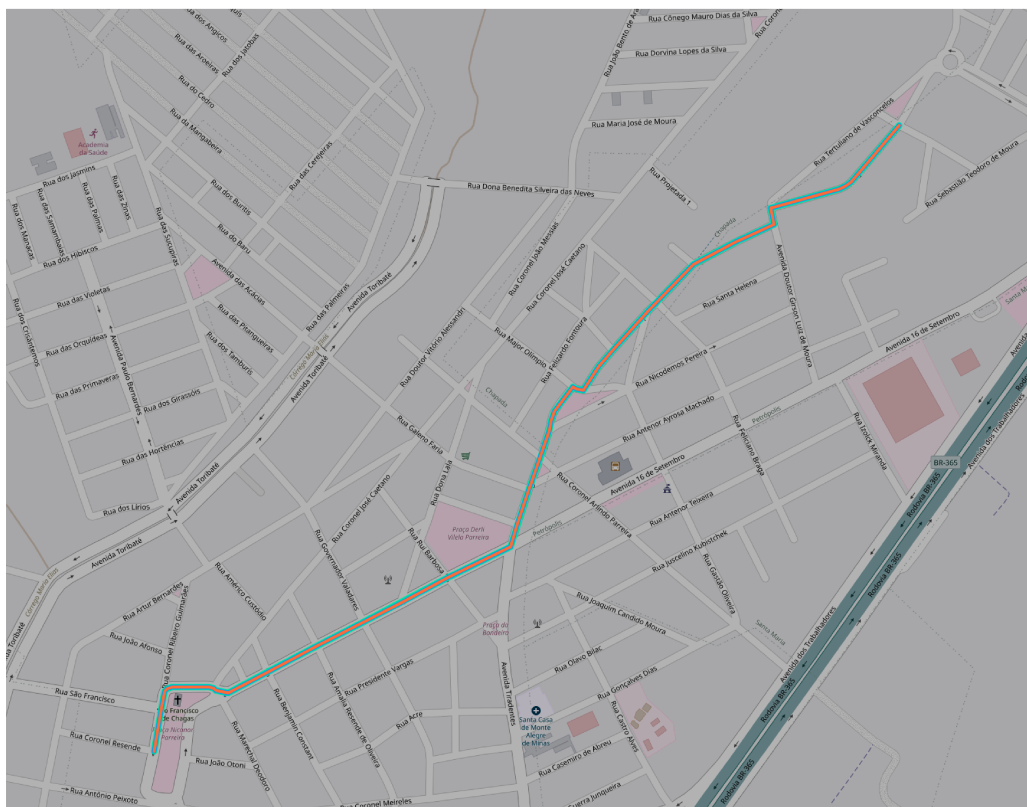


Figura 5.1 – Rotas calculadas pelos três algoritmos, sobrepostas, para um mesmo par origem–destino em Monte Alegre de Minas

Fonte: Elaborado pelo autor.

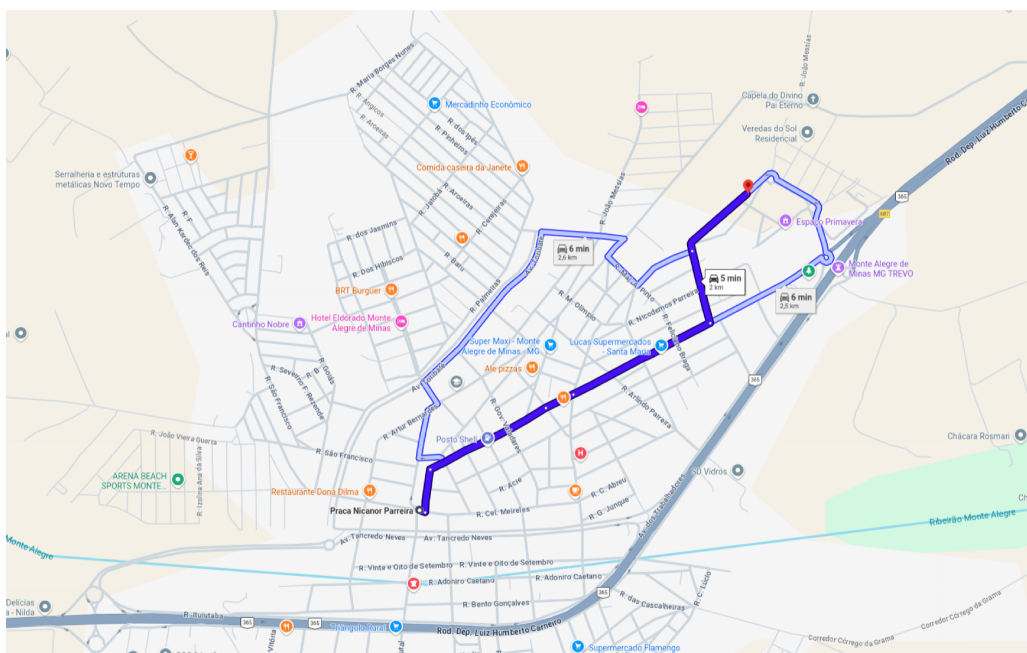


Figura 5.2 – Mesmo percurso apresentado pelo Google Maps, como referência qualitativa externa

Fonte: Google Maps (2026).

5.3 Teste de Desempenho — Comparação dos Algoritmos

A Tabela 5.1 e a Figura 5.3 apresentam os tempos médios de resposta obtidos para as rotas geradas, agrupadas por faixa de distância. Em análise a cada uma das rotas obtidas no trabalho, o algoritmo de Dijkstra foi o que obteve resultados com menor tempo médio de resposta, logo após o A* bidirecional conseguiu o segundo lugar, e por último, o A*. Para a malha nacional houveram resultados sobre os tempos do Dijkstra, que se encontravam entre 1.000 a 1.120 milissegundos, diferentemente do 1.330 a 1.410 milissegundos em que foram encontrados os resultados de tempo para o uso das variantes com heurística.

Tabela 5.1 – Tempos médios de resposta dos algoritmos por rota

Rota	Malha	Dist. (km)	Dijkstra (ms)	A* (ms)	Bi-A* (ms)
Monte Alegre	Detalhada	1,94	31	34	33
UFU–Glória	Nacional	6,54	1041	1345	1350
UFU–HC	Nacional	15,57	1029	1337	1322
UDI–BH	Nacional	644,63	1087	1378	1397
UDI–Salvador	Nacional	1961,24	1118	1413	1405
UDI–Fortaleza	Nacional	2615,09	1103	1390	1354

Fonte: Elaborado pelo autor.

Em períodos menores, a diferença entre os algoritmos foi pequena, como na rota intraurbana da cidade de Monte Alegre, em que os três obtiveram diferenças de aproximadamente 32 milissegundos, sendo que o grafo reduzido dessa malha detalhada possibilitou que o custo de busca se tornasse desprezível perante o tempo fixo necessário para a montagem da busca.

Já para os percursos grandes, de modo contrário ao que foi interpretado nas coletas da fundamentação teórica, o método Dijkstra permaneceu com vantagem sobre outras. Foi verificado também que, o tempo do Dijkstra foi mantido quase sempre constante, mesmo com a variação da distância, como por exemplo a apresentação de resultados de aproximadamente 1.090 milissegundos de Belo Horizonte a Fortaleza, que apresentavam uma variação de distância entre 645 a 2.615 quilômetros, ou seja, tal fato demonstra que o tempo de resposta é dependente de um custo fixo e não pela própria busca, sendo sua estruturação de busca e a inserção do grafo através da função do pgRouting elementos desse custo fixo.

Ainda sobre esse cenário, o custo adicional para realizar a análise heurística euclidiana de cada um dos vértices, como é requisitado pelos métodos A* e A* bidirecional, acaba ultrapassando a economia anteriormente adquirida por meio da diminuição no número de vértices expandidos, o que demonstra a obtenção de tempos maiores do que os obtidos pelo

Dijkstra. Já os desvios padrão permaneceram com valores reduzidos frente às médias, em sua maioria abaixo de 4%, ou seja, indicando medições estáveis. Para o A* bidirecional foi observado um desenvolvimento similar ao A* padrão, sem possuir ganhos significativos, justificados através da necessidade de manutenção de duas diferentes frentes para uma busca simultânea em grafo único.

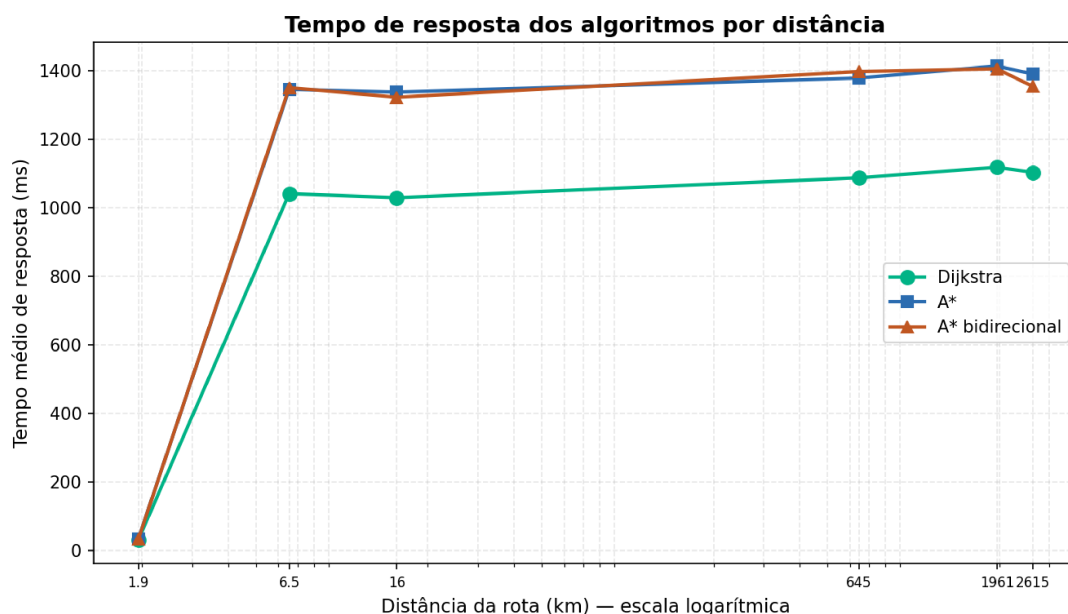


Figura 5.3 – Tempo médio de resposta dos três algoritmos em função da distância da rota (escala logarítmica)

Fonte: Elaborado pelo autor.

5.4 Teste de Escalabilidade

A importação do extrato nacional com as malhas brasileiras foi realizada a partir de um arquivo PBF de aproximadamente 1,9 GB e gerou a tabela `planet_osm_roads` com um total de 673.282 registros, conforme detalhado no Capítulo 2, e sua conclusão ocorreu em 10 minutos e 5 segundos, apresentando como resultado final uma base de dados de aproximadamente 59 GB. Tal valor de tempo foi obtido por meio de um *cache* composto por nós de 24 GB em memória, sendo nestes armazenada a quantidade de 331 milhões de nós do extrato, possuindo uma taxa de acerto de 100%, o que dispensa a releitura em disco, já calculado seu dimensionamento através da viabilidade disponível na memória da máquina de testes. Já a formação da topologia com a função `pgr_createTopology` foi executada em 2 minutos e 36 segundos, resultando em um grafo de 673.282 arestas e 671.093 vértices.

A partir da análise da rede que contém a função `pgr_analyzeGraph` foram obtidas informações sobre os detalhes característicos da base colaborativa como o OpenStreetMap,

sendo encontrado entre elas 54.373 segmentos isolados e também 194.493 segmentos sem saída. Assim, pode-se identificar como o maior componente conectado consegue reunir um número de 370.310 arestas, ou seja, cerca de 55% do total, motivando assim a limitação da busca pelo vértice mais próximo deste, e evitando também que sejam escolhidos pontos que não possuam rota possível.

Ainda nos dados obtidos sobre essa base, o sistema preservou os mesmos tempos de resposta, sendo seus resultados verificados em ordem de um segundo, até para os percursos interestaduais de mais de dois mil quilômetros, sem degradação crítica de desempenho, o que aponta para a viabilidade da arquitetura no escopo nacional. Conforme observado na Seção 5.3, os valores do tempo de resposta se encontram estáveis, independentemente da grande expansão da distância, mostrando que a eficácia pouco interferida pelo tamanho do trajeto, mas pode variar conforme o custo fixo.

5.5 Comparação com o Sistema de (Oliveira, 2021)

Para a comparação, ressalta-se que o sistema desenvolvido por (Oliveira, 2021) operava em escopo municipal, com importação realizada pelo `osm2pgrouting` e arquitetura baseada em Java com Spring Boot, disponibilizando apenas o algoritmo de Dijkstra. Já a presente proposta amplia esse escopo para a malha viária nacional, substituindo a etapa de importação pelo `osm2pgsql` com a criação posterior da topologia, adotando o FastAPI na camada de serviços e adicionando os algoritmos A* e A* bidirecional, além da distribuição por contêineres. A Tabela 5.2 sintetiza a comparação entre os dois sistemas.

Tabela 5.2 – Comparação entre o sistema de Oliveira (2021) e a presente proposta

Característica	Oliveira (2021)	Presente proposta
Escopo	Municipal	Nacional (Brasil)
Importação	<code>osm2pgrouting</code>	<code>osm2pgsql</code> + topologia
Arquitetura	Java / Spring Boot	Python / FastAPI
Algoritmos	Dijkstra	Dijkstra, A*, A* bidirecional
Malha detalhada	Não	Sob demanda por cidade
Distribuição em contêineres	Não	Docker Compose

Fonte: Elaborado pelo autor.

Quando comparados ambos os sistemas, encontra-se a limitação aos aspectos relacionados ao escopo arquitetura e recursos disponibilizados, assumindo que a medição de maneira direto do desempenho não supria o objetivo da pesquisa e o modelo de sistema anterior não foi novamente executado para essa análise. Apesar deste fato, é ressaltado que a evolução que foi proposta proposta pode ser verificada, sendo ela referente ao aumento do escopo municipal para o nacional, a substituição da cadeia de importação por outra alternativa que permita a escalada, a inserção de mais dois algoritmos adicionais para o

caminho mínimo e também a adoção de uma arquitetura que pudesse ser distribuída em contêineres.

5.6 Discussão dos Resultados

Os resultados obtidos durante o trabalho indicam que o algoritmo de Dijkstra teve o produto final com menor tempo médio de resposta em cada um dos cenários avaliados, inclusive junto com os percursos maiores, conseguindo ultrapassar tanto o A^* quanto o A^* bidirecional. Tal comportamento vai contra a expectativa levantada anteriormente na fundamentação teórica que foi apresentada no Capítulo 2 a busca direcionada tenderia a reduzir o número de vértices expandidos quando comparada à expansão uniforme do Dijkstra, com ganho mais evidente conforme aumenta a distância entre origem e destino, enquanto a variante bidirecional dependeria do critério de parada e da qualidade da heurística para compensar o custo das duas frentes de busca.

A divergência ocorrida entre os comportamentos esperado e observado justifica-se por meio de dois diferentes fatores. O primeiro, trata-se do custo para realizar a avaliação da heurística euclidiana para cada vértice expandido, conforme é exigido pelas metodologias A^* e A^* bidirecional, o que resultou, na prática, em uma economia adquirida através da diminuição do número de vértices explorados. Ainda de forma mais evidente, o segundo fator foi o tempo de resposta, que apresentou-se mais estável, onde o tempo de resposta permaneceu estático mesmo com a variação da distância do trajeto, mostrando que a eficácia é atrelada a um custo fixo, que é composto pela associado à montagem da consulta e ao carregamento do grafo através das funções do pgRouting, e não de acordo com a procura da rota propriamente dita. Por meio desse regime então, a procura da rota é responsável apenas por uma parcela reduzida do tempo total calculado, sendo que a heurística não apresenta espaço para poder compensar seu próprio custo, e a variante bidirecional não teria um ganho consistente para apresentar nos resultados sobre o A^* padrão.

Deve-se ainda destacar que as medidas de tempo refletem as funções padrão do pgRouting, sem técnicas de pré-processamento como as adotadas pelos motores especializados discutidos no Capítulo 3, de modo que os valores não podem ser comparados aos relatados por aqueles trabalhos. Por fim, tem-se que as limitações assumidas de custos fixos e grafo não direcionado precisam ser consideradas na leitura e interpretação dos resultados, pois as rotas calculadas podem divergir das condições reais de circulação.

Conclusão

6.1 Considerações Finais

Este trabalho realizou uma análise de desempenho de três algoritmos de caminho mínimo: Dijkstra, A* e A* bidirecional, aplicados ao roteamento geoespacial sobre a malha viária brasileira em escala nacional. Para viabilizar essa comparação em condições controladas e reprodutíveis, foi construído um ambiente experimental baseado no PostgreSQL com as extensões PostGIS e pgRouting, com camada de serviços em FastAPI, interface em React e Leaflet e distribuição por contêineres Docker. A base nacional do OpenStreetMap foi importada com o `osm2pgsql` em modo *slim*, seguida da construção da topologia do grafo, permitindo que os três algoritmos fossem executados exatamente sobre os mesmos dados e sob as mesmas condições de execução.

O principal resultado da análise é que o algoritmo de Dijkstra apresentou o menor tempo de resposta em todas as rotas avaliadas, independentemente da distância, contrariando a expectativa teórica e resultados empíricos anteriores, como os de [Rachmawati e Gustin \(2020\)](#), de que as buscas guiadas por heurística seriam mais rápidas. A investigação dos tempos medidos indicou que essa diferença decorre de um custo fixo de montagem da consulta e carregamento do grafo, que domina o tempo total, e não da etapa de busca propriamente dita. Em escala nacional, portanto, o gargalo de desempenho desloca-se da complexidade do algoritmo para o custo de preparação dos dados, constituindo a principal contribuição analítica deste trabalho e um ponto de atenção relevante para o uso do pgRouting em bases de grande porte.

Como contribuição metodológica adicional, foi proposto um mecanismo de construção de malhas urbanas detalhadas por demanda, acionado pela própria solicitação do usuário, que aumenta a precisão do roteamento intraurbano sem comprometer a escala nacional da base. O ambiente experimental desenvolvido, reprodutível por um único comando e independente de serviços pagos de roteamento e geocodificação, mostrou-se adequado como instrumento para esse tipo de análise comparativa e pode ser reaproveitado em estudos futuros.

6.2 Limitações

Algumas limitações delimitam o alcance dos resultados. O grafo foi tratado como não direcionado, conforme explicado no início do trabalho, de modo que as restrições de sentido único das vias não são modeladas e, assim, as rotas calculadas podem divergir das condições reais de circulação. Os custos das arestas permanecem fixos e foram baseados apenas no comprimento dos trechos, sem inserção de dados de tráfego em tempo real, limites de velocidade ou restrições veiculares. A qualidade dos resultados também foi condicionada pela completude da base do OpenStreetMap, a qual apresenta heterogeneidade territorial, possuindo lacunas de cobertura e de toponímia em determinadas regiões, de acordo com o discutido no Capítulo 2.

Ainda em análise a malha nacional que foi importada, observou-se que a mesma possui somente vias com maior hierarquia, gerando como produto rotas que são estritamente urbanas e mais extensas do que seriam os percursos reais, sendo ultrapassadas as restrições através da inserção de malhas detalhadas por demanda do usuário, mas ainda assim, não abrangente para ser encontrada em todo o território nacional. Por fim, a metodologia desse sistema é dependente de um trabalho externo comandado pelo usuário, onde a geocodificação é feita para converter endereços e coordenadas.

6.3 Trabalhos Futuros

Como evolução natural do sistema, propõe-se, em trabalhos futuros, o tratamento do grafo como direcionado, com o uso da coluna `reverse_cost` para modelar as vias de mão única, aproximando as rotas calculadas das condições reais de circulação. Outra extensão relevante é a oferta de múltiplas rotas alternativas para um mesmo par origem-destino, a partir de algoritmos de K caminhos mínimos (Yen, 1971), e a incorporação de custos dinâmicos baseados em dados de tráfego em tempo real. . Em relação a malha detalhada, a proposta visa aumentar a formação de malhas por todo o território nacional, de modo pré-processado, em que seja possível realizar o roteamento intraurbano preciso em qualquer localidade selecionada. No plano da cobertura, o sistema pode ainda ser estendido a mais países da América do Sul, aproveitando a mesma cadeia de importação, e disponibilizado também em aplicativo móvel. Por fim, propõe-se a comparação do desempenho das funções do pgRouting com motores de roteamento especializados, como o OSRM (Luxen; Vetter, 2011) e o GraphHopper.

Referências

AGAFONKIN, Volodymyr. **Leaflet — an open-source JavaScript library for interactive maps**. [S. l.: s. n.], 2026. <https://leafletjs.com>. Acesso em: 5 ago. 2026.

ANTONIOU, Vyron. A Review of OpenStreetMap Data. *In: MAPPING and the Citizen Sensor*. [S. l.]: Ubiquity Press, 2017. p. 37–59. ISBN 978-1-911529-16-3. DOI: 10.5334/bbf.c. Disponível em: <https://doi.org/10.5334/bbf.c>.

BARRINGTON-LEIGH, Christopher; MILLARD-BALL, Adam. The world's user-generated road map is more than 80% complete. **PLoS ONE**, v. 12, n. 8, e0180698, 2017. Disponível em: <https://doi.org/10.1371/journal.pone.0180698>.

BONDI, André B. Characteristics of scalability and their impact on performance. *In: 2ND International Workshop on Software and Performance*. New York, NY, USA: ACM, 2000. p. 195–203. (WOSP '00). ISBN 978-1-58113-195-6. DOI: 10.1145/350391.350432. Disponível em: <https://doi.org/10.1145/350391.350432>.

BUTLER, Howard; DALY, Martin; DOYLE, Allan; GILLIES, Sean; HAGEN, Stefan; SCHAUB, Tim. **The GeoJSON Format**. [S. l.], ago. 2016. DOI: 10.17487/RFC7946. Disponível em: <https://www.rfc-editor.org/rfc/rfc7946>.

BY, Rolf A. de; KNIPPERS, Richard A.; SUN, Yuxian; ELLIS, Martin C.; KRAAK, Menno-Jan; WEIR, Michael J. C.; GEORGIADOU, Yola; RADWAN, Mostafa M.; WESTEN, Cees J. van; KAINZ, Wolfgang; SIDES, Edmund J. **Principles of Geographic Information Systems: An Introductory Textbook**. Enschede: ITC, 2001.

CASANOVA, Marco A.; CAMARA, Gilberto; DAVIS, Clodoveu; VINHAS, Lúbia; QUEIROZ, Gilberto Ribeiro de. **Bancos de Dados Geográficos**. Curitiba: Mundogeo, 2005.

CHADHA, Chetan; GARG, Shivang. Shortest Path Analysis on Geospatial Data Using pgRouting. *In: INTERNATIONAL Conference on Innovative Computing and Communications*. [S. l.]: Springer, 2019. v. 55. (Lecture Notes in Networks and

Systems). DOI: 10.1007/978-981-13-2324-9_20. Disponível em: https://doi.org/10.1007/978-981-13-2324-9_20.

CHAMBERLAIN, Ryan; SCHOMMER, Jennifer. Using Docker to Support Reproducible Research. figshare, 2014. DOI: 10.6084/m9.figshare.1101910.v1. Disponível em: https://figshare.com/articles/journal_contribution/Using_Docker_to_Support_Reproducible_Research/1101910/1. Acesso em: 21 abr. 2026.

CHANG, Kang-tsung. **Introduction to Geographic Information Systems**. 9a ed. New York: McGraw-Hill, 2019.

CLARKE, Keith C. **Analytical and Computer Cartography**. 2. ed. Englewood Cliffs: Prentice Hall, 1995. p. 290. ISBN 0130334812.

CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. **Algoritmos: teoria e prática**. 3. ed. Rio de Janeiro: Elsevier, 2012.

DIBBELT, Julian; STRASSER, Ben; WAGNER, Dorothea. Fast Exact Shortest Path and Distance Queries on Road Networks with Parametrized Costs. **arXiv:1509.03165 [cs.DS]**, 2015. DOI: 10.48550/arXiv.1509.03165. Disponível em: <https://doi.org/10.48550/arXiv.1509.03165>.

DIJKSTRA, Edsger W. A note on two problems in connexion with graphs. **Numerische Mathematik**, v. 1, p. 269–271, 1959.

DOCKER, INC. **Docker Compose documentation**. [*S. l.: s. n.*], 2026. <https://docs.docker.com/compose/>. Acesso em: 6 ago. 2026.

ELIAS, Elias Nasr Naim; OLIVEIRA FERNANDES, Vivian de; JUNIOR, Mauro José Alixandrini; SCHMIDT, Marcio Augusto Reolon. The quality of OpenStreetMap in a large metropolis in northeast Brazil. **Boletim de Ciências Geodésicas**, v. 26, n. 3, e2020012, 2020. Disponível em: <https://doi.org/10.1590/s1982-21702020000300012>.

GEOFABRIK GMBH. **Brazil OpenStreetMap Data Extract**. [*S. l.: s. n.*], 2024. Disponível em: <https://download.geofabrik.de/south-america/brazil.html>. Acesso em: 2025.

GEOFABRIK GMBH. **Brazil OpenStreetMap Data Extract**. [*S. l.: s. n.*], 2026. Disponível em: <https://download.geofabrik.de/south-america/brazil.html>. Extrato de 5 ago. 2026. Acesso em: 6 ago. 2026.

GRAHAM, Alastair. **Geographic Information Systems & Science**. [*S. l.*]: Taylor & Francis, 2012. v. 33, p. 4291–4292. DOI: 10.1080/01431161.2011.587098. Disponível em: <https://doi.org/10.1080/01431161.2011.587098>.

KIVIJARVI, Hannu; PARNANEN, Karoliina. Instrumental Usability and Effective User Experience: Interwoven Drivers and Outcomes of Human-Computer Interaction.

International Journal of Human-Computer Interaction, 2021. DOI:

10.1080/10447318.2021.2016236. Disponível em:

<https://doi.org/10.1080/10447318.2021.2016236>. Acesso em: 21 abr. 2026.

LI, Songnian; DRAGICEVIC, Suzana; CASTRO, Francesc Antón; SESTER, Monika; WINTER, Stephan; COLTEKIN, Arzu; PETTIT, Christopher; JIANG, Bin; HAWORTH, James; STEIN, Alfred; CHENG, Tao. Geospatial big data handling theory and methods: A review and research challenges. **ISPRS Journal of**

Photogrammetry and Remote Sensing, v. 115, p. 119–133, 2016. DOI:

10.1016/j.isprsjprs.2015.10.012. Disponível em:

<https://doi.org/10.1016/j.isprsjprs.2015.10.012>.

LUXEN, Dennis; VETTER, Christian. Real-Time Routing with OpenStreetMap Data.

In: 19TH ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. New York, NY, USA: ACM, 2011. (GIS'11), p. 513–516. DOI:

10.1145/2093973.2094062. Disponível em:

<https://doi.org/10.1145/2093973.2094062>.

MERKEL, Dirk. Docker: Lightweight Linux Containers for Consistent Development and Deployment. **Linux Journal**, v. 2014, n. 239, 2014.

META PLATFORMS. **React: a JavaScript library for building user interfaces**.

[*S. l.: s. n.*], 2013. Disponível em: <https://react.dev>. Versão 18. Acesso em: 16 abr. 2026.

MILER, Mario; MEDAK, Damir; ODOBAŠIĆ, Dražen. The Shortest Path Algorithm Performance Comparison in Graph and Relational Database on a Transportation

Network. **Promet – Traffic & Transportation**, v. 26, n. 1, p. 75–82, 2014. Disponível em: <https://doi.org/10.7307/ptt.v26i1.1268>.

NIK YUSOFF, Nik Mohd Ramli; MOHD SHAFRI, Helmi Zulhaidi;

MUNIANDY, Ratnasamy; WALI, Islam Mohammad. Development of Obstacle Avoidance Technique in Web-Based Geographic Information System for Traffic

Management Using Open Source Software. **Journal of Computer Science**, v. 10, n. 7, p. 1259–1268, 2014. DOI: 10.3844/jcssp.2014.1259.1268. Disponível em:

<https://doi.org/10.3844/jcssp.2014.1259.1268>.

OLIVEIRA, Eduardo Ferreira de. **Um sistema para cálculo de rotas de caminho mínimo utilizando pgRouting e dados do OpenStreetMap**. [*S. l.: s. n.*], 2021.

<https://repositorio.ufu.br/handle/123456789/32224>. Trabalho de Conclusão de Curso, Graduação em Sistemas de Informação, Universidade Federal de Uberlândia (UFU).

OPENSTREETMAP FOUNDATION. **Nominatim — Open-source geocoding with OpenStreetMap data**. [S. l.: s. n.], 2026. <https://nominatim.org>. Acesso em: 7 ago. 2026.

OPENSTREETMAP WIKI. **Node**. [S. l.: s. n.], 2026. <https://wiki.openstreetmap.org/wiki/Node>. Acesso em: 4 ago. 2026.

OSM2PGSQL DEVELOPERS. **osm2pgsql v2 Manual**. [S. l.: s. n.], 2026. <https://osm2pgsql.org/doc/manual.html>. Acesso em: 5 ago. 2026.

PGROUTING. **pgRouting Manual**. [S. l.], 2026. Disponível em: <https://docs.pgrouting.org/>.

POSTGIS. **PostGIS 3.6 Manual**. [S. l.], 2026. Disponível em: <https://postgis.net/docs/manual-3.6/>.

RACHMAWATI, Dian; GUSTIN, Lysander. Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem. **Journal of Physics: Conference Series**, v. 1566, n. 1, p. 012061, 2020. ISSN 1742-6588, 1742-6596. DOI: 10.1088/1742-6596/1566/1/012061. Disponível em: <https://doi.org/10.1088/1742-6596/1566/1/012061>. Acesso em: 21 abr. 2026.

RAMÍREZ, Sebastián. **FastAPI: framework moderno e de alta performance para construção de APIs com Python**. [S. l.: s. n.], 2018. Disponível em: <https://fastapi.tiangolo.com>. Acesso em: 16 abr. 2026.

RIOS, Luis Henrique Oliveira; CHAIMOWICZ, Luiz. PNBA*: A Parallel Bidirectional Heuristic Search Algorithm. *In: XXXI Congresso da Sociedade Brasileira de Computação*. [S. l.: s. n.], 2011. p. 346-357.

THE AEROSPACE CORPORATION. **A Brief History of GPS**. [S. l.: s. n.], 2021. Disponível em: <https://aerospace.org/article/brief-history-gps>.

TIAN, Ting; WU, Huijing; WEI, Haitao; WU, Fang; XU, Mingliang. An Efficient Route Planning Algorithm for Special Vehicles with Large-Scale Road Network Data. **ISPRS International Journal of Geo-Information**, v. 14, n. 2, p. 71, 2025. DOI: 10.3390/ijgi14020071. Disponível em: <https://doi.org/10.3390/ijgi14020071>.

TOPF, Jochen. **Osmium Tool — a versatile command line tool for working with OpenStreetMap data**. [S. l.: s. n.], 2026. <https://osmcode.org/osmium-tool/>. Acesso em: 6 ago. 2026.

WEI, Ching-Chuan; LIN, Jing-Shian; CHANG, Chia-Chi; HUANG, Yung-Fa; LIN, Chuan-Bi. The Development of E-Bike Navigation Technology Based on an OpenStreetMap. **Smart Science**, Taylor & Francis, v. 6, n. 1, p. 29-35, 2018. DOI:

10.1080/23080477.2017.1410011. Disponível em:
<https://doi.org/10.1080/23080477.2017.1410011>.

YEN, Jin Y. Finding the K Shortest Loopless Paths in a Network. **Management Science**, 1971. DOI: 10.1287/mnsc.17.11.712. Disponível em:
<https://doi.org/10.1287/mnsc.17.11.712>. Acesso em: 21 abr. 2026.

ZHANG, Lijing; HE, Xuanhui. Route Search Based on pgRouting. *In*: WU, Y. (ed.). **Software Engineering and Knowledge Engineering: Theory and Practice**. [S. l.]: Springer, 2012. v. 115. (Advances in Intelligent and Soft Computing). p. 1003–1007. DOI: 10.1007/978-3-642-25349-2_133. Disponível em:
https://doi.org/10.1007/978-3-642-25349-2_133.

ZHU, Qixu; LI, Zhichao; DONG, Jinwei; FU, Ping; CHENG, Qu; CAI, Jun; GURGEL, Helen; YANG, Linsheng. Spatiotemporal dataset of dengue influencing factors in Brazil based on geospatial big data cloud computing. **Scientific Data**, abr. 2025. DOI: 10.1038/s41597-025-05045-1. Disponível em:
<https://www.nature.com/articles/s41597-025-05045-1>.

Apêndices

Comandos de Instalação e Configuração

Este apêndice reúne os comandos necessários para reproduzir o ambiente do sistema. Como todas as ferramentas necessárias — incluindo o `osm2pgsql` — estão contidas nos contêineres, o único pré-requisito no computador do usuário é o Docker; os comandos são, portanto, idênticos em Linux, macOS e Windows.

Pré-requisitos

São necessários apenas o Git, o Docker e o Docker Compose instalados.

Obtenção do código

```
git clone https://github.com/coutinhosf/roteamento-geoespacial.git
cd roteamento-geoespacial
```

Configuração da senha

A senha do banco é definida em um arquivo `.env`, criado a partir do modelo `.env.example`.
Linux e macOS:

```
cp .env.example .env
```

Windows (PowerShell):

```
copy .env.example .env
```

Subida da aplicação

Todo o sistema é iniciado com um único comando, a partir da raiz do projeto:

```
docker compose up --build
```

Esse comando sobe os serviços de banco de dados, *back-end* e *front-end*, além do serviço auxiliar **importer**, responsável pela preparação da base. Na primeira execução, o **importer** detecta que o banco está vazio e realiza, de forma automática e sem intervenção manual, as seguintes etapas: o download do extrato do Brasil da Geofabrik (arquivo **brazil-latest.osm.pbf**, de aproximadamente 1,9 GB, salvo na pasta **data/** do projeto), a importação com o **osm2pgsql**, a construção da topologia do grafo com o **pgr_createTopology** e o registro do maior componente conectado da malha, utilizado pelo *back-end*. O progresso pode ser acompanhado com o comando a seguir:

```
docker compose logs -f importer
```

A primeira subida leva algumas horas, em razão do download e da importação do extrato nacional completo. O *back-end* só é iniciado após a conclusão do **importer**; encerrada essa etapa, a interface torna-se plenamente funcional no navegador, conforme descrito no Capítulo 4. Nas execuções seguintes, como os dados são persistidos em volume, o **importer** verifica que a base já está pronta e encerra em segundos, e o sistema inicia com a base disponível.

Uso de um extrato já baixado ou de data específica

Caso o arquivo **brazil.osm.pbf** já tenha sido baixado previamente, basta copiá-lo para a pasta **data/** do projeto antes da subida, e o download é dispensado. Para reproduzir exatamente o extrato utilizado neste trabalho (consolidado em 5 de agosto de 2026), em vez do **latest**, define-se a variável **PBF_URL** no arquivo **.env**:

```
PBF_URL=https://download.geofabrik.de/south-america/brazil-260805.osm.pbf
```

Cabe observar que a Geofabrik mantém os extratos diários datados por tempo limitado no diretório público, preservando a longo prazo apenas os extratos de 1º de janeiro de cada ano. Caso o arquivo datado não esteja mais disponível, deve-se utilizar o extrato **latest** (configuração padrão do projeto), ciente de que pequenas diferenças nos resultados podem ocorrer em razão da evolução natural dos dados colaborativos do OpenStreetMap.

Comandos da Tentativa de Importação com osm2pgrouting

Os comandos a seguir registram a tentativa de importação do extrato nacional por meio do osm2pgrouting, descrita na Seção 2.4.1, que motivou a adoção do osm2pgsql.

```
# Conversao do extrato PBF para OSM XML
sudo apt install osmium-tool
osmium cat brazil.osm.pbf -o brazil.osm

# Cópia para o container e importacao com osm2pgrouting
docker cp brazil.osm my_postgis:/osm_data.osm
docker exec -it my_postgis osm2pgrouting -f /osm_data.osm \
    -d mydatabase -U postgres -W ***** -p 5432 \
    --attributes --tags --clean
```