Universidade Federal de Santa Catarina
Departamento de Informática e Estatística

Matheus Dhanyel Cândido Roque

ESTIMATIVA DE CADEIA DE MARKOV DE MALHA VIÁRIA COM BASE EM DADOS DE TRÁFEGO URBANO

Florianópolis
2024

Matheus Dhanyel Cândido Roque

ESTIMATIVA DE CADEIA DE MARKOV DE MALHA VIÁRIA COM BASE EM DADOS DE TRÁFEGO URBANO

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como parte dos requisitos necessários para a obtenção do Grau de Bacharel em Ciência da Computação.
Orientador: Prof. Dr. Rafael de Santiago

Universidade Federal de Santa Catarina
Departamento de Informática e Estatística

Florianópolis
2024

Matheus Dhanyel Cândido Roque

ESTIMATIVA DE CADEIA DE MARKOV DE MALHA VIÁRIA COM BASE EM DADOS DE TRÁFEGO URBANO

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Comissão Examinadora

Prof. Dr. Rafael de Santiago
Universidade Federal de Santa Catarina
Orientador

Prof. Dr. Maicon Rafael Zatelli
Universidade Federal de Santa Catarina

Prof. Dr.Pedro Belin Castellucci
Universidade Federal de Santa Catarina

Florianópolis, 18 de dezembro de 2024

"In the darkest times, hope is something you give yourself. That is the meaning of inner strength"

(Iroh)

Resumo

Com quase um terço da população de grandes cidades no Brasil sofrendo com engarrafamento todos os dias (GALINDO ERNESTO PEREIRA E LIMA NETO, 2019), melhorias proporcionadas por trabalhos na área de mobilidade urbana tornam-se cada vez mais valiosas. Contudo, comumente são necessários dados reais ou, pelo menos, boas representações da realidade para o desenvolvimento de pesquisas nessa área. Com isso em pauta, este trabalho tem como foco expandir em cima de tais representações, utilizando dados de alta qualidade (REZZOUQI et al., 2019) para realizar tais estimativas. Baseando-se de modelos de representações da malha viária como uma cadeia de Markov a partir de dados gratuitos do *OpenStreetMap*, como utilizados por Salman Sinan; Alaswad (2018), este trabalho utiliza também dados obtidos pelo *Google Maps* para criar cadeias de Markov a partir de dados com maior representatividade da realidade dos fluxos urbanos. A partir de análises das estimativas obtidas pela ferramenta desenvolvida nota-se que os resultados das novas representações são semelhantes dos obtidos a partir do *OpenStreetMap* sozinho, tornando necessário discussões sobre que fatores causam essas similaridades bem como formas de comparar a qualidade dos resultados de cada abordagem.

Palavras-Chave: Cadeias de Markov; Dados de tráfego; Google Maps; OpenStreetMap.

Abstract

With almost a third of population from big cities from Brazil dealing with traffic jams daily (GALINDO ERNESTO PEREIRA E LIMA NETO, 2019), improvements brought from papers in the field of urban mobility become increasingly more valuable. However, real data or, at least, good representations from reality are commonly needed for researches developed in this area. With this in mind, this paper focus on expanding upon said representations, using high quality data (REZZOUQI et al., 2019) for such estimates. Based on representation models for traffic networks as Markov chains out of free data from OpenStreetMap, as used by Salman Sinan; Alaswad (2018), this paper also uses data obtained from Google Maps to create Markov chain out of data with better representativeness real urban flows. Based on analyses done with estimates obtained from the tool developed, it is noted that the results from the new representations are resembles those obtained from OpenStreetMap alone, making it necessary to discuss which factors causes this similarity as well ways to compare the quality from the results for each approach.

Keywords: Markov chain;Traffic data; Google Maps; OpenStreetMap.

Lista de figuras

Figura 1 – Exemplo de malha com três vias representada como uma cadeia de Markov (SALMAN SINAN; ALASWAD, 2018) 14
Figura 2 – Exemplos de representação do diagrama fundamental. À esquerda, velocidade vs densidade; À direita, fluxo de tráfego vs densidade. (KACHROO; SASTRY, 2016) 17
Figura 3 – Diagrama fundamental separado por níveis de serviço e com linha de tendência gerada por modelo exponencial. Obtido por observações em rodovia de Paris em 2006. (GUESSOUS et al., 2014) 18
Figura 4 – Resultados de experimento. À esquerda, mapa de calor de densidade de tráfego com a disposição atual das vias; À direita, mapa de calor de densidade de tráfego com a disposição de vias sugerida (SALMAN SINAN; ALASWAD, 2018) 21
Figura 5 – Exemplo de consulta no *Google Maps* para um dos trechos selecionados (CHURI, 2021) 22
Figura 6 – Gráfico de velocidade observada vs tempo em experimento realizado com a ferramenta desenvolvida por Mostafi e Elgazzar (2021) 24
Figura 7 – Gráfico de número de iterações realizadas vs número de observações de tempo de travessia, colorido com diferentes cores para cada valor de *callbackInterval* testado (MOSTAFI; ELGAZZAR, 2021). 25
Figura 8 – Gráfico de tempo de processamento decorrido vs número de iterações realizadas, com pontos para cada valor de *callbackInterval* testado (MOSTAFI; ELGAZZAR, 2021). 25
Figura 9 – Grafo representativo dos dados provenientes da OSMnx. 29
Figura 10 – Grafo representativo da cadeia de Markov. 30
Figura 11 – Mapa de calor de densidade estimada baseado em dados da OSM. Legenda em veículos/km/faixa 36
Figura 12 – Mapa de calor de densidade estimada baseado em dados da *Google Maps*. Legenda em veículos/km/faixa 37
Figura 13 – Diferença de densidades entre resultados da OSM vs *Google Maps* . . . 38
Figura 14 – Diferença de densidades entre resultados da OSM vs *Google Maps* para experimento realizado com o bairro do Brás, São Paulo, horário aproximado às 17 horas de uma terça-feira 39

Lista de tabelas

Tabela 1 – Tabela com principais elementos de trabalhos relacionados discutidos . 26
Tabela 2 – Tabela de preços para o serviço do *Google Directions API* 28

Sumário

1 INTRODUÇÃO 10
1.1 Objetivos 11
1.1.1 Objetivo Geral 11
1.1.2 Objetivos Específicos 11
1.2 Definição do problema computacional 12
1.2.1 Metodologia 12
1.2.2 Estrutura do Texto 13

2 FUNDAMENTAÇÃO TEÓRICA 14
2.1 Cadeias de Markov 14
2.1.1 Propriedades de cadeias de Markov 15
2.1.1.1 Classes, periodicidade e redutibilidade 15
2.1.1.2 Transiência, recorrência e ergodicidade 15
2.1.1.3 Aplicação das propriedades 16
2.2 Variáveis Macroscópicas de Tráfego 16
2.2.1 Fluxo de tráfego 16
2.2.2 Velocidade 16
2.2.2.1 Velocidade de fluxo livre 17
2.2.3 Densidade 17
2.2.4 Diagrama Fundamental 17

3 TRABALHOS RELACIONADOS 20
3.1 Alleviating Road Network Congestion: Traffic Pattern Optimization Using Markov Chain Traffic Assignment 20
3.2 Google Map Traffic Data Scraping and Mining 21
3.3 An Open Source Tool to Extract Traffic Data from Google Maps: Limitations and Challenges 23
3.4 Tabela comparativa 26

4 DESENVOLVIMENTO 27
4.1 Ferramentas utilizadas 27
4.1.1 OSMnx 27
4.1.2 google_maps 27
4.2 Implementação 28
4.2.1 Pré-processamento de dados 28
4.2.2 Estruturação da cadeia de Markov 29

4.2.3 Cálculo de velocidade média e tempo de travessia 30
4.2.4 Cálculo de transições 31
4.2.5 Cálculo de densidades 32

5 ANÁLISE DE RESULTADOS 34
5.1 Objetivo dos experimentos 34
5.2 Experimentos 34
5.3 Discussão de resultados 35

6 CONCLUSÕES 40
6.1 Trabalhos futuros 41

REFERÊNCIAS BIBLIOGRÁFICAS 42

7 APÊNDICES 45
7.1 Exemplos 45
7.2 Código-fonte 48
7.2.1 Python 49
7.2.1.1 ___init___.py 49
7.2.1.2 cli.py 56
7.2.1.3 pyproject.toml 57
7.2.2 Rust 58
7.2.2.1 main.rs 58
7.2.2.2 lib.rs 60
7.2.2.3 data_reader.rs 60
7.2.2.4 osm.rs 62
7.2.2.5 google_routes.rs 62
7.2.2.6 markov_chain.rs 64
7.2.2.7 Cargo.toml 75

1 Introdução

Com o Brasil contando com uma frota terrestre de mais de 113 milhões de veículos, sendo 1.2% deste valor referente ao crescimento no primeiro semestre de 2022 (SENATRAN, 2022), cada vez mais a malha viária enfrenta dificuldades para suprir com qualidade as necessidades de deslocamentos intra-urbanos das populações. Segundo pesquisa (GALINDO ERNESTO PEREIRA E LIMA NETO, 2019), mais de 30% da população das cidades com mais de 100 mil habitantes alegavam enfrentar engarrafamentos diariamente, mostrando um grande espaço para melhoria na eficiência do planejamento da mobilidade urbana.

Para melhorar a satisfação das necessidades de mobilidade urbana várias frentes podem ser abordadas, seja ampliando o uso de transportes coletivos, investindo em novos modais para dividir a carga ou replanejando a disposição e utilização das vias a fim de otimizar a sua vazão. Independente de qual a abordagem feita, para todas é crítico o conhecimento de qual o comportamento dos fluxos urbanos, identificando quais rotas e vias se encontram em sobrecarga e utilizando desta informação para traçar alvos de ação mais bem definidos e métricas de qualidade mais direcionadas.

A obtenção de tais dados é usualmente feita por pesquisas de campo, um processo caro e lento. Uma alternativa que vem crescendo é a coleta e processamento computacional de dados diversos para construir matrizes de origem-destino, como dados de localização de *smartphones* (JUNIOR; MEDRANO; CRUVINEL, 2018) ou bilhetes eletrônicos de embarque (PELLETIER; TRéPANIER; MORENCY, 2011), em muitos casos facilitando o processo e obtendo resultados comparáveis com a pesquisa de campo.

Outra via a ser considerada na questão de dados de tráfego é substituir a implementação de processos de obtenção de informações como os já citados por uma extração em dados já coletados por aplicações e serviços de mobilidade urbana como *Waze* e *Google Maps*, que utilizam dados de uma grande quantidade de usuários para alimentar seus cálculos. Tais dados de tráfego gratuitos possuem poucos detalhes, gerando uma necessidade de maior pré-processamento antes de seu uso (MOSTAFI; ELGAZZAR, 2021), o que não impede que ferramentas desenvolvidas tratem esses dados não estruturados para gerar informações úteis (CHURI, 2021).

Tanto para Churi (2021) quanto Mostafi e Elgazzar (2021) a informação alvo a ser alcançada em suas pesquisas foi um perfil de velocidade média para o trecho de coleta de dados. Essas informações podem ser utilizados em processamentos adicionais para obter novas informações, já que existe uma relação entre velocidade média, densidade de veículos e fluxo de tráfego (REILLY, 1997). Com a evolução desta área de estudo, diversos modelos foram propostos para equacionar essa relação (GADDAM; RAO, 2019),

desde os primeiros modelos propostos como Greenshield e Underwood quanto outros mais recentes como de Wang et al. (2013), progressivamente melhorando a capacidade de prever e representar o comportamento dessa relação com maior precisão.

Todas essas informações podem ser utilizadas para diversos fins práticos diferentes:

- Wang e Xu (2011) utilizaram dados de tempo de viagem que foram obtidos usando a *Google Maps API* para calibrar a matriz de tempo de viagem de pontos origem-destino em um *software* de geoprocessamento;
- Muñoz-Villamizar et al. (2021) utilizando também a *Google Maps API* apresentaram uma metodologia para traçar um perfil de velocidade média para cada região postal de uma cidade a fim de auxiliar tomadas de decisão quanto a regiões que carecem de ações para melhorar sua mobilidade urbana.

Para o escopo deste trabalho, a ferramenta produzida processará as informações coletadas em uma cadeia de Markov, com suas probabilidades de transição devidamente populadas, facilitando que trabalhos de *Road Network Design Problem* como de Salman Sinan; Alaswad (2018) possam ser mais facilmente executados em qualquer região que tenha acesso à aplicações de cálculos de rota como o *Google Maps API* e aplicações de informações de geoprocessamento como o *Openstreetmap API*(OSM), ambas ferramentas que serão utilizadas para o desenvolvimento da aplicação deste estudo.

1.1 Objetivos

Esta seção visa apresentar os objetivos propostos pelo trabalho que será realizado.

1.1.1 Objetivo Geral

Desenvolver uma aplicação capaz de estimar a densidade de tráfego de uma malha com base em dados do OSM e *Google Maps* utilizando como representação da malha uma cadeia de Markov.

1.1.2 Objetivos Específicos

Para atingir o objetivo geral, pretende-se cumprir os seguintes objetivos específicos:

- Levantar equacionamentos necessários para correlação entre a métrica de densidade e velocidade média de tráfego na literatura;
- Determinar quais informações da malha viária são necessárias para a ferramenta proposta;

- Desenvolver a ferramenta de estimativa de cadeia de Markov representativa e de densidade de tráfego com base em dados de tempo estimado de viagem, incluindo todos os módulos necessários para extração e tratamento de dados;
- Definir uma região de análise para experimentos;
- Aplicar experimentos e analisar os resultados da estimativa.

1.2 Definição do problema computacional

O problema a ser endereçado neste trabalho deve, a partir de dados públicos e gratuitos referentes a disposição de vias, velocidade máxima e extensão, assim como dados não gratuitos de estimativa de tempo de viagem, construir uma cadeia de Markov que represente uma região de malha viária real. A cadeia de Markov será representada por uma matriz que contém as probabilidades de transição por iteração entre membros da cadeia, sendo cada elemento a_{ij} a probabilidade de transição saindo do trecho i para o trecho j.

1.2.1 Metodologia

Este trabalho se trata de uma pesquisa quantitativa e exploratória, pois pretende-se implementar modelos pré-existentes em aplicação análoga a outras também encontradas na literatura, expandido o tipo de resultado obtido pelas mesmas e avaliando a qualidade dos resultados experimentais contra informações factuais.

Para desenvolver a pesquisa proposta pretende-se realizar as seguintes etapas:

Inicialmente será realizada uma pesquisa teórica para fornecer fundamentação e garantir a validade do trabalho proposto. Para tal pesquisa serão consultados artigos e livros em repositórios como o IEEE Xplore (ieeexplore.ieee.org) e Google Scholar (scholar.google.com).

Após este levantamento da literatura pretende-se especificar o modelo utilizado para expandir o processamento de dados feitos pela aplicação, bem como os valores de quaisquer parâmetros necessários, de forma que ao fim desta etapa estejam identificados todos os métodos e fórmulas necessárias para o trabalho.

Com a etapa de especificação da modelagem realizada segue a etapa de desenvolvimento, que consiste na implementação das ferramentas necessárias e da própria aplicação em que o trabalho é centrado. Para tal o código-fonte será escrito usando as linguagens de programação Rust e Python. Paralelamente a esta etapa será determinada a região de malha viária em que serão conduzidos os testes de validação.

Por fim, a última etapa consiste na realização de experimentos com a aplicação desenvolvida e a análise dos resultados, pautada em definir a qualidade das estimativas obtidas.

1.2.2 Estrutura do Texto

Este trabalho está estruturado em seis capítulos.

O Capítulo 1, Introdução, apresentou um apanhado geral quanto a objetivos, motivações e metodologia de desenvolvimento do trabalho.

No Capítulo 2, Fundamentação Teórica, são apresentados conceitos fundamentais para a construção deste trabalho, tratando de definições a cerca de cadeias de Markov e variáveis macroscópicas de tráfego.

No Capítulo 3, Trabalhos Relacionados, trabalhos com temáticas ou assuntos relacionados com o objeto central deste trabalho são analisados e discutidos.

No Capítulo 4, Desenvolvimento, são apresentadas as ferramentas utilizadas para a confecção da aplicação deste trabalho e descrito todo o processo de processamento de dados implementado na aplicação.

No Capítulo 5, Análise de Resultados, são apresentados os experimentos utilizados para validar o funcionamento da ferramenta e as análises realizadas sob os mesmos.

No Capítulo 6, são apresentadas as Conclusões e trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo são apresentados os principais conceitos utilizados no decorrer deste trabalho com suas respectivas definições fundamentadas na literatura da área de estudo.

2.1 Cadeias de Markov

Para Salman Sinan; Alaswad (2018), uma cadeia de Markov é um processo estocástico onde se observa a propriedade de Markov, em que a probabilidade de uma variável se encontrar em determinado estado depende somente do seu estado anterior. Este processo pode ser utilizado em uma variedade de áreas diferentes, como por exemplo na modelagem de malhas viárias.

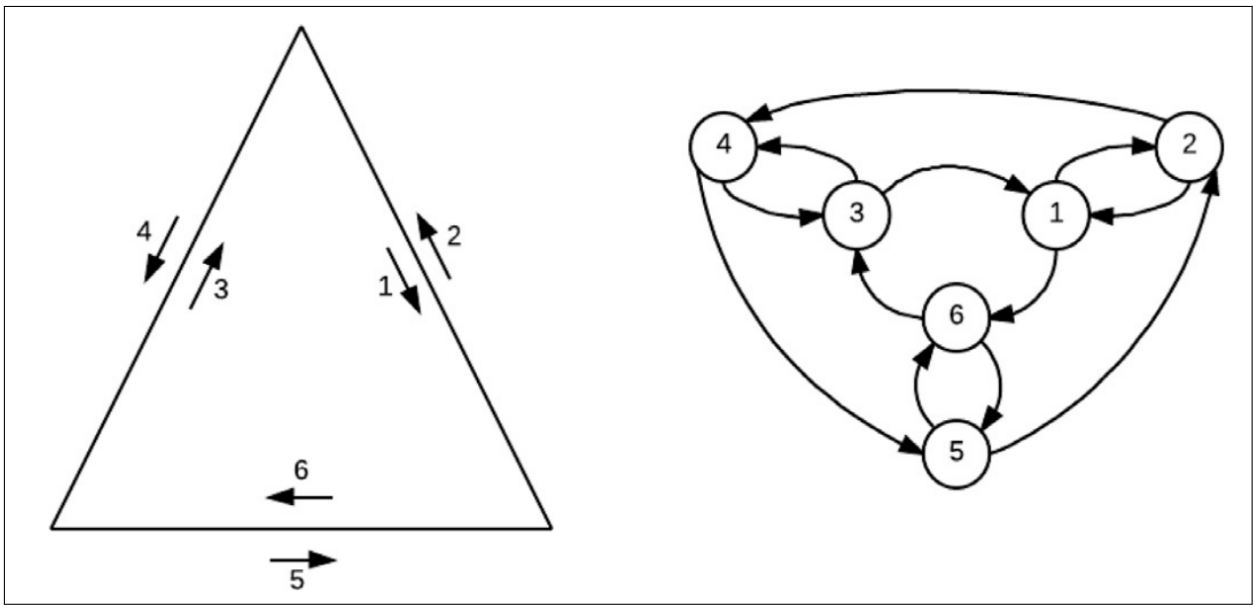

Figura 1 – Exemplo de malha com três vias representada como uma cadeia de Markov (SALMAN SINAN; ALASWAD, 2018)

A Figura 1 representa como uma malha viária pode ser representada por uma cadeia de Markov na forma de um grafo direcionado, com os nodos representando uma orientação de movimentação de um veículo na malha e as arestas a transição para outra direção ou via.

Sempre que uma cadeia de Markov possuir um número finito de estados ela poderá ser representada por um grafo dirigido, onde os nodos do grafo representam o estado na cadeia e as arestas representam a transição entre os estados. Além disso, pode ser definida uma matriz de transição $\mathbf{P}$ que para cada par de estados i e j o elemento $p_{i,j}$ contém a

probabilidade P de chegar no estado j no passo seguinte do processo quando o estado atual for i, ou seja:

$$\mathbf{P} = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,n} \end{pmatrix}, p_{i,j} = P(X_{k+1} = j | X_k = i) \tag{2.1}$$

2.1.1 Propriedades de cadeias de Markov

Nesta seção são apresentadas algumas propriedades e definições no contexto das cadeias de Markov, utilizando como referência as definições trazidas nos trabalhos de Guazzelli (1993), Grigoletti (2011), Garcia, Campos e Botta (2021) e Levin e Peres (2017).

2.1.1.1 Classes, periodicidade e redutibilidade

Segundo Levin e Peres (2017) dois estados se comunicam se ambos são alcançáveis entre si por pelo menos um caminho de transição com probabilidade maior que zero na cadeia. Logo, uma classe é definida por um conjunto de estados onde todos os estados daquele conjunto se comunicam entre si. Uma classe é dita fechada se cada estado pertencente à ela não possui nenhuma transição com probabilidade maior que zero para qualquer estado não pertencente à classe.

A periodicidade de um estado i é definido por um valor $k \geq 1$ tal que qualquer cadeia iniciando em i leva um valor múltiplo de k para chegar novamente em i. Se $k > 1$ o estado é dito periódico, se $k = 1$ o estado é dito aperiódico.

Uma cadeia de Markov é dita irredutível se é formada por apenas uma classe. Todos os estados de uma cadeia irredutível possuem a mesma periodicidade.

2.1.1.2 Transiência, recorrência e ergodicidade

Um estado de uma cadeia de Markov é dito transiente se existe uma cadeia de probabilidade positiva a partir dele em que nunca mais se retorna a esse mesmo estado. Caso todas as cadeias com probabilidade positiva a partir de um estado retornem ao mesmo estado eventualmente esse estado é dito recorrente.

Um estado é dito ergódico se é aperiódico e recorrente. Se todos os estados de uma classe são ergóticos a classe é dita ergótica. Para uma classe ser ergótica basta que sua periodicidade seja igual à 1, já que a definição de classe implica que todos os estados membros dela já possuam a propriedade de recorrência.

2.1.1.3 Aplicação das propriedades

Para o contexto de utilização em malhas viárias do presente trabalho, as propriedades de cadeias de Markov são conferidas e utilizadas apenas durante a estruturação da cadeia de Markov. Com uma cadeia de Markov da malha que seja formada por apenas uma classe é garantido que todos os cálculos necessário para probabilidades de transição e densidade de tráfego poderão ser realizados corretamente.

2.2 Variáveis Macroscópicas de Tráfego

Conforme o livro *Traffic Engineering* (ROESS; PRASSAS; MCSHANE, 2011) as características do fluxo de tráfego podem ser estudadas em duas escalas: a microscópica, que tem como foco os veículos de forma individual no tráfego, e a macroscópica, que descreve o comportamento do fluxo como um todo e que será de maior importância para este trabalho. Segundo Guessous et al. (2014), as três variáveis macroscópicas de tráfego são fluxo de tráfego, velocidade e densidade.

2.2.1 Fluxo de tráfego

Medida da quantidade de veículos que passam por um determinado ponto em um período de tempo (REILLY, 1997) e o valor máximo de fluxo de tráfego para uma determinada via é chamado de capacidade da via. Comumente este valor é obtido de forma empírica através da contagem de veículos.

2.2.2 Velocidade

A velocidade de um fluxo é representada como em qualquer outro contexto, utilizando uma medida de distância por tempo, porém para retratar um fluxo é necessário definir que valor utilizar já que cada veículo interno ao fluxo possui uma velocidade específica, sendo a velocidade média de travessia dos veículos do fluxo a de mais fácil cálculo e de maior relevância estatística (REILLY, 1997). Para determinar a velocidade média v para uma via de comprimento L é observado o tempo de travessia t_i de n veículos e então utilizando a seguinte fórmula:

$$v = \frac{L}{\frac{1}{n}\sum_{i=1}^{n} t_i} \tag{2.2}$$

Por fim de facilidade na leitura, ao longo do texto, sempre que a velocidade for mencionada sem maiores especificações é a velocidade média de travessia na via que se faz referência.

2.2.2.1 Velocidade de fluxo livre

Dentro do tópico de velocidade macroscópica do tráfego também é importante mencionar a velocidade de fluxo livre, que corresponde a velocidade em que os motoristas atravessam a via quando a quantidade de veículos é baixa o suficiente para que a distância entre diferentes veículos não influencie na velocidade adotada pelos motoristas. Apesar da velocidade de fluxo livre possua correlação com as leis de trânsito aplicáveis quanto ao limite de velocidade essa medida também é obtida por observação e varia com vários outros aspectos da via (SILVANO; KOUTSOPOULOS; FARAH, 2020).

2.2.3 Densidade

A densidade de um tráfego corresponde à quantidade de veículos existentes em determinada via em um instante de tempo, geralmente referido na medida de veículos por quilômetro (REILLY, 1997). Ela pode ser calculada usando das outras variáveis macroscópicas do tráfego, o fluxo de tráfego q e a velocidade v pela fórmula:

$$d = \frac{q}{v} \tag{2.3}$$

Essa medida é importante principalmente para identificar o comportamento geral do fluxo, sendo o fluxo livre associado a uma baixa densidade e o fluxo congestionado associado a uma alta densidade.

2.2.4 Diagrama Fundamental

No contexto de tráfego e trânsito o diagrama fundamental é um gráfico de fluxo de tráfego vs densidade (ou velocidade vs densidade).

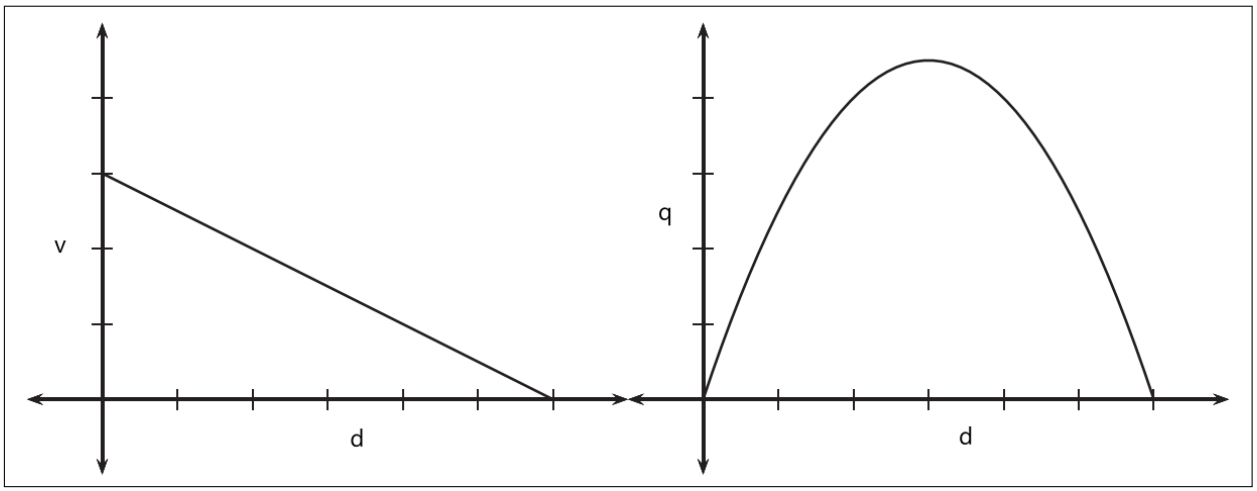

Figura 2 – Exemplos de representação do diagrama fundamental. À esquerda, velocidade vs densidade; À direita, fluxo de tráfego vs densidade. (KACHROO; SASTRY, 2016)

A Figura 2 apresenta uma noção do comportamento de cada um dos diagramas com o aumento da densidade na via, com a velocidade diminuindo conforme a densidade aumenta e o fluxo de tráfego aumentando até um ponto crítico e então reduzindo. Já na Figura 3 podemos observar um diagrama fundamental de fluxo vs densidade real contendo separações por níveis de serviço (ou *Level Of Service*, LOS), que são categorias que agrupam aspectos gerais de fluxo similares. Para o diagrama fundamental de fluxo vs densidade a identificação dos níveis de serviço é a principal importância, já o de velocidade vs densidade é geralmente utilizado para trabalhos teóricos (REILLY, 1997).

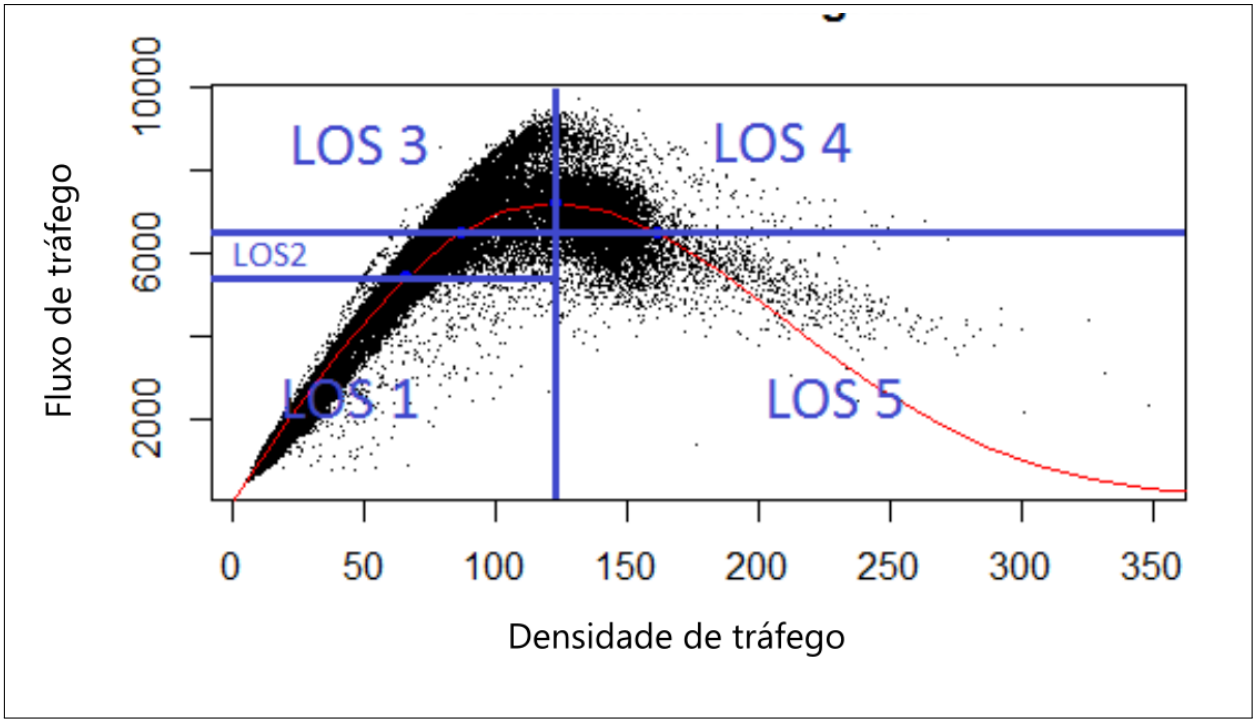

Figura 3 – Diagrama fundamental separado por níveis de serviço e com linha de tendência gerada por modelo exponencial. Obtido por observações em rodovia de Paris em 2006. (GUESSOUS et al., 2014)

O primeiro nível de serviço corresponde ao de fluxo livre, onde a velocidade é máxima. Neste regime o aumento de densidade só aumenta o fluxo de veículos. Na literatura (ZEGEER et al., 2008) é estimado que esse nível de serviço ocorre em uma densidade de veículos de até 7 veículos/km/faixa.

O segundo nível de serviço é de fluxo relativamente livre porém a densidade de veículos causa redução na velocidade adotada pelos motoristas.

O terceiro nível é de transição entre fluxo livre e fluxo congestionado. O limite deste nível é onde a via atinge a sua densidade crítica, onde a vazão de veículos é máxima. Esse valor de fluxo de tráfego máximo corresponde à capacidade da via.

O quarto nível de serviço é onde o fluxo já se mostra congestionado mas ainda com alta vazão.

O quinto nível é de congestionamento pesado, afetando fortemente o fluxo de tráfego e com comportamento altamente caótico.

3 Trabalhos Relacionados

Para a elaboração deste capítulo foi conduzida uma pesquisa por trabalhos focados em coleta de dados de tráfego, processamento de dados de tráfego e relação entre as diferentes variáveis de tráfego. Vários dos trabalhos encontrados constroem as referências e fundamentação para este trabalho. Dentre eles foram escolhidos três para discutir mais profundamente. Essa escolha foi feita pelo grau de similaridade com a pesquisa proposta neste trabalho, focando em publicações que envolvem o desenvolvimento de aplicações similares.

3.1 Alleviating Road Network Congestion: Traffic Pattern Optimization Using Markov Chain Traffic Assignment

Salman Sinan; Alaswad (2018) buscam neste trabalho realizar um experimento com métodos para reduzir os acúmulos de densidade de tráfego em vias urbanas, através de apenas alterações no sentido de uso das vias.

Os autores desenvolvem seu experimento representando a malha viária por meio de cadeias de Markov e utilizando métodos de algoritmos genéticos para chegar em uma solução ótima. Para construção da cadeia de Markov representativa são necessárias diversas informações sobre a região do experimento. Tais dados, como de velocidade, número de faixas e estrutura da malha viária, foram coletados com a *Openstreet Map*.

Após definir a região do experimento como a parte de ilha da cidade de Abu Dhabi, EAU, foram identificados e coletados os dados de 360 ruas inclusas na região. Também foi estimado que aproximadamente 10000 veículos populam as vias. Na solução proposta, as técnicas de algoritmo genético tentariam minimizar a densidade máxima em qualquer via da malha, considerando um indivíduo uma configuração da malha completa e o genoma do indivíduo a informação quanto a inversão das vias. Além de minimizar a densidade máxima, o algoritmo dos autores também priorizava indivíduos com menor número de inversões.

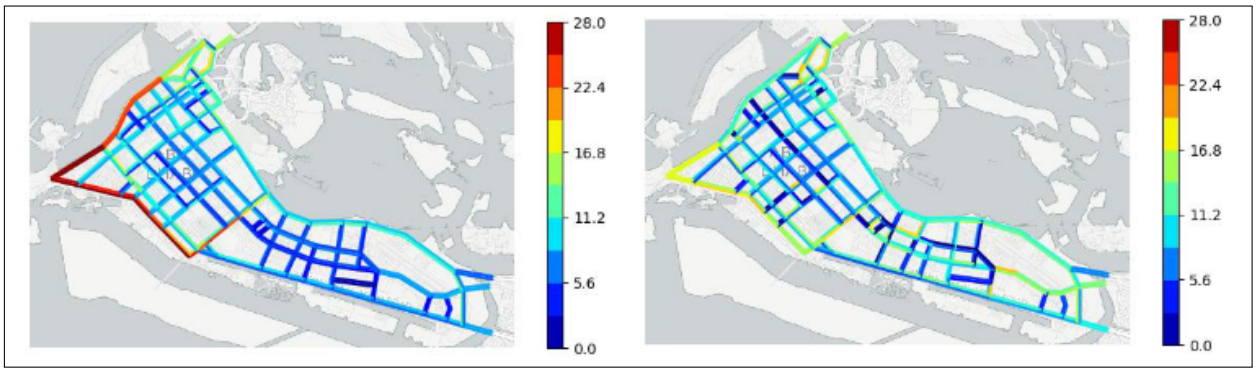

Figura 4 – Resultados de experimento. À esquerda, mapa de calor de densidade de tráfego com a disposição atual das vias; À direita, mapa de calor de densidade de tráfego com a disposição de vias sugerida (SALMAN SINAN; ALASWAD, 2018)

Na Figura 4 é apresentada uma comparação entre a densidade das vias no estado inicial da malha viária e após a aplicação do modelo proposto pelos autores. Com apenas 24 inversões a solução encontrada reduziria a densidade máxima encontra na região de 34,2 veículos/km/faixa para 19,8 veículos/km/faixa.

Com resultados interessantes, o trabalho de Salman Sinan; Alaswad (2018) demonstra um grande potencial a ser explorado no uso de cadeias de Markov para aplicações na área de mobilidade e planejamento de tráfego. Partindo desse potencial, o trabalho aqui proposto procura explorar formas obter de dados de tráfego com maior representatividade do fluxos reais, fornecendo fundação para que outros autores e pesquisadores sigam progredindo este campo de pesquisa.

3.2 Google Map Traffic Data Scraping and Mining

Neste trabalho Churi (2021) realiza uma pesquisa utilizando ferramentas de *data scrapping* para estudar o tempo de travessia de um trajeto específico em Mumbai, na Índia, com o objetivo de identificar regiões e períodos de maior congestionamento.

Para isso o autor segmentou o trajeto de análise em 30 trechos de 1km, marcando as coordenadas de início e fim de cada segmento. Com essa informação, a abordagem decidida foi por usar o *Selenium*, um *framework* focado em testes de aplicações em *web* que também é utilizado para acesso e coleta de dados em páginas da internet de forma automatizada. Utilizando as funcionalidades fornecidas pelo *Selenium* foram conduzidas consultas no *Google Maps* de forma automatizada para cada um dos trechos mapeados em ambos os sentidos de via e anotados o tempo de travessia indicado.

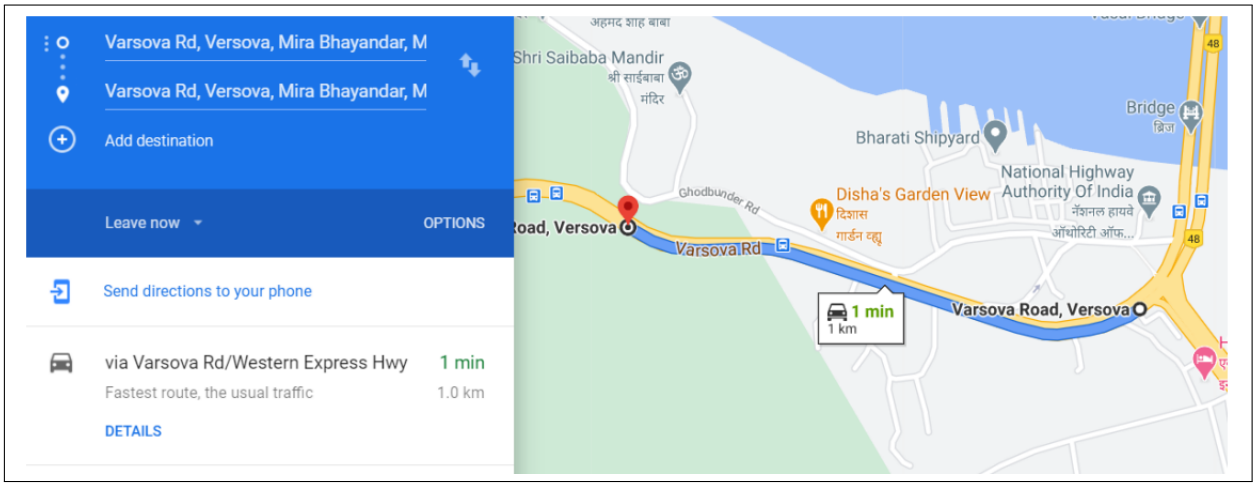

Figura 5 – Exemplo de consulta no *Google Maps* para um dos trechos selecionados (CHURI, 2021)

Após realizado o processo de coleta de dados estes foram organizados em uma tabela e importados em uma ferramenta de análise de dados. Por seguinte, o autor apresenta uma comparação dos tempos de travessia para cada período do dia e elabora algumas hipóteses da diferença desses valores, cruzando os resultados da coleta de dados com entendimentos empíricos da região de realização do estudo como obras, características da jornada de trabalho local e informações sobre a frota de veículos da cidade.

Interessante notar que entre a data de publicação da pesquisa de Churi (2021) para o presente uma *API* nova foi disponibilizada pela *Google* para a coleta dos dados de distância e tempo estimado de travessia, permitindo que o mesmo tipo de coleta de dados seja feita sem a utilização de um *framework* como o *Selenium*, reduzindo a complexidade e a quantidade de recursos necessárias nestes processos. Além disso, a forma de coleta de dados como no exemplo na Figura 5 apresenta uma granularidade de tempo de travessia na ordem dos minutos e de distância na centena de metros enquanto na *API* nova a granularidade desses valores passam a ser na ordem de segundos e de metros, respectivamente, melhorando a qualidade dos dados coletados.

Por fim, esta publicação demonstra que o processo de coleta de dados pode ser realizado de forma razoavelmente simples mesmo sem considerar os avanços nos serviços disponibilizados para consulta de tempos de viagem. Este processo de coleta é fundamental para o trabalho proposto e somente a partir dele é possível fazer todos os processamentos e transformações nos dados para alcançar os objetivos desta pesquisa.

3.3 An Open Source Tool to Extract Traffic Data from Google Maps: Limitations and Challenges

Logo na introdução da publicação de Mostafi e Elgazzar (2021) é mencionado como motivação do trabalho a falta de ferramentas de qualidade confiável para extração de dados de tráfego urbano para pesquisa científica, algo que seria então facilitado pelas novas *APIs* da *Google* citadas na seção anterior deste capítulo. Além disso, os autores mencionam a existência de ferramentas pagas disponibilizadas pela *Google* que poderiam ser utilizadas para tal fim mas buscavam maximizar a utilidade dos dados já disponibilizados gratuitamente pela plataforma.

Após definidas as motivações e problemas abordados no estudo são levantadas discussões sobre a confiabilidade dos dados de estimativa de tempo de viagem fornecidos pela ferramenta do *Google Maps*, algo crucial para o trabalho aqui proposto logo que toda a metodologia de desenvolvimento da aplicação final depende da qualidade de tais dados para ponderar o seu sucesso e alcance de seus objetivos. Dentre essas discussões são elencados alguns pontos sobre a plataforma do *Google Maps* e sobre os dados nela obtidos tais como:

- Os dados para realizar as estimativas são alimentados por *crowdsourcing* de telefones que utilizam o sistema da Android. Além disso, o uso desse tipo de dado fornece bons resultados para realizar tais estimativas de tráfego (TAHMASSEBY, 2015);
- Em estudos de caso diferentes em Hong Kong (HE; CHOW; ZHANG, 2019) e Paris (REZZOUQI et al., 2019) ambos demonstraram alta qualidade nas estimativas realizadas pela plataforma, sendo a acurácia de 95,8% para situações de trânsito fluido no segundo experimento;
- Em uma pesquisa (CAIZA et al., 2018) é demonstrada a capacidade de extrair informações suficientes para determinar regiões de congestionamento de tráfego, utilizando apenas das imagens de mapa fornecidas pela plataforma.

Em seguida são apresentadas as características da ferramenta proposta pelos autores, incluindo seu fluxo de execução e a interface que será fornecida para os usuários. O usuário teria como entrada sua localização, coordenadas ou selecionaria a posição diretamente na ferramenta do *Google Maps* para definir a origem do trajeto de interesse. Em seguida faria o mesmo tipo de entrada para definir o destino do trajeto de interesse, selecionaria entre opção de analisar por horário de saída ou de chegada no destino e informar o número de dias de observação.

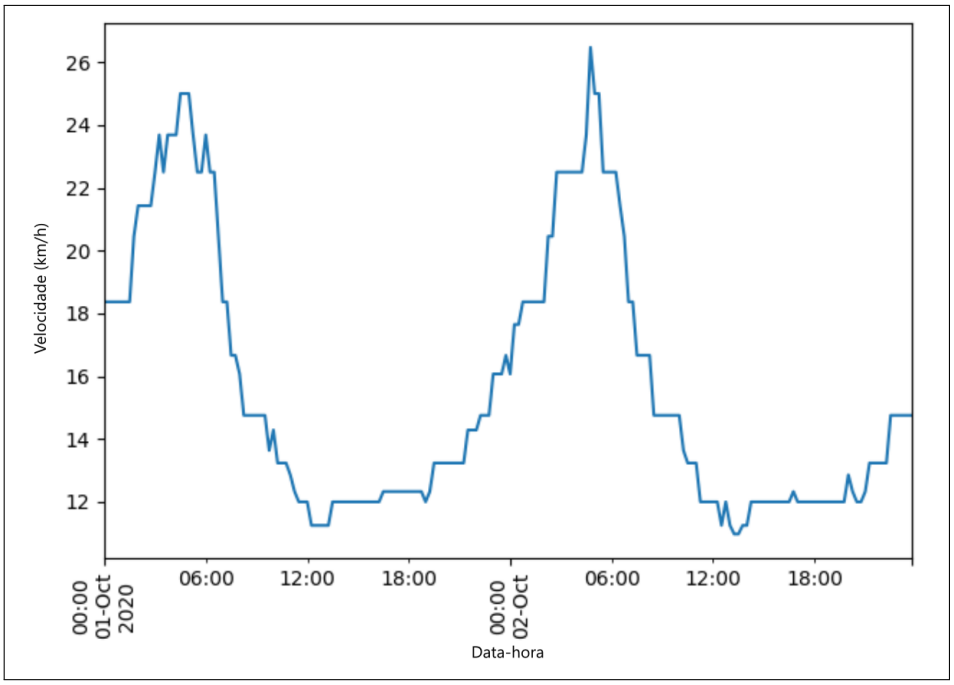

Figura 6 – Gráfico de velocidade observada vs tempo em experimento realizado com a ferramenta desenvolvida por Mostafi e Elgazzar (2021)
aplicada em um trecho na cidade de Toronto, Canadá, nos dias 01/10/20 e 02/10/20.

A aplicação então faria a coleta de informações de estimativa de tempo de viagem em intervalos de 15 em 15 minutos durante o número de dias requisitados e estruturaria um compilado dos resultados em um arquivo de formato CSV e também na forma de gráficos como na Figura 6, onde são apresentados valores de velocidade média estimada.

Discutido o funcionamento da ferramenta e demonstrado com exemplo reais os resultados que podem ser obtidos com o *software*, o foco dos autores se volta para a configuração da variável *callbackInterval*, um parâmetro da importante da ferramenta. Esta variável é uma configuração da ferramenta que determina o intervalo de tempo no laço principal da ferramenta para realizar consultas na aplicação do *Google Maps* em caso de falha. A importância da boa definição desse parâmetro para uma aplicação que usa recursos de *web* é que maximizando seu valor, ou seja, aumentando o intervalo de tempo entre cada tentativa é reduzida a chance de que uma falha temporária em um serviço invalide diversas tentativas de utilização do recurso *online* e consequentemente reduzindo o custo computacional da plataforma em que a ferramenta está sendo executada. Em contrapartida, este aumento no tempo entre tentativas também acarreta em maior tempo de execução médio da ferramenta. Além disso diversas chamadas consecutivas em serviços de *web*, principalmente gratuitos, podem levantar suspeitas de uso malicioso e/ou causar bloqueios por sobreuso do recurso por um determinado IP.

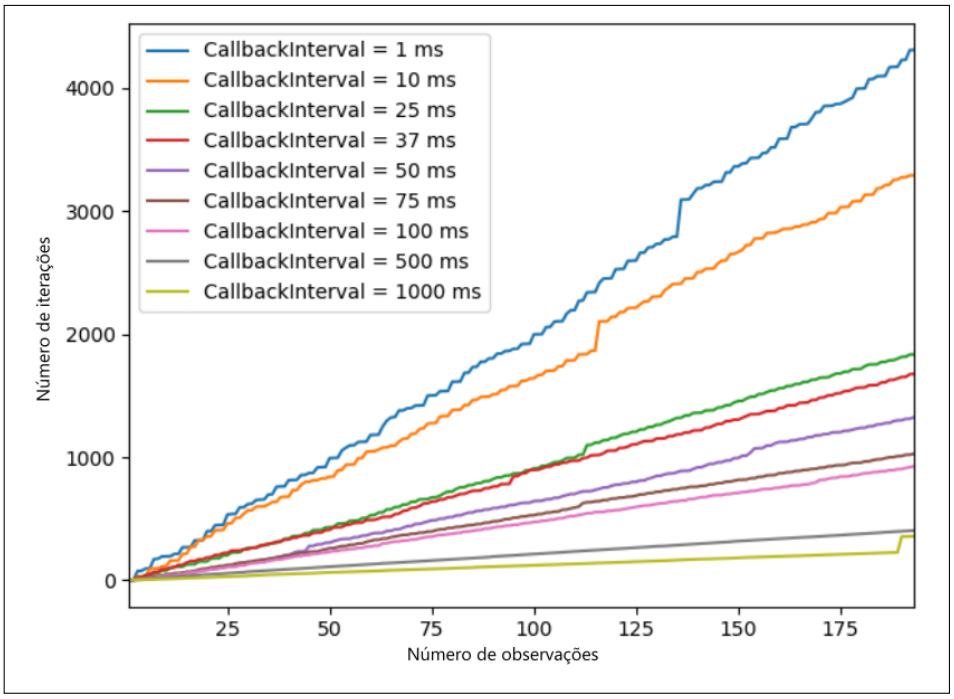

Figura 7 – Gráfico de número de iterações realizadas vs número de observações de tempo de travessia, colorido com diferentes cores para cada valor de *callbackInterval* testado (MOSTAFI; ELGAZZAR, 2021).

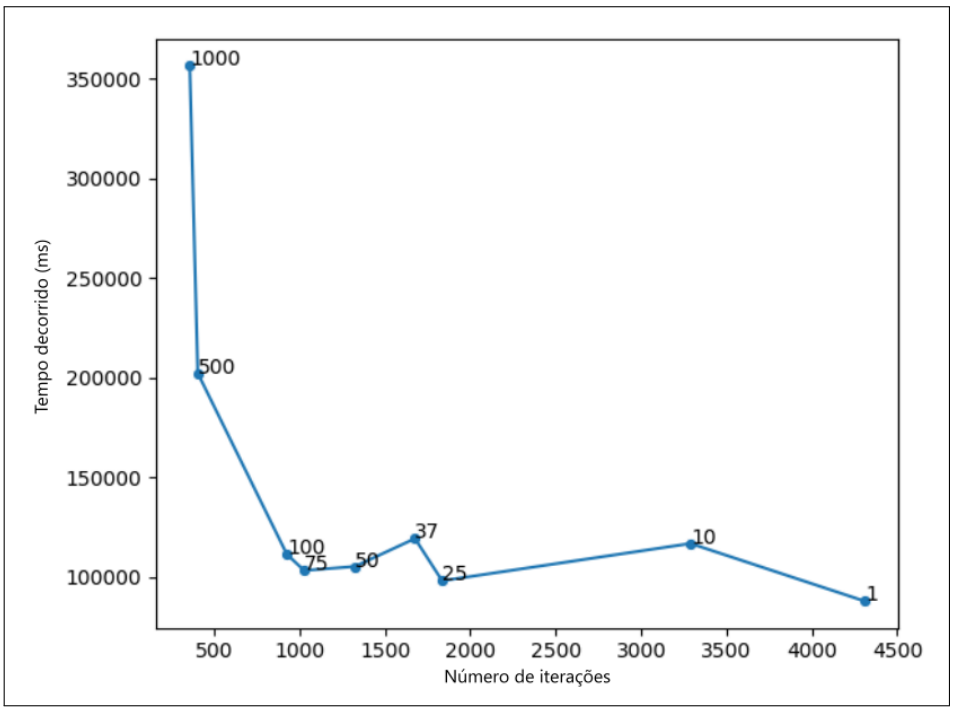

Figura 8 – Gráfico de tempo de processamento decorrido vs número de iterações realizadas, com pontos para cada valor de *callbackInterval* testado (MOSTAFI; ELGAZZAR, 2021).

Acima pode-se observar os resultados que foram obtidos pelos autores para seus testes

de *callbackInterval* para diversos valores. Na Figura 7 é notável que o número de iterações da ferramenta aumenta drasticamente com a redução do *callbackInterval*. Já na Figura 8 pode-se notar que a partir de certo ponto na redução do valor de *callbackInterval* não há significativa diminuição no tempo de execução. Com isso é elencado pelos autores que o valor ideal para a ferramenta desenvolvida pelos mesmos na pesquisa é o de 100ms, fornecendo um meio termo ótimo entre número de iterações e tempo de processamento.

Assim como no trabalho previamente discutido, o *An Open Source Tool to Extract Traffic Data from Google Maps: Limitations and Challenges* desenvolve uma ferramenta de caráter similar com a ferramenta que o presente trabalho se propõe a desenvolver e fornece análises interessantes acerca da mesma, que apesar de intrínsecas ao processo de desenvolvimento e do objetivo da pesquisa dos autores são cruciais para orientar o planejamento e execução do trabalho proposto.

3.4 Tabela comparativa

Nesta sessão está organizada uma sumarização em formato de tabela dos tópicos abordados por cada trabalho relacionado previamente citado neste capítulo, bem como a forma com que o presente trabalho se fundamenta e pretende expandir em alguns de seus pontos:

Citação	Principais elementos relacionados	Progresso do trabalho proposto
(SALMAN SINAN; ALASWAD, 2018)	Estimativa de cadeia de markov; Utilização de dados generalizados da via como velocidade para para estimativa de densidade dos fluxos em malha viária	Utilização de dados reais de velocidade para estimativa de densidade dos fluxos em malha viária
(CHURI, 2021)	Coleta de dados de expectativa de tempo de viagem utilizando *framework* de navegação web	Coleta de dados utilizando API oficial para fornecimento de expectativas de tempo de viagem e utilizando dos dados para estimar cadeias de Markov
(MOSTAFI; ELGAZZAR, 2021)	Processamento da expectativa do tempo de viagem com distância para obtenção de valores de velocidade média	Processamento da expectativa do tempo de viagem com distância para obtenção de uma cadeia de markov, com estimativas para densidade de tráfego

Tabela 1 – Tabela com principais elementos de trabalhos relacionados discutidos

4 Desenvolvimento

Neste capítulo são apresentados detalhes e especificações acerca deste trabalho, como as ferramentas utilizadas no desenvolvimento da aplicação proposta, forma de implementação e metologia de execução de testes, bem como seus resultados.

4.1 Ferramentas utilizadas

Sendo o objeto principal deste trabalho uma produção de *software*, algumas bibliotecas de uso livre foram utilizadas durante o processo de desenvolvimento da aplicação. Nesta seção, tais bibliotecas são brevemente apresentadas.

4.1.1 OSMnx

A OSMnx (BOEING, 2024) é uma biblioteca de Python que acessa dados da *OpenStreetMap* para facilitar o processo de aquisição, modelagem, análise e utilização de dados de malhas viárias urbanas.

Dentre as diversas funcionalidades oferecidas pela biblioteca poucas foram utilizadas neste trabalho, sendo elas:

- Funcionalidades para obter um grafo representativo de uma região, com nodos representando interseções e arestas as ruas. Neste grafo são inclusas informações sobre as vias, como nodo origem, nodo destino, número de faixas, velocidade máxima e tipo de rodovia. Também estão inclusos as coordenadas geográficas de cada nodo;
- Funcionalidades para obter um grafo representativo de uma região a um raio de distância de uma coordenada geográfica, com detalhamento de informações obtidas idênticos ao obtido por região.
- Funcionalidades para criar visualizações de grafos gerados com a biblioteca.

A aplicação utiliza essa biblioteca para obter as informações necessárias para construir a cadeia de Markov bem como para realizar estimativas de densidade baseadas na velocidade média das vias.

4.1.2 google_maps

A *google_maps* é uma biblioteca disponível para Rust que encapsula vários métodos de cliente da *Google Maps Platform*, além de incluir estruturas que facilitam o uso dos

serviços da *Google*. Neste trabalho a biblioteca é utilizada para facilitar a autenticação de cliente e realização de requisições de tempo de viagem dado através da *Directions API*.

Vale mencionar aqui que os serviços de rota fornecidos pela *Google Maps Platform* não são gratuitos, requerem a contratação dos serviços na plataforma e possuem custo por requisição feita atráves da *Directions API*, com valores de acordo com a tabela abaixo (GOOGLE, 2024):

Volume de uso mensal	Preço por requisição
0 - 100.000	RS$ 0.005
100.001 - 500.000	RS$ 0.004
500.001 - 500.000	Preço a definir sob consulta

Tabela 2 – Tabela de preços para o serviço do *Google Directions API*

A plataforma também disponibiliza US$ 300 de crédito no momento de contratação e US$ 200 de crédito mensalmente, sendo suficiente para cobrir quaisquer gastos provenientes dos testes durante o desenvolvimento deste trabalho.

4.2 Implementação

Nesta seção é apresentado brevemente a estrutura da implementação feita neste trabalho, focando em alguns detalhes de cálculos e algoritmos utilizados. A implementação da ferramenta proposta neste trabalho foi divida em duas partes bem definidas:

- *Scripts* em *Python* para obter dados da OSM e para gerar visualizações para análise;
- Aplicação em *Rust* para obter dados da *Directions API* e para processar juntamente os dados da OSM em cadeias de Markov.

4.2.1 Pré-processamento de dados

A partir do nome de uma região geográfica (como o nome de uma cidade ou bairro) a aplicação obtém, através da OSMnx, um grafo direcionado representante da região populado com as informações sobre cada uma das vias. Então são realizados alguns pré-processamentos dos dados: conferência de que todos os campos necessários para a aplicação estão presente; remoção de informações que não serão úteis para a aplicação; reorganização dos dados para facilitar processamentos posteriores. Este processo é feito para os elementos de arestas do grafo, com as informações das vias, bem como para os elementos de nodo, com as informações das interseções.

Após tratados e com garantia de qualidade suficiente para uso, os dados de arestas e nodos são salvos separadamente em formato JSON com uma estrutura bem definida para posterior leitura.

```
{
    "id": 973189203,
    "start": 90199946,
    "end": 447351081,
    "lanes": 4.0,
    "maxspeed": 80,
    "length": 199.767,
    "oneway": true,
    "highway": "trunk"
}
```

Listing 4.1 – Elemento de aresta pré-processado

```
{
    "id": 90199946,
    "latitude": -27.5994188,
    "longitude": -48.5548547
}
```

Listing 4.2 – Elemento de nodo pré-processado

4.2.2 Estruturação da cadeia de Markov

Após a fase de pré-processamento de dados é estruturado um novo grafo com todos os dados já disponíveis, no formato de uma cadeia de Markov. Neste novo grafo, assim como no trabalho de Salman Sinan; Alaswad (2018), os nodos representam as vias e as arestas uma transição entre vias, diferente dos dados obtidos anteriormente pela OSMnx onde as arestas que representam as vias.

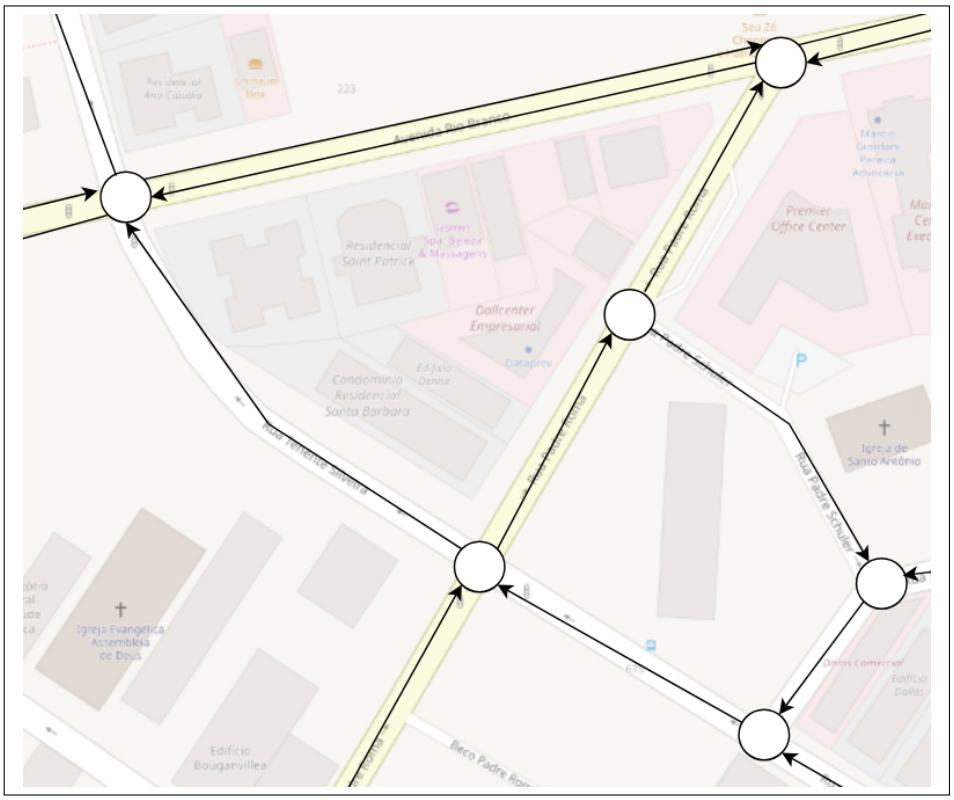

Figura 9 – Grafo representativo dos dados provenientes da OSMnx.

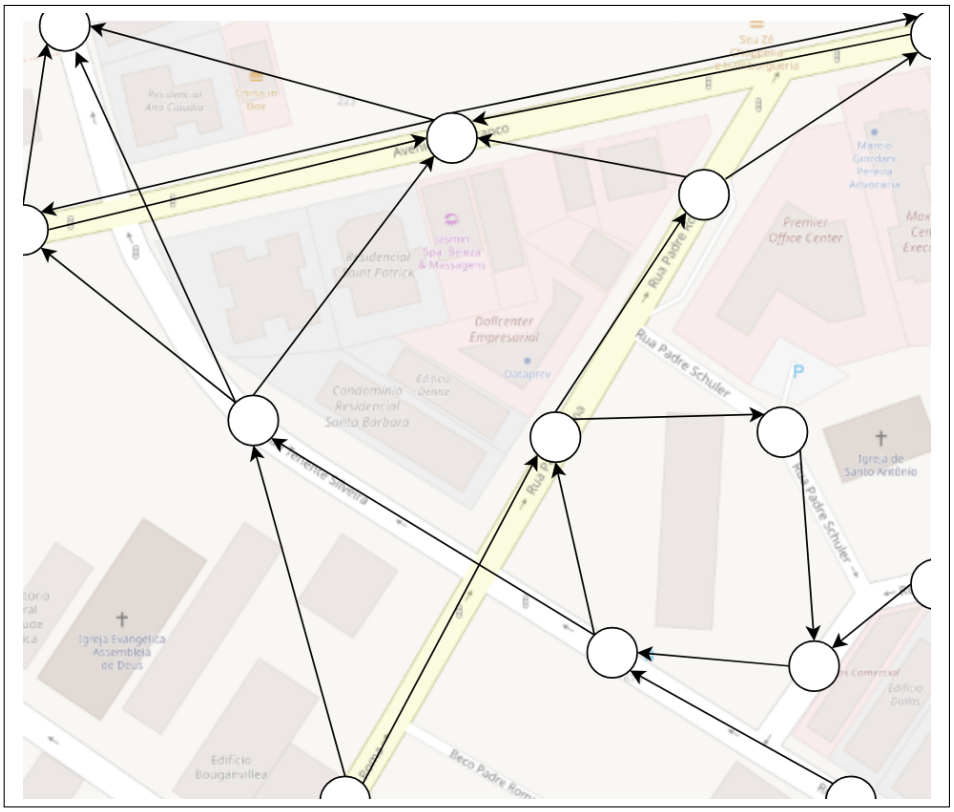

Figura 10 – Grafo representativo da cadeia de Markov.

Por conta dessa diferença as informações das vias passam a ser referentes aos nodos do grafo e as arestas passam a possuir apenas informações sobre movimentos possíveis a partir de uma via.

Neste passo também foi optado por alterar as estruturas de dados que representam o grafo. Ao invés de possuir uma lista para nodos e para arestas separados, as cadeias de Markov são estruturadas com apenas uma lista de nodos, cada qual contendo uma lista de transições possíveis. Essa abordagem elimina a necessidade de pesquisas pelas transições ou de manutenção de ponteiros para as transições.

É então atribuído um novo identificador único para cada nodo (iniciando em 0), utilizado os dados das interseções (antigos nodos) para determinar o início e o fim de cada via e populado a lista de transições possíveis para cada nodo. Até então, dados sobre a probabilidade de transição, densidade, velocidade média e tempo de travessia de cada via não foram estimados e são desconhecidos, como pode ser observado no exemplo 7.1, disponível no Capítulo 7.

4.2.3 Cálculo de velocidade média e tempo de travessia

Para estimar os dados faltantes na cadeia de Markov são primeiramente necessárias informações sobre velocidade média e tempo de travessia de cada via em cada via. A

aplicação desenvolvida possui duas opções de estimativa de tais informações dependendo da fonte de dados utilizada para tal.

Caso a base de estimativa sejam apenas os dados da OSMnx, como nos trabalhos de Salman Sinan; Alaswad (2018) e Santiago (2020), a aplicação considera que a velocidade média do usuário da via é sempre o valor de velocidade máxima da via. A partir dessa suposição é possível calcular o tempo de travessia:

$$tt = \frac{l}{v_e} \tag{4.1}$$

Sendo tt o tempo de travessia estimado em segundos, v_e a velocidade estimada em km/h e l o tamanho da via em quilômetros.

A aplicação também pode utilizar dados da *Google Maps* como base de estimativa dessas informações, sendo o ponto central da proposta deste trabalho. A principal motivação de utilizar tais dados para essa estimativa de velocidade média baseia-se em buscar informações com maior aderência com a realidade, em vista de que por influência de questões geométricas das vias e do volume de indivíduos na malha, a velocidade máxima da via nem sempre é a melhor suposição de velocidade média para os indivíduos.

Para isso é utilizado a coordenada de início e de fim de cada via como pontos de origem e destino, respectivamente, em consulta pelo *Directions API*, atráves das funcionalidades fornecidas pela biblioteca do *google_maps* discutida anteriormente, para obter a estimativa de tempo de travessia. Com o tempo de travessia é possível utilizar a equação 4.1 para determinar a velocidade média estimada.

4.2.4 Cálculo de transições

Para o cálculo das probabilidades de transição é primeiramente calculada a probabilidade de um indivíduo permanecer no estado em que se encontra.

Uma transição em uma cadeia de Markov representativa de uma malha viária corresponde à indivíduos saindo de um trecho de via e se posicionando em outro trecho. Para tal o indivíduo necessita percorrer completamente a via onde se encontra, relacionando então o tempo de travessia com os passos do processo da cadeia de Markov. Normalizando os tempos de travessia na via com base no menor tempo da malha, a probabilidade de um indivíduo permanecer no mesmo estado após um passo do processo é dada pela seguinte equação (SALMAN SINAN; ALASWAD, 2018):

$$p_{ii} = \frac{tt_n - 1}{tt_n} \tag{4.2}$$

Sendo p_{ii} a probabilidade do indivíduo permanecer no mesmo estado e tt_n o tempo de travessia normalizado.

Essa probabilidade é também chamada probabilidade auto transição, interpretada de modo que o indivíduo que realiza a auto transição permanece na mesma via e mesmo

sentido no próximo passo do processo. Para as demais transições é suposto que o indivíduo escolhe aleatoriamente qualquer uma das opções. Dessa forma, a probabilidade de transição dada por:

$$p_{ij} = \frac{1 - pii}{n_p - 1} \tag{4.3}$$

Sendo p_{ij} a probabilidade do indivíduo transitar do estado i para o estado j, p_{ii} a probabilidade do indivíduo permanecer no mesmo estado e n_p o número de transições possíveis (auto transição inclusa).

Com os dados de transição calculados, os dados são organizados em uma matriz de transição e salvos. Nesta matriz é utilizado o novo identificador atribuído para cada via durante a construção da estrutura da cadeia de Markov como índice, de modo que o elemento p_{ij} da matriz é a probabilidade de transição partindo da via de identificador i para a via de identificador j.

4.2.5 Cálculo de densidades

A densidade de veículos em cada via é calculada de forma análoga a de Salman Sinan; Alaswad (2018):

$$D_i = \sum_{\forall j | p_{ji} > 0} \frac{V p_{ji}}{l_i N_i} \tag{4.4}$$

Sendo D_i a densidade de veículos na via i em veículos/km/faixa, V o número total de veículos na malha, p_{ji} a probabilidade do indivíduo transitar do estado j para o estado i, l_i a extensão da via em quilômetros e N_i o número de faixas na via.

Durante esse cálculo o número de faixas N_i é dividido pela metade para vias que são de mão duplas, considerando que cada sentido da via pode utilizar metade da sua capacidade.

Neste momento também é necessário supor o número total de veículos na malha V. Para tal é considerado que todas as vias inicialmente possuem um número de veículos suficiente para se encontrarem ainda em fluxo livre, sendo então 7 veículos/km/faixa como previamente apresentado em na seção 2.2.4. Pode-se obter então o valor suposto de veículos que populam a cadeia de Markov pela seguinte equação.

$$V = \sum_{i=0}^{n} 7 N_i l_i \tag{4.5}$$

Essa suposição para o número de veículos é suficiente para garantir que a escala em que os valores de densidade são obtidos seja de fácil comparação com as densidades que definem a transição entre níveis de serviço das vias.

Por fim, são obtidos os resultados do processamento da ferramenta, seja utilizado os valores de velocidade máxima da OSM como velocidade média ou utilizando o tempo

de travessia estimado proveniente da *Directions API*. No Capítulo 7 são encontrados no Exemplo 7.2 e Exemplo 7.3 dados de nodo da cadeia de Markov resultante do processamento utilizando apenas os dados da OSM e de processamento utilizando dados da *Direction API*, respectivamente.

5 Análise de resultados

Neste capítulo são apresentados a metodologia de teste e análise da ferramenta desenvolvida, os resultados obtidos de seu uso e discutido sobre a qualidade dos resultados obtidos.

5.1 Objetivo dos experimentos

Para a guiar a realização dos experimentos e a análise dos resultados conferem os seguintes objetivos:

- Analisar se o tempo de processamento é adequado para a realização dos experimentos e uso da ferramenta;
- Analisar o custo financeiro de uso dos serviços da *Google Maps*;
- Analisar a capacidade da ferramenta de estimar cadeias de Markov para diferentes regiões;
- Analisar as diferenças entre os resultados utilizando a velocidade máxima obtida do *OpenStreetMap* para cálculo do tempo de travessia com os resultados utilizando o tempo de travessia obtido pela *Google Directions API*.

5.2 Experimentos

Para os experimentos compilou-se os códigos-fonte em Rust e a partir do binário resultante e dos *scripts* em Python, foram realizadas execuções da aplicação para posterior análise. O equipamento utilizado para os experimentos possui as seguintes configurações:

- Processador: Ryzen 5 4650g
- Memória RAM: 2x8GB DDR4
- Sistema Operacional: Arch Linux

Considerando que a aplicação desenvolvida utiliza serviços pagos de terceiros, a aplicação foi executada com regiões de porte adequado para que múltiplas execuções da aplicação não acarretasse em custos fora da cota gratuita fornecida pela plataforma da *Google Maps*.

Pela falta de dados quantitativos de cadeias de Markov representativa de malha viária ou de perfil de densidades de tráfego optou-se por uma análise majoritariamente qualitativa dos resultados, justapondo os resultados obtidos utilizando a velocidade média como a velocidade máxima obtida pelo OSM com os resultados obtidos através do cálculo de velocidade média com base no tempo de travessia obtido pela *Directions API*.

Para testes durante o desenvolvimento foi escolhido o bairro José Mendes, Florianópolis, com apenas 81 segmentos de via. Nos experimentos foram realizadas execuções aproximadamente as 9 horas e as 18 horas de uma segunda-feira com o Centro de Florianópolis, contendo 920 segmentos de via. Também foram realizadas experimentos aproximadamente as 17 horas de uma terça-feira com o bairro do Brás, São Paulo, contendo 656 segmentos de via.

5.3 Discussão de resultados

O primeiro teste realizado foi feito com o Centro de Florianópolis. Após estimada a densidade de tráfego para a malha com ambas as fontes de dados previamente descritas, foi obtido um mapa de calor com as densidades seguindo uma separação pelos níveis de serviço segundo a densidade de tráfego em cada trecho (ZEGEER et al., 2008).

Pode-se observar que existe semelhança entre os resultados obtidos por ambas as fontes. Contudo, a partir da visualização das figuras 11 e 12 é notável alguns trechos de via possuem fortes diferenças entre as duas fontes. Para facilitar a visualização e análise foi utilizado um mapa de calor onde a densidade obtida com os dados da OSM é subtraída da densidade encontrada com dados da *Google Maps*.

Na Figura 13, de acordo com a legenda na imagem, tons de azul correspondem a trechos onde a densidade estimada utilizando apenas os dados do OSM possuem valores maiores que os encontrados utilizando dados da *Google Maps*. Tons em vermelho correspondem a trechos onde o valor de densidade estimada utilizando dados da *Google Maps* foram maiores que os valores obtidos utilizando apenas os dados da OSM. Por exemplo, se um trecho estimasse uma densidade de 6 veículos/km/faixa utilizando apenas o OSM e estimasse 28 veículos/km/faixa utilizando dados da *Google Maps*, o trecho estaria colorido com o tom mais vivo de vermelho.

Ao analisar a Figura 13 não é possível identificar padrão onde a OSM possui leituras de densidade maiores. Pelas características do cálculo de densidades nesse tipo de representação de cadeias de Markov (Equações 4.2, 4.3 e 4.4) e levando em consideração que vias coletoras tendem a possuir velocidades de tráfego maiores que arteriais (REILLY, 1997), algumas dessas vias de caráter mais coletor podem ter leituras de densidade menores justamente pelas vias arteriais conectadas a ela possuírem velocidade média menor. Como hipótese, a estimativa fornecida atráves do uso da *Google Maps* representa melhor

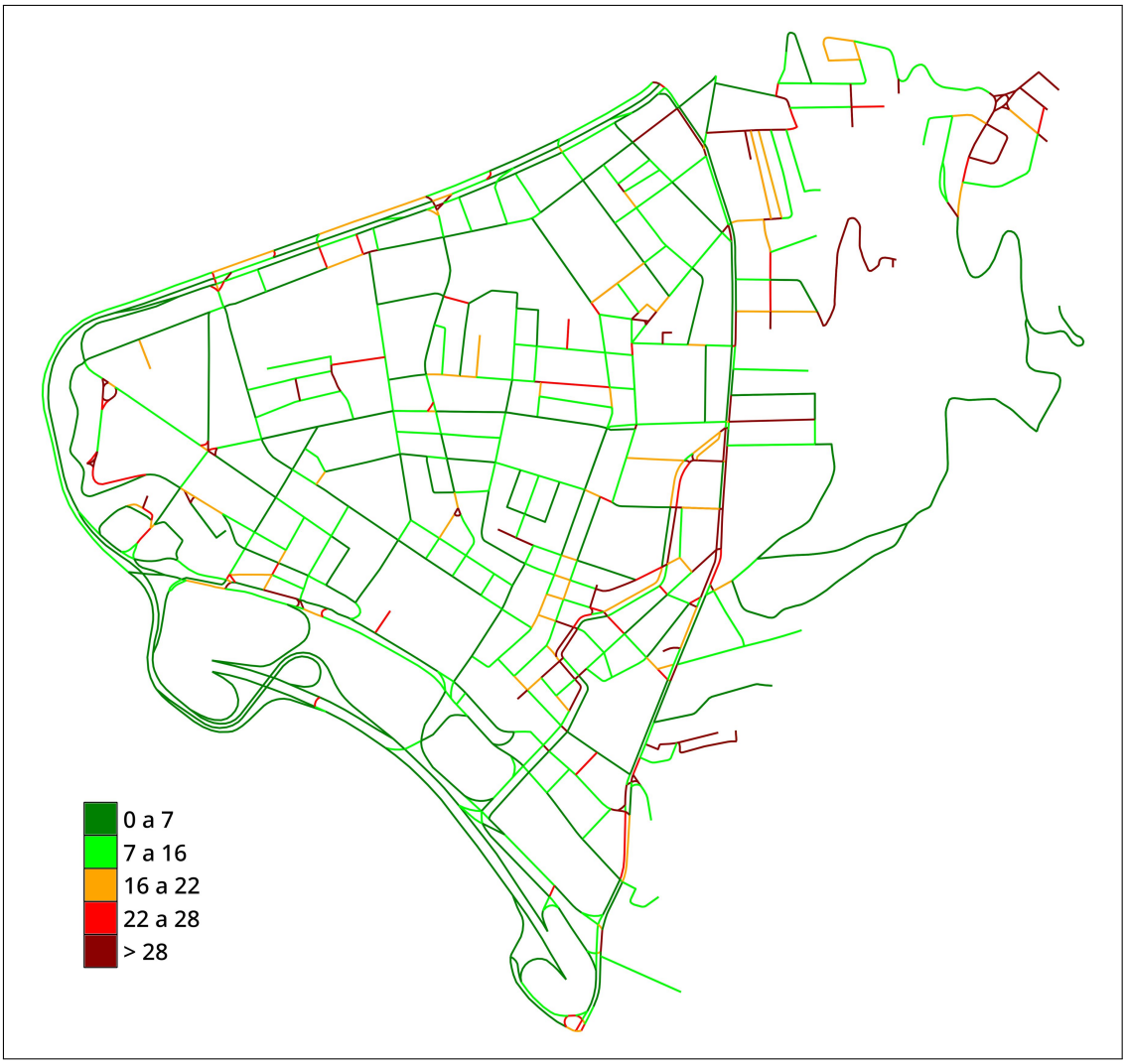

Figura 11 – Mapa de calor de densidade estimada baseado em dados da OSM. Legenda em veículos/km/faixa

essa diferença em velocidade de uso da via, uma vez que a OSM considera que os usuários da rede utilizariam vias arteriais no limite de velocidade permitido.

Já os trechos onde os dados da *Google Maps* levam a acumular densidade recorrentemente se encontra em trechos pequenos de cruzamentos. Supondo que os resultados da *Google Maps* representam melhor a realidade que os da OSM, uma hipótese que pode ser levantada para tal efeito nos cruzamentos é que por conta de cruzamento exigirem dos indivíduos redução de velocidade, seja por conta de sinalizações ou para realizar curvas, os dados de *crowdsourcing* acumulam dados notavelmente menores de velocidade do que o máximo permitido na via.

Experimentos realizados com o bairro do Brás, São Paulo, reforçam a confiança na capacidade de ser utilizada com diferentes regiões de análise diferentes. Vale citar que tal experimento não trouxe resultados significativos para análises mais profundas ou diferentes das previamente realizadas no experimento com o bairro do Centro, Florianópolis. Na

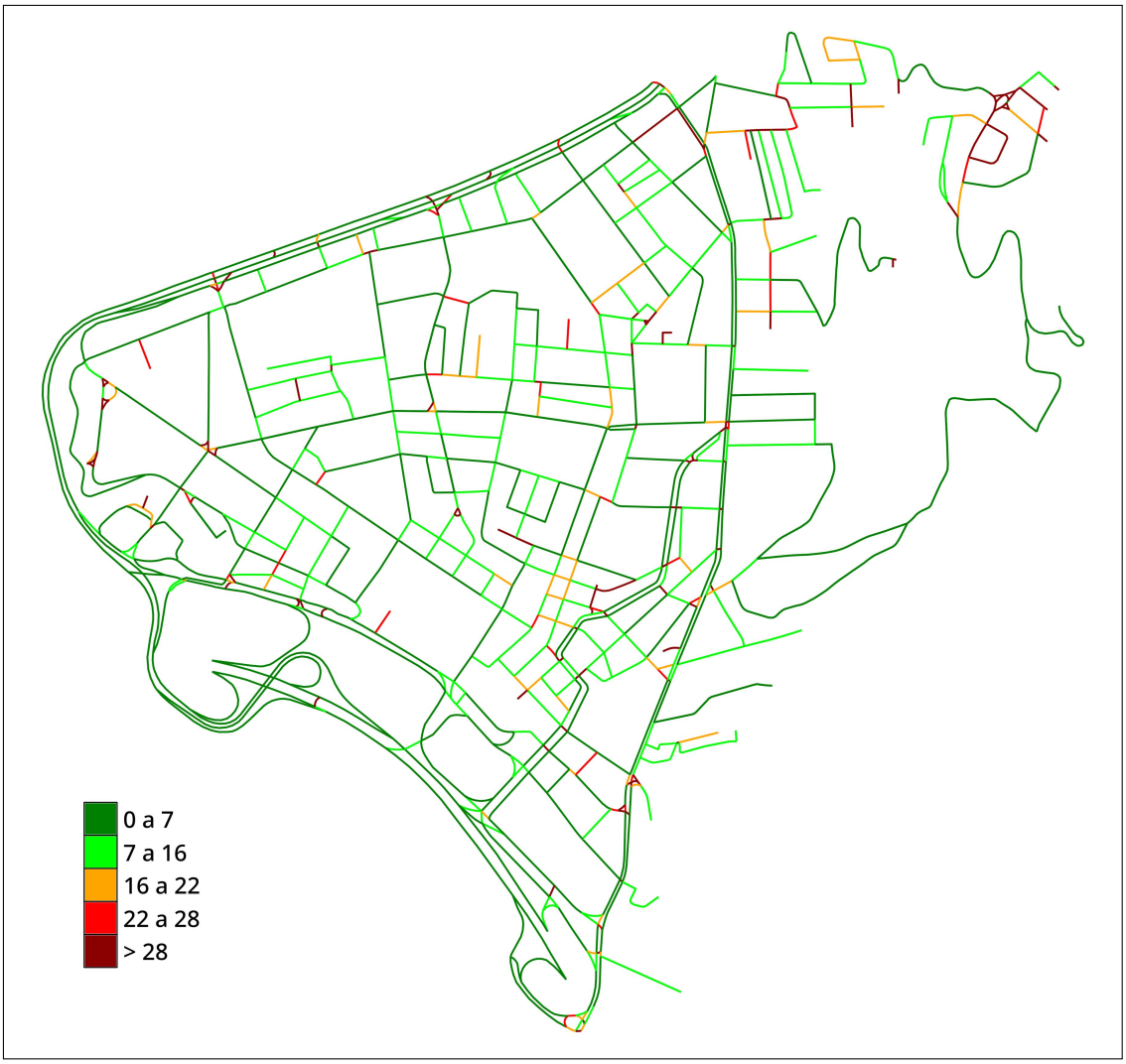

Figura 12 – Mapa de calor de densidade estimada baseado em dados da *Google Maps*. Legenda em veículos/km/faixa

Figura 14 é possível observar os resultados do experimento.

Considerando que os serviços da *Google Maps* fornecem dados baseado no tráfego em tempo real foram feitas tentativas de encontrar variações no comportamento da malha viária em diferentes horários. Sendo a Figura 12 relativa a uma coleta de dados realizada em torno das 9 horas de uma terça-feira, também foram executados testes em torno das 18 horas do mesmo dia.

Visualmente a figura obtida na leitura das 9 horas é idêntica à Figura 12. Apesar disso, leituras mais profundas nos dados obtidos mostram que em fato são resultados diferentes, como por exemplo, para a R. Antônio Pereira Oliveira Neto foi obtido uma velocidade média de 65.45 km/h na coleta das 18 horas e 55.38 km/h na coleta das 9 horas.

Leituras posteriores de documentações acerca da *Directions API* levaram ao entendimento de que o serviço utiliza de um tratamento de dados históricos pouco refinado, a fim de tornar o tempo de resposta da ferramenta mais ágil. A própria empresa também apre-

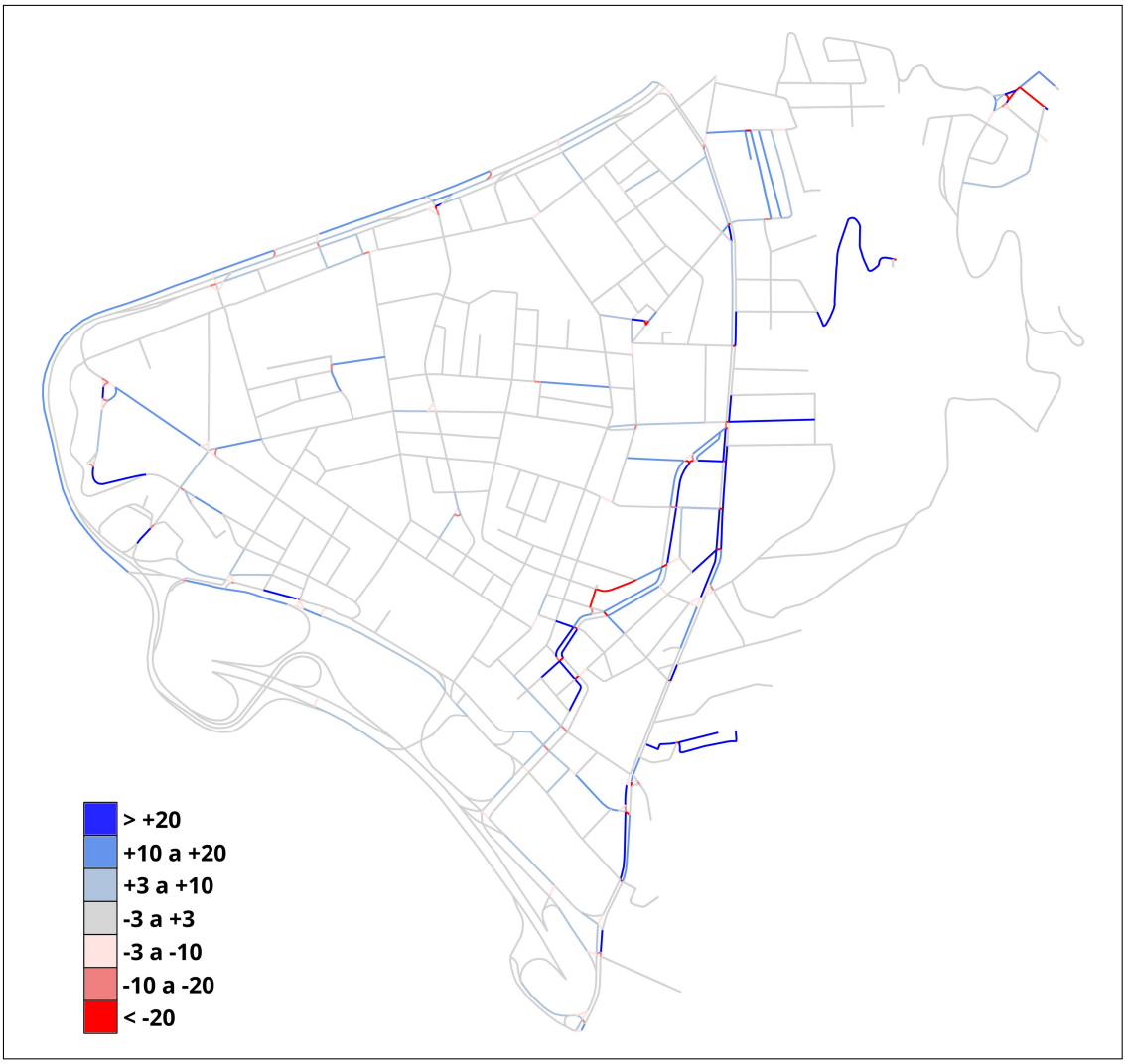

Figura 13 – Diferença de densidades entre resultados da OSM vs *Google Maps*

senta um outro serviço, *Routes API*, que alegadamente possui estimativas mais precisas de tempo de viagem estimada baseado em condições de tráfego em tempo real.

Por fim, sobre o custo financeiro do uso do serviço da *Directions API* neste trabalho, todos os gastos foram coberto pro créditos gratuitos oferecidos pela plataforma, sendo um total de R$ 246,31 de créditos utilizados durante testes na fase de desenvolvimento e R$ 431,12 de créditos utilizados durante a execução dos experimentos.

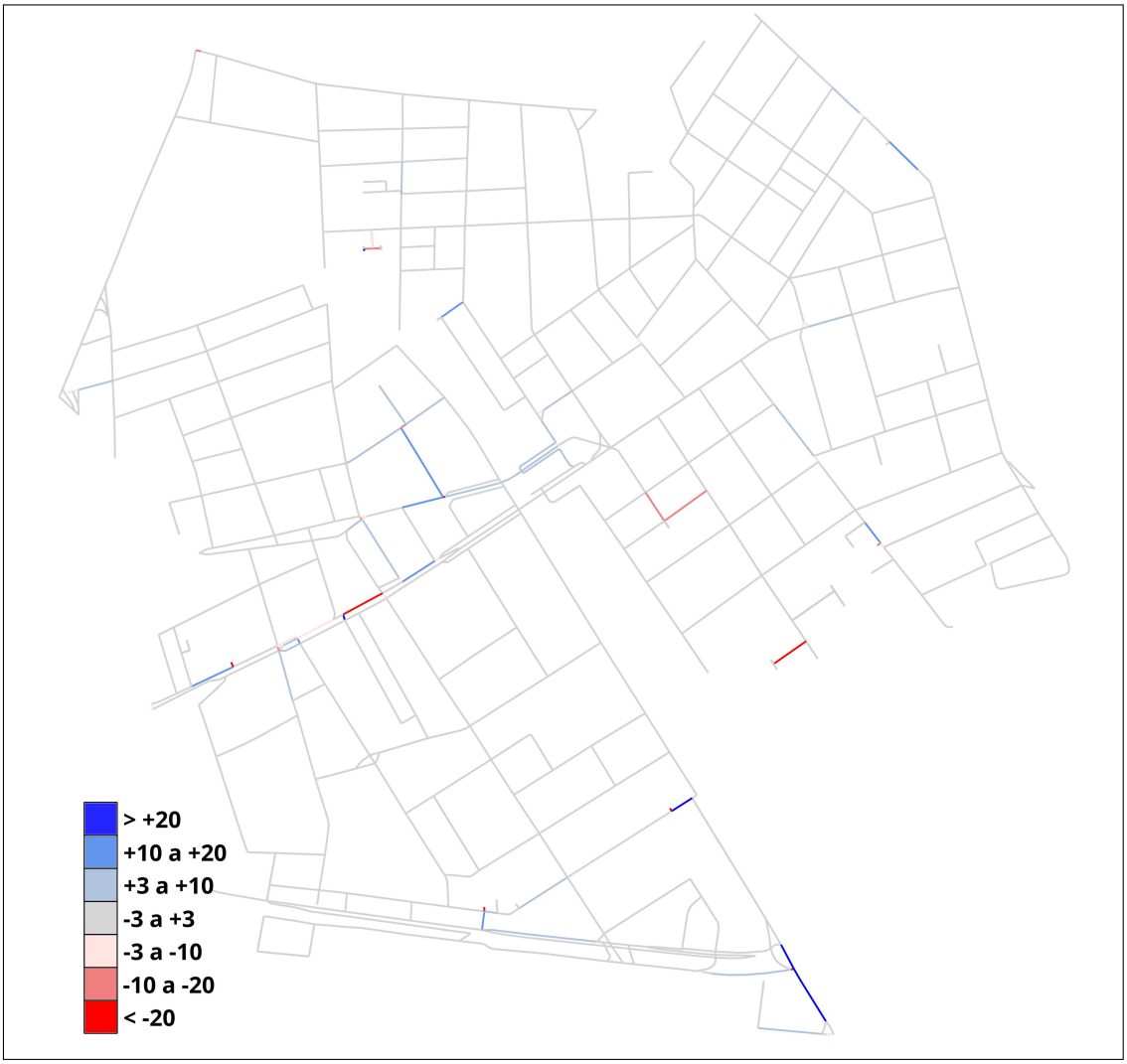

Figura 14 – Diferença de densidades entre resultados da OSM vs *Google Maps* para experimento realizado com o bairro do Brás, São Paulo, horário aproximado às 17 horas de uma terça-feira

6 Conclusões

Buscando explorar melhores alternativas para a estimativa de densidade de tráfego, o trabalho presente explora as possibilidades trazidas por serviços bem consolidados de estimativa de tempo de viagem como o *Google Maps*. Com resultados que representem melhor a realidade da malha viária em estudo, pesquisas e projetos de mobilidade urbana beneficiam-se ao partir de tais dados para seus experimentos.

A partir das análises realizadas pode-se concluir que a implementação da ferramenta é bem sucedida em processar dados em estimativas úteis para outros trabalhos que usam de cadeias de Markov para representação de tráfego. Apesar disso, fica evidente também de que necessita-se de mais dados para definir ou quantificar a melhoria proporcionada pelo uso de dados além dos provenientes do *OpenStreetMap* nesse tipo de estimativa.

Finalmente, quanto aos objetivos do trabalho proposto, considerando o objetivo geral apresentado no Capítulo 1, pode-se afirmar que o mesmo foi atingido com a execução do trabalho, ilustrado pelos códigos-fonte que compõe a aplicação e pelos resultados experimentais previamente apresentados. Quanto aos objetivos específicos apresentados no Capítulo 1:

- Nos Capítulos 3 e 4, todos os equacionamentos necessários para correlação entre a métrica de densidade de tráfego e velocidade média foram apresentadas, incluindo modificações necessárias no contexto do uso em cadeias de Markov;
- Com os equacionamentos devidamente apresentados, foi possível determinar quais informações da malha viária são necessárias para o uso da aplicação. A partir disso, foi garantido que as informações coletadas pela *OSMnx* e *google_maps* seriam suficientes para realizar os cálculos necessários para a aplicação;
- No Capítulo 4 também é apresentado o desenvolvimento da aplicação, explicando seu fluxo de processamento para estimar tanto a cadeia de Markov representativa e quanto os valores de densidade de tráfego;
- Define-se no Capítulo 5 em quais regiões os experimentos serão realizados, bem como a metodologia de análise dos resultados obtidos;
- Também no Capítulo 5 os resultados provenientes da execução dos experimentos são apresentados juntamente com as análises feitas sob os mesmos.

6.1 Trabalhos futuros

Considerando o trabalho desenvolvido, abaixo são listados alguns dos possíveis tópicos para trabalhos futuros:

- Utilização da *Routes API* ao invés da *Directions API* para requisições de tempo de viagem;
- Implementação de variação de número de veículos na malha de estudo;
- Implementação de novas metodologias de análise e validação de resultados;
- Revisitar trabalhos que utilizam técnicas similares de cadeias de Markov a partir do OSM e comparar resultados utilizando os dados da *Google Maps*;
- Implementação de conexão com *softwares* GIS.

Referências Bibliográficas

BOEING, G. Modeling and analyzing urban networks and amenities with osmnx. 2024. 27

CAIZA, L. J. et al. Vtm: Vehicular traffic monitor via images processing of googlemaps. In: **Proceedings of the 15th ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks**. [S.l.: s.n.], 2018. p. 40–46. 23

CHURI, A. D. Google map traffic data scraping and mining. **no**, v. 4, p. 891–895, 2021. 6, 10, 21, 22, 26

GADDAM, H. K.; RAO, K. R. Speed-density functional relationship for heterogeneous traffic data: a statistical and theoretical investigation. **Journal of modern transportation**, Springer, v. 27, p. 61–74, 2019. 10

GALINDO ERNESTO PEREIRA E LIMA NETO, V. C. **A mobilidade urbana no Brasil: Percepções de sua população**. 2019. Disponível em: <https://www.econstor.eu/bitstream/10419/211420/1/1666102709.pdf>. 4, 5, 10

GARCIA, V. F.; CAMPOS, P. H. O.; BOTTA, V. Cadeias de markov: propriedades e aplicações. **Proceeding Series of the Brazilian Society of Computational and Applied Mathematics**, v. 8, n. 1, 2021. 15

GOOGLE. **Directions API Pricing**. 2024. Disponível em: <https://developers.google.com/maps/documentation/directions/usage-and-billing?hl=pt-br#pricing-for-product>. 28

GRIGOLETTI, P. S. Cadeias de markov. **Recuperado em**, v. 19, n. 10, p. 2014, 2011. 15

GUAZZELLI, M. R. Teoria e prática sobre as cadeias de markov. **Revista Ambiente**, v. 7, n. 1, p. 45–51, 1993. 15

GUESSOUS, Y. et al. Estimating travel time distribution under different traffic conditions. **Transportation Research Procedia**, Elsevier, v. 3, p. 339–348, 2014. 6, 16, 18

HE, Z.; CHOW, C.-Y.; ZHANG, J.-D. A comparative analysis of journey time from google maps and intelligent transport system in hong kong. In: IEEE. **2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)**. [S.l.], 2019. p. 2610–2617. 23

JUNIOR, M. R.; MEDRANO, R. A.; CRUVINEL, K. O uso de sinais wi-fi para estimação de pares origem destino de usuários do transporte público em ônibus. In: **XX Congreso Latinoamericano de Transporte Público y Urbano, At Medellín, Colombia**. [S.l.: s.n.], 2018. 10

KACHROO, P.; SASTRY, S. Traffic assignment using a density-based travel-time function for intelligent transportation systems. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 17, n. 5, p. 1438–1447, 2016. 6, 17

LEVIN, D. A.; PERES, Y. **Markov chains and mixing times**. [S.l.]: American Mathematical Soc., 2017. v. 107. 15

MOSTAFI, S.; ELGAZZAR, K. An open source tool to extract traffic data from google maps: Limitations and challenges. In: **2021 International Symposium on Networks, Computers and Communications (ISNCC)**. [S.l.: s.n.], 2021. p. 1–8. 6, 10, 23, 24, 25, 26

MUÑOZ-VILLAMIZAR, A. et al. Study of urban-traffic congestion based on google maps api: the case of boston. **IFAC-PapersOnLine**, Elsevier, v. 54, n. 1, p. 211–216, 2021. 11

PELLETIER, M.-P.; TRéPANIER, M.; MORENCY, C. Smart card data use in public transit: A literature review. **Transportation Research Part C: Emerging Technologies**, v. 19, n. 4, p. 557–568, 2011. ISSN 0968-090X. 10

REILLY, W. **Highway capacity manual 2000**. [S.l.: s.n.], 1997. 10, 16, 17, 18, 35

REZZOUQI, H. et al. Analyzing the accuracy of historical average for urban traffic forecasting using google maps. In: SPRINGER. **Intelligent Systems and Applications: Proceedings of the 2018 Intelligent Systems Conference (IntelliSys) Volume 1**. [S.l.], 2019. p. 1145–1156. 4, 5, 23

ROESS, R.; PRASSAS, E.; MCSHANE, W. **Traffic engineering**. 4th. ed. [S.l.]: Prentice Hall, 2011. Includes bibliographical references and index. 16

SALMAN SINAN; ALASWAD, S. Alleviating road network congestion: Traffic pattern optimization using markov chain traffic assignment. **Computers & Operations Research**, v. 99, 07 2018. 4, 5, 6, 11, 14, 20, 21, 26, 29, 31, 32

SANTIAGO, R. de. **Método Computacional para a Otimização do Projeto da Malha Viária de Florianópolis-SC**. [S.l.], 2020. 31

SENATRAN, S. N. de T. **Frota de Veículos - 2022**. 2022. Disponível em: <https://www.gov.br/transportes/pt-br/assuntos/transito/conteudo-Senatran/frota-de-veiculos-2022>. 10

SILVANO, A. P.; KOUTSOPOULOS, H. N.; FARAH, H. Free flow speed estimation: A probabilistic, latent approach. impact of speed limit changes and road characteristics. **Transportation Research Part A: Policy and Practice**, v. 138, p. 283–298, 2020. ISSN 0965-8564. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0965856420306066>. 17

TAHMASSEBY, S. Traffic data: Bluetooth sensors vs. crowdsourcing—a comparative study to calculate travel time reliability in calgary, alberta, canada. **Journal of Traffic and Transportation Engineering**, v. 3, n. 2, p. 63–79, 2015. 23

WANG, F.; XU, Y. Estimating o–d travel time matrix by google maps api: implementation, advantages, and implications. **Annals of GIS**, Taylor & Francis, v. 17, n. 4, p. 199–209, 2011. Disponível em: <https://doi.org/10.1080/19475683.2011.625977>. 11

WANG, H. et al. Stochastic modeling of the equilibrium speed–density relationship. **Journal of Advanced Transportation**, v. 47, n. 1, p. 126–150, 2013. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/atr.172>. 11

ZEGEER, J. D. et al. Default values for highway capacity and level-of-service analyses. **Transportation Research Record**, SAGE Publications Sage CA: Los Angeles, CA, v. 2071, n. 1, p. 35–43, 2008. 18, 35

7 Apêndices

7.1 Exemplos


```
{
  "id": 0,
  "id_osm": 973189203,
  "street_start": {
    "id": 90199946,
    "latitude": -27.5994188,
    "longitude": -48.5548547
  },
  "street_end": {
    "id": 447351081,
    "latitude": -27.600352,
    "longitude": -48.553126
  },
  "street_data": {
    "id": 973189203,
    "start": 90199946,
    "end": 447351081,
    "lanes": 4.0,
    "maxspeed": 80,
    "length": 199.767,
    "oneway": true,
    "highway": "trunk"
  },
"traffic_data": {
    "estimated_travel_time": {
      "Unknown": 0.0
    },
    "estimated_average_speed": {
      "Unknown": 0.0
    },
    "estimated_density": {
      "Unknown": 0.0
    }
  },
  "transitions": [
    {
      "id_to": 0,
      "probability": {
        "Unknown": 0.0
      }
```

```
    },
    {
      "id_to": 125,
      "probability": {
        "Unknown": 0.0
      }
    },
    {
      "id_to": 126,
      "probability": {
        "Unknown": 0.0
      }
    }
  ]
}
```

Listing 7.1 – Elemento de nodo de grafo de cadeia de Markov pré-estimativas

```
    {
      "id": 0,
      "id_osm": 973189203,
      "street_start": {
        "id": 90199946,
        "latitude": -27.5994188,
        "longitude": -48.5548547
      },
      "street_end": {
        "id": 447351081,
        "latitude": -27.600352,
        "longitude": -48.553126
      },
      "street_data": {
        "id": 973189203,
        "start": 90199946,
        "end": 447351081,
        "lanes": 4.0,
        "maxspeed": 80,
        "length": 199.767,
        "oneway": true,
        "highway": "trunk"
      },
      "traffic_data": {
        "estimated_travel_time": {
          "Known": 0.0024970875
        },
        "estimated_average_speed": {
          "Known": 80.0
        },
```

```
        "estimated_density": {
          "Known": 4.641404125167535
        }
      },
      "transitions": [
        {
          "id_to": 0,
          "probability": {
            "Known": 0.9410169990896537
          }
        },
        {
          "id_to": 125,
          "probability": {
            "Known": 0.029491500455173125
          }
        },
        {
          "id_to": 126,
          "probability": {
            "Known": 0.029491500455173125
          }
        }
      ]
    }
```

Listing 7.2 – Elemento de nodo de grafo de cadeia de Markov com estimativas baseadas na OSM

```
    {
      "id": 0,
      "id_osm": 973189203,
      "street_start": {
        "id": 90199946,
        "latitude": -27.5994188,
        "longitude": -48.5548547
      },
      "street_end": {
        "id": 447351081,
        "latitude": -27.600352,
        "longitude": -48.553126
      },
      "street_data": {
        "id": 973189203,
        "start": 90199946,
        "end": 447351081,
        "lanes": 4.0,
        "maxspeed": 80,
```

```
        "length": 199.767,
        "oneway": true,
        "highway": "trunk"
      },
      "traffic_data": {
        "estimated_travel_time": {
          "Known": 0.0030555555555555557
        },
        "estimated_average_speed": {
          "Known": 65.45454545454545
        },
        "estimated_density": {
          "Known": 1.258691749154272
        }
      },
      "transitions": [
        {
          "id_to": 0,
          "probability": {
            "Known": 0.999990909090909
          }
        },
        {
          "id_to": 125,
          "probability": {
            "Known": 4.545454545479277e-6
          }
        },
        {
          "id_to": 126,
          "probability": {
            "Known": 4.545454545479277e-6
          }
        }
      ]
    }
```

Listing 7.3 – Elemento de nodo de grafo de cadeia de Markov com estimativas baseadas na *Directions API*

7.2 Código-fonte

Na data de publicação deste trabalho, o projeto contendo todos os códigos-fonte deste trabalho se encontram no projetodcroque/geomarkover no repositório do Github sob licença de distribuição e uso GPL-3.0.

7.2.1 Python

7.2.1.1 ___init___.py

```
import osmnx as ox
import networkx as nx
import pathlib
import json
import matplotlib.pyplot as plt

import cli

default_needed_keys = ["osmid", "oneway", "lanes", "reversed", "maxspeed", "highway"]
result_dict_needed = {
    "osmid": str,
    "oneway": bool,
    "lanes": int,
    "reversed": bool,
    "maxspeed": float,
    "highway": str,
}

#para vias urbanas:
default_speeds = {
    "secundary": 40,
    "service": 30,
    "residential": 30,
}

# CLI interaction functions

def process_args() -> dict:
    request_info = {}
    if cli.args.image_generation:
        request_info["retrieval_type"] = "image"
        request_info["place"] = cli.args.place
        request_info["source"] = cli.args.source
    elif cli.args.place is not None:
        request_info["retrieval_type"] = "place"
        request_info["place"] = cli.args.place
    elif cli.args.latitude is not None and cli.args.longitude is not None and cli.args.radius is not None:
        request_info["retrieval_type"] = "coordinates"
        request_info["latitude"] = cli.args.latitude
        request_info["longitude"] = cli.args.longitude
        request_info["radius"] = cli.args.radius
```

```
42      else:
43          request_info["retrieval_type"] = "noop"
44          return request_info
45
46      request_info["path"] = cli.args.file_path if cli.args.file_path is not
    None else "output"
47      request_info["name"] = cli.args.name if cli.args.name is not None else
    "default"
48      request_info["save_results"] = not cli.args.dry_run
49
50      return request_info
51
52  def process_request(request_info: dict) -> bool:
53      nw = None
54      match request_info["retrieval_type"]:
55          case "image":
56              match request_info["source"]:
57                  case "osm":
58                      print_image("osm", request_info["place"], request_info
    ["path"])
59                      return True
60                  case "gmaps":
61                      print_image("gmaps", request_info["place"],
    request_info["path"])
62                      return True
63                  case "all":
64                      print_all_images(request_info["place"], request_info["
    path"])
65                      return True
66                  case _:
67                      print(f"Faile to process images for source")
68                      return False
69          case "place":
70              nw = get_graph_from_place(request_info["place"])
71          case "coordinates":
72              nw = get_graph_from_coord(request_info["latitude"],
    request_info["longitude"], request_info["radius"])
73          case _:
74              print("Noop")
75              return False
76      return save_graph(graph=nw, graph_name=request_info["name"], base_path=
    request_info["path"], prune_keys=["geometry"], inverted_prune=False)
77
78  # OSMNX graph retrieval functions
79
80  def get_graph_from_place(place: str) -> nx.MultiDiGraph:
```

```
81      nw = ox.graph_from_place(query=place, simplify=True, network_type='
        drive')
82      nw = ox.add_edge_speeds(nw)
83      nw = ox.add_edge_travel_times(nw)
84      return nw
85
86  def get_graph_from_coord(lati: float, long: float, radius: int) -> nx.
        MultiDiGraph:
87      point = lati, long
88      nw = ox.graph_from_point(center_point=point, dist=radius, simplify=True
        , network_type="drive")
89      nw = ox.add_edge_speeds(nw)
90      nw = ox.add_edge_travel_times(nw)
91      return nw
92
93  # Graph processing functions
94
95  def save_graph(
96          graph: nx.MultiDiGraph,
97          graph_name: str,
98          base_path: str = "output",
99          prune_keys: list[str] = [],
100         inverted_prune: bool = False,
101         needed_keys: list[str] = result_dict_needed) -> bool:
102     path = base_path + "/" + graph_name + "/"
103
104     if not create_path(path):
105         print(f"Error saving graph at {path}: Failed to create directory")
106         return False
107
108     if not check_graph_integrity(graph, needed_keys):
109         print(f"Error saving graph at {path}: Graph integrity issues")
110         return False
111
112     if len(prune_keys) >= 1:
113         graph = prune_graph_info(graph, prune_keys, inverted_prune)
114
115     if not save_nodes_info(graph, path):
116         print(f"Error saving graph at {path}: Failed to save node data")
117         return False
118
119     if not save_edges_info(graph, path):
120         print(f"Error saving graph at {path}: Failed to save edge data")
121         return False
122
123     return True
124
```

```
125 def create_path(path: str) -> bool:
126     try:
127         pathlib.Path(path).mkdir(parents=True, exist_ok=True)
128         return True
129     except Exception as e:
130         print(f"Failed to create path {path} with exception: {e}")
131         return False
132 
133 def check_graph_integrity(graph: nx.MultiDiGraph, needed_keys: list[str])
    -> bool:
134     def try_fix(missing_field: str, edge_info: dict) -> any:
135         match missing_field:
136             case "maxspeed":
137                 if "speed_kph" in edge_info:
138                     return edge_info["speed_kph"]
139                 highway_type = edge_info["highway"]
140                 return default_speeds[highway_type]
141             case "lanes":
142                 return 1
143             case _:
144                 raise Exception()
145 
146     for node in graph.nodes(data=True):
147         if (node[1]["x"] is None or
148             node[1]["y"] is None or
149             node[0] is None):
150             return False
151 
152     node_list = list(graph.nodes)
153 
154     for edge in graph.edges(data=True):
155         if not edge[0] in node_list or not edge[1] in node_list:
156             print(f"Edge ({edge[0]}, {edge[1]}) not found in node data")
157             return False
158         for key, value_type in needed_keys.items():
159             if key not in edge[2]:
160                 # print(f"WARN Edge ({edge[0]}, {edge[1]}) missing '{key}'
    data, trying default values")
161                 try:
162                     edge[2][key] = try_fix(key, edge[2])
163                     # print(f"Default value inserted for {key} field!")
164                 except:
165                     print("Failed to fix. Full edge info: \n")
166                     print(edge[2])
167                     return False
168             if type(edge[2][key]) == list:
169                 try:
```

```
170                     edge[2][key] = [value_type(x) for x in edge[2][key]]
171                     edge[2][key] = min(edge[2][key])
172                 except:
173                     print(f"Type fixing error: List {edge[2][key]} from [{
    key}] for type {value_type} has no minimum")
174                     print(edge)
175                     return False
176             elif type(edge[2][key]) != value_type:
177                 try:
178                     edge[2][key] = value_type(edge[2][key])
179                 except:
180                     print(f"Type fixing error: {edge[2][key]} from [{key}]
    for type {value_type}")
181                     print(edge)
182                     return False
183 
184     return True
185 
186 def prune_graph_info(graph: nx.MultiDiGraph, prune_keys: list[str],
    is_inverted: bool = False) -> nx.MultiDiGraph:
187     if is_inverted:
188         for edge in graph.edges(data=True):
189             remove_list = []
190             for key, _ in edge[2].items():
191                 if key not in prune_keys:
192                     remove_list.append(key)
193             for key in remove_list:
194                 edge[2].pop(key)
195     else:
196         for edge in graph.edges(data=True):
197             for key in prune_keys:
198                 if key in edge[2]:
199                     edge[2].pop(key)
200     return graph
201 
202 def save_nodes_info(graph: nx.MultiDiGraph, path: str) -> bool:
203     data = list(graph.nodes(data=True))
204     for i in range(len(data)):
205         entry = {
206             "id": data[i][0],
207             "latitude": data[i][1]["y"],
208             "longitude": data[i][1]["x"],
209         }
210         data[i] = entry
211     try:
212         fullpath = path + "/nodes.json"
213         with open(fullpath, 'w') as f:
```

```
214             json.dump(data, f, indent=4, ensure_ascii=False)
215         return True
216     except Exception as e:
217         print(f"Exception: {e}")
218         return False
219
220 def save_edges_info(graph: nx.MultiDiGraph, path: str) -> bool:
221     data = list(graph.edges(data=True))
222     for i in range(len(data)):
223         entry = {
224             "id": int(data[i][2]["osmid"]),
225             "start": data[i][0],
226             "end": data[i][1],
227             "lanes": float(data[i][2]["lanes"]),
228             "maxspeed": int(data[i][2]["maxspeed"]),
229             "length": data[i][2]["length"],
230             "oneway": data[i][2]["oneway"],
231             "highway": data[i][2]["highway"],
232         }
233         data[i] = entry
234     try:
235         fullpath = path + "/edges.json"
236         with open(fullpath, 'w') as f:
237             json.dump(data, f, indent=4, ensure_ascii=False)
238         return True
239     except Exception as e:
240         print(f"Exception: {e}")
241         return False
242
243 # Image processing functions
244
245 def print_all_images(place: str, path: str) -> bool:
246     osm_data = print_image("osm", place, path)
247     gmaps_data = print_image("gmaps", place, path)
248     diff_data = print_diff(osm_data, gmaps_data, place, path)
249     return True
250
251 def print_diff(source1: dict, source2: dict, place: str, path:str) -> dict:
252     px = 1/plt.rcParams['figure.dpi']
253     size = 2048*px
254
255     graph = get_graph_from_place(place)
256     diff_data_dict = {}
257     for key in source1:
258         diff_data_dict[key] = source1[key] - source2[key]
259     ec = [diff_color_dict(diff_data_dict[(u, v)], 15) for u, v, _ in graph.
    edges(keys=True)]
```

```
260     _, _ = ox.plot_graph(graph, node_color='w', node_edgecolor='k',
        node_size=0, figsize=(size,size),
261                          node_zorder=1, edge_color=ec, edge_linewidth=2,
        bgcolor='white', show=False, save=True, filepath=path+"/test_diff.png")
262     return diff_data_dict
263
264 def print_image(source: str, place: str, path: str) -> dict:
265     px = 1/plt.rcParams['figure.dpi']
266     size = 2048*px
267
268     graph = get_graph_from_place(place)
269     markov_chain_filename = path + "/markov_chain_" + source + ".json"
270     markov_chain_data = {}
271     with open(markov_chain_filename) as json_file:
272         markov_chain_data = json.load(json_file)
273     markov_chain_data = [(x["street_data"]["start"], x["street_data"]["end
        "], x["traffic_data"]["estimated_density"]["Known"]) for x in
        markov_chain_data["graph"]]
274     markov_chain_data_dict = {}
275     for (u, v, d) in markov_chain_data:
276         markov_chain_data_dict[(u, v)] = d
277     ec = [density_color_dict(markov_chain_data_dict[(u, v)], 15) for u, v,
        _ in graph.edges(keys=True)]
278     _, _ = ox.plot_graph(graph, node_color='w', node_edgecolor='k',
        node_size=0, figsize=(size,size),
279                          node_zorder=1, edge_color=ec, edge_linewidth=2,
        bgcolor='white', show=False, save=True, filepath=path+"/test_"+source+".
        png")
280     return markov_chain_data_dict
281
282 def density_color_dict(value: float, max: float):
283     if value is None:
284         return 'blue'
285
286     if value <= 7:
287         return 'green'
288     elif value <= 16:
289         return 'lime'
290     elif value <= 22:
291         return 'orange'
292     elif value <= 28:
293         return 'red'
294     else:
295         return 'darkred'
296
297 def diff_color_dict(value: float, max: float):
298     if value is None:
```

```
299         return 'yellow'
300
301     if value <= -20:
302         return 'red'
303     elif value <= -10:
304         return 'lightcoral'
305     elif value <= -3:
306         return 'mistyrose'
307     elif value <= 3:
308         return 'lightgray'
309     elif value <= 10:
310         return 'lightsteelblue'
311     elif value <= 20:
312         return 'cornflowerblue'
313     else:
314         return 'blue'
315
316 # Main function
317
318 def main():
319     request = process_args()
320     if process_request(request_info=request):
321         pass
322     else:
323         print("Failed to process request")
324
325 if __name__ == "__main__":
326     main()
```

7.2.1.2 cli.py

```
1  import argparse
2
3  # Example
4  # poetry run python3 osm_tool/__init__.py -p "Jos  Mendes, Florian polis"
       -n jose_mendes
5
6  parser = argparse.ArgumentParser(description='Retrieve OSM graphs and data
       ')
7  parser.add_argument('-d',
8                      '--dry_run',
9                      action='store_true',
10                     help='Save results to file')
11
12 parser.add_argument('-i',
13                     '--image_generation',
14                     action='store_true',
```

```
                    help='Create image from data')

parser.add_argument('-s',
                    '--source',
                    type=str,
                    help='Source of data for image genaration')

parser.add_argument('-p',
                    '--place',
                    type=str,
                    help='Place for retrieving data')

parser.add_argument('-a',
                    '--latitude',
                    type=float,
                    help='Latitude for the center of data retrieval')

parser.add_argument('-o',
                    '--longitude',
                    type=float,
                    help='Longitude for the center of data retrieval')

parser.add_argument('-r',
                    '--radius',
                    type=int,
                    help='Radius around the center of data retrieval')

parser.add_argument('-f',
                    '--file_path',
                    type=str,
                    help='Path for storing results')

parser.add_argument('-n',
                    '--name',
                    type=str,
                    help='Name for the data retrieval results')

args = parser.parse_args()
```

7.2.1.3 pyproject.toml

```
[tool.poetry]
name = "osm-tool"
version = "0.1.0"
description = ""
authors = ["Matheus Roque <matheusdcroque@gmail.com>"]
readme = "README.md"
```

```
7
8 [tool.poetry.dependencies]
9 python = ">=3.12, <3.13"
10 osmnx = "^1.9.4"
11 matplotlib = "^3.9.2"
12
13 [build-system]
14 requires = ["poetry-core"]
15 build-backend = "poetry.core.masonry.api"
```

7.2.2 Rust

7.2.2.1 main.rs

```
1 use std::process::exit;
2
3 use geomarkover::{data_reader, markov_chain, osm};
4
5 use structopt::StructOpt;
6
7 #[derive(StructOpt)]
8 struct ArgsTransitionMatrix {
9     #[structopt(short = "n", long = "name")]
10     name: String,
11     #[structopt(short = "p", long = "place")]
12     place_name: Option<String>,
13     #[structopt(short = "f", long = "filepath")]
14     nw_graph_path: Option<String>,
15     #[structopt(short = "d", long = "datasource", default_value = "osm")]
16     data_source: String,
17     #[structopt(short = "o", long = "output")]
18     show_output: bool,
19     #[structopt(short = "s", long = "save")]
20     save_results: bool,
21 }
22
23 #[derive(StructOpt)]
24 enum Cli {
25     #[structopt(about = "Calculate transition matrix for a given location
    .")]
26     CalcTransitionMatrix(ArgsTransitionMatrix),
27 }
28
29 #[tokio::main]
30 async fn main() {
31     let cli = Cli::from_args();
32
```

```
33      match cli {
34          Cli::CalcTransitionMatrix(args) => {
35              let data_source = markov_chain::TrafficDataSource::from_str(&
    args.data_source).await;
36
37              let filepath: String;
38              let nw = match args.nw_graph_path {
39                  Some(path) => {
40                      filepath = path.clone();
41                      data_reader::NetworkData::new_from_file(args.name, path
    )
42                  }
43                  None => match args.place_name {
44                      Some(place) => {
45                          filepath = format!("output/{}", args.name);
46                          osm::get_data_from_place(&args.name, &place);
47                          data_reader::NetworkData::new_from_file(args.name.
    clone(), filepath.clone())
48                      }
49                      None => {
50                          println!("noop");
51                          exit(0)
52                      }
53                  },
54              };
55
56              let mut mkv_chain = markov_chain::MarkovChain::new_from_network
    (data_source, nw).await;
57              let t_mtx = markov_chain::TransitionMatrix::
    new_from_markov_chain(&mkv_chain);
58              mkv_chain.calculate_density_from_matrix(&t_mtx, None);
59
60              if args.show_output {
61                  println!("PRINT");
62              }
63
64              if args.save_results {
65                  if mkv_chain.save_data(filepath.clone(), args.data_source.
    clone()) {
66                      println!(
67                          "Saved markov chain data to {}/markov_chain.json",
68                          filepath.clone()
69                      );
70                  } else {
71                      println!(
72                          "Failed to save markov chain data to {}/
    markov_chain.json",
```

```
73                          filepath.clone()
74                      );
75                  }
76
77                  if t_mtx.save_to_file(filepath.clone(), args.data_source.
    clone()) {
78                      println!(
79                          "Saved markov chain data to {}/transition_matrix.
    csv",
80                          filepath.clone()
81                      );
82                  } else {
83                      println!(
84                          "Failed to save markov chain data to {}/
    transition_matrix.csv",
85                          filepath.clone()
86                      );
87                  }
88              }
89          }
90      }
91 }
```

7.2.2.2 lib.rs

```
1 pub mod data_reader;
2 pub mod google_routes;
3 pub mod markov_chain;
4 pub mod osm;
```

7.2.2.3 data_reader.rs

```
1 use std::fs::File;
2 use std::io::BufReader;
3 use std::path::Path;
4
5 use serde::{Deserialize, Serialize};
6
7 #[derive(Debug, Serialize, Deserialize, Clone)]
8 pub struct Intersection {
9     pub id: u64,
10     pub latitude: f64,
11     pub longitude: f64,
12 }
13
14 #[derive(Debug, Serialize, Deserialize, Clone)]
```

```
15 pub struct Street {
16     pub id: u64,
17     pub start: u64,
18     pub end: u64,
19     pub lanes: f64,
20     pub maxspeed: u8,
21     pub length: f64,
22     pub oneway: bool,
23     pub highway: String,
24 }
25
26 #[derive(Debug)]
27 pub struct NetworkData {
28     pub name: String,
29     pub nodes: Vec<Intersection>,
30     pub edges: Vec<Street>,
31 }
32
33 impl NetworkData {
34     pub fn new(name: String, nodes: Vec<Intersection>, edges: Vec<Street>)
    -> Self {
35         NetworkData { name, nodes, edges }
36     }
37
38     pub fn new_from_file(name: String, files_location: String) -> Self {
39         let node_filename = format!("{files_location}/nodes.json");
40         let path = Path::new(&node_filename);
41         let file = File::open(path).unwrap();
42         let reader = BufReader::new(file);
43         let nodes: Vec<Intersection> =
44             serde_json::from_reader(reader).expect("Nodes JSON was not well
    -formatted");
45
46         let edge_filename = format!("{files_location}/edges.json");
47         let path = Path::new(&edge_filename);
48         let file = File::open(path).unwrap();
49         let reader = BufReader::new(file);
50         let edges: Vec<Street> =
51             serde_json::from_reader(reader).expect("Edges JSON was not well
    -formatted");
52
53         NetworkData { name, nodes, edges }
54     }
55 }
56
57 mod tests {
58
```

```
59      #[test]
60      fn read_output_from_osm() {
61          crate::osm::get_data_from_place("jose_mendes", "Jos  Mendes,
    Florian polis");
62          let _ = super::NetworkData::new_from_file(
63              "jose_mendes".to_string(),
64              "output/jose_mendes".to_string(),
65          );
66      }
67  }
```

7.2.2.4 osm.rs

```
1  use std::process::Command;
2
3  pub fn get_data_from_place(name: &str, place: &str) {
4      let _ = Command::new("poetry")
5          .arg("-C")
6          .arg("python-scripts")
7          .arg("run")
8          .arg("python3")
9          .arg("python-scripts/osm_tool/__init__.py")
10         .arg("-p")
11         .arg(place)
12         .arg("-n")
13         .arg(name)
14         .spawn()
15         .expect("Rust hereby announces that Python forsaken ourselves")
16         .wait();
17 }
18
19 mod tests {
20     #[test]
21     fn get_osm_data() {
22         super::get_data_from_place("jose_mendes", "Jos  Mendes,
    Florian polis")
23     }
24 }
```

7.2.2.5 google_routes.rs

```
1  use google_maps::prelude::*;
2  use google_maps::directions::DepartureTime;
3
4  pub struct GoogleMapsHandler {
5      client: GoogleMapsClient,
```

```
}

#[derive(Debug)]
pub struct RoutesResponse {
    pub distance: f64,
    pub time_secs: f64,
    pub estimated_average_speed: f64,
    pub estimated_travel_time: f64,
}

impl GoogleMapsHandler {
    pub async fn new(gcp_key: String) -> Self {
        let client = GoogleMapsClient::try_new(gcp_key).unwrap();
        GoogleMapsHandler { client }
    }

    pub async fn directions(
        &self,
        from: (f64, f64),
        to: (f64, f64),
    ) -> Result<RoutesResponse, Error> {
        let result = self
            .client
            .directions(
                Location::try_from_f64(from.0, from.1).unwrap(),
                Location::try_from_f64(to.0, to.1).unwrap(),
            )
            .with_travel_mode(TravelMode::Driving)
            .with_departure_time(DepartureTime::Now)
            .execute()
            .await;

        match result {
            Ok(r) => {
                let distance = (r.routes[0].legs[0].distance.value as f64).
    max(1.0);
                let time_secs = (match &r.routes[0].legs[0].
    duration_in_traffic {
                    None => r.routes[0].legs[0].duration.value.num_seconds
    () as f64,
                    Some(v) => v.value.num_seconds() as f64,
                }).max(0.0001);
                let estimated_average_speed = (3.6 * distance) / time_secs;
                let estimated_travel_time = (distance / 1000.0) /
    estimated_average_speed;
                Ok(RoutesResponse {
                    distance,
```

```
                    time_secs,
                    estimated_average_speed,
                    estimated_travel_time,
                })
            }
            e => Err(e.err().unwrap()),
        }
    }
}

mod tests {
    #[actix_rt::test]
    #[ignore]
    async fn test_directions() {
        println!("create client");
        let handler =
            super::GoogleMapsHandler::new("insert_key_here".to_string())
                .await;
        println!("get directions");
        let directions = handler
            .directions((-27.6075094, -48.5478889), (-27.6078129,
    -48.5477348))
            .await;
        println!("print directions");
        println!("{:?}", directions);
    }
}
```

7.2.2.6 markov_chain.rs

```
use std::fs::{self, File, OpenOptions};
use std::io::Write;
use std::sync::atomic::{AtomicUsize, Ordering};

use crate::data_reader::*;
use crate::google_routes::*;

use futures::future;
use futures::stream::Collect;
use serde::Serialize;
use serde_json;

#[derive(Debug, Serialize, Clone)]
pub enum Value {
    Known(f64),
    Unknown(f64),
}
```

```
18
19 impl Value {
20     pub fn as_f64(&self) -> f64 {
21         match &self {
22             Value::Known(v) => *v,
23             Value::Unknown(_) => f64::NAN,
24         }
25     }
26 }
27
28 impl std::fmt::Display for Value {
29     fn fmt(&self, f: &mut std::fmt::Formatter) -> std::fmt::Result {
30         match &self {
31             Value::Known(v) => write!(f, "{}", v),
32             Value::Unknown(_) => write!(f, "U"),
33         }
34     }
35 }
36
37 #[derive(Debug, Serialize, Clone)]
38 pub struct MarkovNode {
39     id: u64,
40     id_osm: u64,
41     street_start: Intersection,
42     street_end: Intersection,
43     street_data: Street,
44     traffic_data: Option<TrafficFlow>,
45     transitions: Vec<MarkovTransition>,
46 }
47
48 #[derive(Debug, Serialize, Clone)]
49 pub struct TrafficFlow {
50     estimated_travel_time: Value,
51     estimated_average_speed: Value,
52     estimated_density: Value,
53 }
54
55 #[derive(Debug, Serialize, Clone)]
56 pub struct MarkovTransition {
57     id_to: u64,
58     probability: Value,
59 }
60
61 #[derive(Debug, Serialize, Clone)]
62 pub struct TransitionMatrix {
63     dim: usize,
64     pub matrix: Vec<(usize, usize, f64)>,
```

```
65 }
66
67 impl std::ops::Index<(u64, u64)> for TransitionMatrix {
68     type Output = f64;
69
70     fn index(&self, i: (u64, u64)) -> &f64 {
71         match &self
72             .matrix
73             .iter()
74             .find(|(n, m, _)| *n == i.0 as usize && *m == i.1 as usize)
75         {
76             None => &0.0,
77             Some((_, _, value)) => value,
78         }
79     }
80 }
81
82 impl TransitionMatrix {
83     pub fn new_from_markov_chain(mkv_chain: &MarkovChain) -> Self {
84         let dim = mkv_chain.graph.len();
85         let mut matrix: Vec<(usize, usize, f64)> = Vec::new();
86         for node in &mkv_chain.graph {
87             for t in &node.transitions {
88                 matrix.push((node.id as usize, t.id_to as usize, t.
    probability.as_f64()));
89             }
90         }
91
92         TransitionMatrix { dim, matrix }
93     }
94
95     fn to(&self, i: u64) -> Vec<(usize, usize, f64)> {
96         self.clone()
97             .matrix
98             .into_iter()
99             .filter(|(_, m, _)| *m == i as usize)
100            .collect::<Vec<(usize, usize, f64)>>()
101    }
102
103    fn _from(&self, i: u64) -> Vec<(usize, usize, f64)> {
104        self.clone()
105            .matrix
106            .into_iter()
107            .filter(|(n, _, _)| *n == i as usize)
108            .collect::<Vec<(usize, usize, f64)>>()
109    }
110
```

```
111     pub fn print(&self) {
112         let dim = self.dim;
113         let matrix = &self.matrix;
114         for i in 0..dim {
115             let mut line = vec![0.0; dim];
116             for x in matrix {
117                 match x {
118                     (a, b, p) if *a == i => {
119                         line[*b] = *p;
120                     }
121                     _ => (),
122                 }
123             }
124 
125             for x in line {
126                 print!("{}\t", x);
127             }
128             println!();
129         }
130     }
131 
132     pub fn save_to_file(&self, path: String, data_source_str: String) ->
    bool {
133         let dim = self.dim;
134         let matrix = &self.matrix;
135         let path = format!("{}/transtition_matrix_{}.csv", path,
    data_source_str);
136         if fs::remove_file(path.clone()).is_ok() {
137             println!("Removed previous data from {}", path);
138         }
139         let mut file = OpenOptions::new()
140             .create(true)
141             .append(true)
142             .open(path)
143             .unwrap();
144 
145         for i in 0..dim {
146             let mut line = vec![0.0; dim];
147             for x in matrix {
148                 match x {
149                     (a, b, p) if *a == i => {
150                         line[*b] = *p;
151                     }
152                     _ => (),
153                 }
154             }
155 
```

```
156             let mut line_str = "".to_string();
157             for x in line {
158                 let element = format!("{},", x).to_string();
159                 line_str = format!("{}{}", line_str, element);
160             }
161             line_str = format!("{}\n", line_str);
162             match file.write_all(line_str.as_bytes()) {
163                 Ok(_) => (),
164                 _ => {
165                     println!("Failed to save content to file");
166                     return false;
167                 }
168             }
169         }
170         true
171     }
172 }
173
174 pub enum TrafficDataSource {
175     GoogleRoutes(GoogleMapsHandler),
176     OpenStreetMap,
177     NoSource,
178     Unknown,
179 }
180
181 impl TrafficDataSource {
182     pub async fn from_str(s: &str) -> Self {
183         match s {
184             "gmaps" => TrafficDataSource::GoogleRoutes(
185                 GoogleMapsHandler::new("insert_key_here".to_string()).await
    ,
186             ),
187             "osm" => TrafficDataSource::OpenStreetMap,
188             _ => TrafficDataSource::Unknown,
189         }
190     }
191 }
192
193 #[derive(Debug, Serialize)]
194 pub struct MarkovChain {
195     name: String,
196     graph: Vec<MarkovNode>,
197 }
198
199 static ID_COUNTER: AtomicUsize = AtomicUsize::new(0);
200 impl MarkovChain {
201     pub async fn new_from_network(
```

```
202         traffic_data_source: TrafficDataSource,
203         network_graph: NetworkData,
204     ) -> Self {
205         let name = network_graph.name;
206 
207         let mut graph: Vec<MarkovNode> =
208             future::join_all(network_graph.edges.into_iter().map(|x| async
    {
209                 MarkovNode {
210                     id: ID_COUNTER.fetch_add(1, Ordering::Relaxed) as u64,
211                     id_osm: x.id,
212                     street_start: network_graph
213                         .nodes
214                         .iter()
215                         .find(|i| i.id == x.start)
216                         .unwrap()
217                         .clone(),
218                     street_end: network_graph
219                         .nodes
220                         .iter()
221                         .find(|i| i.id == x.end)
222                         .unwrap()
223                         .clone(),
224                     street_data: x,
225                     traffic_data: None,
226                     transitions: Vec::new(),
227                 }
228             }))
229             .await;
230 
231         let street_vec: Vec<(u64, Intersection, Intersection)> = graph
232             .clone()
233             .into_iter()
234             .map(|x| (x.id, x.street_start, x.street_end))
235             .collect();
236 
237         graph = future::join_all(graph.into_iter().map(|mut x| async {
238             x.traffic_data = MarkovChain::get_traffic_data(
239                 &traffic_data_source,
240                 &x.street_data,
241                 &x.street_start,
242                 &x.street_end,
243             )
244             .await;
245             x
246         }))
247         .await;
```

```
248
249         graph = graph
250             .into_iter()
251             .map(|mut x| {
252                 let adjusted_lanes = match x.street_data.oneway {
253                     true => x.street_data.lanes,
254                     false => x.street_data.lanes / 2.0,
255                 };
256                 x.street_data.lanes = adjusted_lanes;
257                 for y in street_vec.iter() {
258                     let x_start = x.street_data.start;
259                     let y_start = y.1.id;
260                     let x_end = x.street_data.end;
261                     let y_end = y.2.id;
262                     match (x_start, y_start, x_end, y_end) {
263                         (xs, ys, xe, ye) if xs == ys && xe == ye => {
264                             x.transitions.push(MarkovTransition {
265                                 id_to: x.id,
266                                 probability: x.traffic_data.clone().unwrap
    ().estimated_travel_time,
267                             });
268                         }
269                         (_, ys, xe, _) if ys == xe => {
270                             x.transitions.push(MarkovTransition {
271                                 id_to: y.0,
272                                 probability: Value::Unknown(0.0),
273                             });
274                         }
275                         (_, _, _, _) => (),
276                     }
277                 }
278                 x
279             })
280             .collect();
281
282         let min_travel_time = graph
283             .iter()
284             .map(|x| {
285                 x.traffic_data
286                     .clone()
287                     .unwrap()
288                     .estimated_travel_time
289                     .as_f64()
290             })
291             .collect::<Vec<f64>>()
292             .into_iter()
293             .min_by(|a, b| match a.partial_cmp(b) {
```

```
294                 Some(x) => x,
295                 None => {
296                     println!("a = {a}, b = {b}");
297                     if a.is_nan() {
298                         b.partial_cmp(b).unwrap()
299                     } else {
300                         a.partial_cmp(a).unwrap()
301                     }
302                 }
303             })
304             .unwrap();
305 
306         graph = graph
307             .into_iter()
308             .map(|mut mkv_node| {
309                 let norm_tt = mkv_node
310                     .traffic_data
311                     .clone()
312                     .unwrap()
313                     .estimated_travel_time
314                     .as_f64()
315                     / min_travel_time;
316                 mkv_node.transitions = mkv_node
317                     .transitions
318                     .into_iter()
319                     .map(|mut t| {
320                         t.probability = match t.id_to {
321                             id if id == mkv_node.id => Value::Known((
    norm_tt - 1.0) / norm_tt),
322                             _ => t.probability,
323                         };
324                         t
325                     })
326                     .collect();
327                 mkv_node
328             })
329             .map(|mut mkv_node| {
330                 let self_transition_prob = mkv_node
331                     .transitions
332                     .clone()
333                     .into_iter()
334                     .find(|t| t.id_to == mkv_node.id)
335                     .unwrap()
336                     .probability
337                     .as_f64();
338 
339                 let num_transitions = mkv_node.transitions.len();
```

```
340
341                 mkv_node.transitions = mkv_node
342                     .transitions
343                     .into_iter()
344                     .map(|mut t| {
345                         t.probability = match t.id_to {
346                             id if id == mkv_node.id => t.probability,
347                             _ => Value::Known(
348                                 (1.0 - self_transition_prob) / (
    num_transitions as f64 - 1.0),
349                             ),
350                         };
351                         t
352                     })
353                     .collect();
354                 mkv_node
355             })
356             .collect();
357         MarkovChain { name, graph }
358     }
359
360     async fn get_traffic_data(
361         source: &TrafficDataSource,
362         street_info: &Street,
363         street_start: &Intersection,
364         street_end: &Intersection,
365     ) -> Option<TrafficFlow> {
366         match source {
367             TrafficDataSource::OpenStreetMap => Some(TrafficFlow {
368                 estimated_travel_time: Value::Known(
369                     (street_info.length / 1000.0) / (street_info.maxspeed
    as f64),
370                 ),
371                 estimated_average_speed: Value::Known(street_info.maxspeed
    as f64),
372                 estimated_density: Value::Unknown(0.0),
373             }),
374             TrafficDataSource::GoogleRoutes(handler) => {
375                 let traffic_data = match handler
376                     .directions(
377                         (street_start.latitude, street_start.longitude),
378                         (street_end.latitude, street_end.longitude),
379                     )
380                     .await
381                 {
382                     Ok(r) => r,
383                     e => e.unwrap(),
```

```
384                 };
385
386                 Some(TrafficFlow {
387                     estimated_travel_time: Value::Known(traffic_data.
    estimated_travel_time),
388                     estimated_average_speed: Value::Known(traffic_data.
    estimated_average_speed),
389                     estimated_density: Value::Unknown(0.0),
390                 })
391             }
392             _ => None,
393         }
394     }
395
396     fn node(graph: &[MarkovNode], i: u64) -> MarkovNode {
397         graph
398             .to_owned()
399             .clone()
400             .into_iter()
401             .find(|x| x.id == i)
402             .unwrap()
403     }
404
405     // Supondo densidade livre em todos os trechos inicialmente -> 7 vei/km
    /faixa
406     pub fn estimate_vehicle_count(&self) -> u64 {
407         self.graph
408             .iter()
409             .map(|x| {
410                 7.0*x.street_data.lanes*x.street_data.length/1000.0
411             })
412             .sum::<f64>() as u64
413     }
414
415     pub fn calculate_density_from_matrix(&mut self, t_mtx: &
    TransitionMatrix, vehicle_count: Option<u64>) {
416         let vehicle_count = match vehicle_count {
417             None => self.estimate_vehicle_count(),
418             Some(v) => v,
419         };
420
421         let h_graph = self.graph.clone();
422
423         self.graph = self
424             .graph
425             .clone()
426             .into_iter()
```

```
427             .map(|mut x| {
428                 let self_prob = t_mtx[(x.street_start.id, x.street_end.id)
    ];
429                 let other_prob = t_mtx.to(x.id);
430                 let mut density = MarkovChain::calculate_density_parcel(
431                     vehicle_count,
432                     self_prob,
433                     x.street_data.length,
434                     x.street_data.lanes,
435                 );
436                 for (from, _, prob) in other_prob {
437                     let node = MarkovChain::node(&h_graph, from as u64);
438                     density += MarkovChain::calculate_density_parcel(
439                         vehicle_count,
440                         prob,
441                         node.street_data.length,
442                         node.street_data.lanes,
443                     );
444                 }
445                 x.traffic_data = Some(TrafficFlow {
446                     estimated_travel_time: x.traffic_data.clone().unwrap().
    estimated_travel_time,
447                     estimated_average_speed: x
448                         .traffic_data
449                         .clone()
450                         .unwrap()
451                         .estimated_average_speed,
452                     estimated_density: Value::Known(density),
453                 });
454                 x
455             })
456             .collect::<Vec<MarkovNode>>();
457     }
458
459     fn calculate_density_parcel(v: u64, prob: f64, l: f64, n: f64) -> f64 {
460         (v as f64 * prob) / (l * n)
461     }
462
463     pub fn save_data(&self, path: String, data_source_str: String) -> bool
    {
464         let output_str: String = match serde_json::to_string_pretty(&self)
    {
465             Ok(v) => v,
466             _ => return false,
467         };
468
469         let path = format!("{}/markov_chain_{}.json", path, data_source_str
```

```
    );

        let mut file = File::create(path).unwrap();
        match file.write_all(output_str.as_bytes()) {
            Ok(_) => true,
            _ => {
                println!("Failed to save content to file");
                false
            }
        }
    }
}

mod tests {
    #[actix_rt::test]
    async fn new_markov_chain_from_file() {
        let nw = crate::data_reader::NetworkData::new_from_file(
            "jose_mendes".to_string(),
            "output/jose_mendes".to_string(),
        );
        let mkv_chain = super::MarkovChain::new_from_network(
            super::TrafficDataSource::from_str("osm").await,
            nw,
        )
        .await;
        for node in mkv_chain.graph {
            println!("NODE ID: {:?}", node.id);
            println!(
                "TT: {:?}",
                node.traffic_data.unwrap().estimated_travel_time
            );
            for t in node.transitions {
                println!("{:?}", t);
            }
        }
    }
}
```

7.2.2.7 Cargo.toml

```
[package]
name = "geomarkover"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/
    reference/manifest.html
```

```
7
8 [dependencies]
9 serde_json = "1.0.128"
10 serde = {version = "1.0.210", features = ["derive"]}
11 chrono = "0.4.38"
12 structopt = { version = "0.3.26", default-features = false }
13 google_maps = "3.7"
14 actix-web = "4.9.0"
15 actix-rt = "2.10.0"
16 futures = "0.3.31"
17 tokio = { version = "1.41.0", features = ["full"] }
```

ESTIMATIVA DE CADEIA DE MARKOV DE MALHA VIÁRIA COM BASE EM DADOS DE TRAFÉGO URBANO

Matheus Dhanyel C. Roque[1], **Rafael de Santiago**[1]

[1]Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

Abstract. *This paper focus on expanding upon traffic network representations, using high quality data [Rezzouqi et al. 2019] for such estimates. Based on representation models for traffic networks as Markov chains out of free data from OpenStreetMap [Salman 2018] this paper also uses data obtained from Google Maps to create Markov chain out of data with better real urban flows representativeness. Based on analyses done with estimates obtained from the tool developed, it is noted that the results from the new representations are resembles those obtained from OpenStreetMap alone.*

Resumo. *Este trabalho tem como foco expandir em cima de representações da malha viária, utilizando dados de alta qualidade [Rezzouqi et al. 2019] para realizar tais estimativas. Baseando-se de modelos de representações da malha viária como uma cadeia de Markov a partir de dados gratuitos do OpenStreetMap [Salman 2018] este trabalho utiliza também dados obtidos pelo Google Maps para criar cadeias de Markov a partir de dados com maior representatividade da realidade dos fluxos urbanos. A partir de análises das estimativas obtidas pela ferramenta desenvolvida nota-se que os resultados das novas representações são semelhantes dos obtidos a partir do OpenStreetMap.*

1. Introdução

Com o Brasil contando com uma frota terrestre de mais de 113 milhões de veículos, sendo 1.2% deste valor referente ao crescimento no primeiro semestre de 2022 [de Trânsito Senatran 2022], cada vez mais a malha viária enfrenta dificuldades para suprir com qualidade as necessidades de deslocamentos intra-urbanos das populações. Segundo pesquisa [Galindo 2019], mais de 30% da população das cidades com mais de 100 mil habitantes alegavam enfrentar engarrafamentos diariamente, mostrando um grande espaço para melhoria na eficiência do planejamento da mobilidade urbana.

Para melhorar a satisfação das necessidades de mobilidade urbana várias frentes podem ser abordadas, seja ampliando o uso de transportes coletivos, investindo em novos modais para dividir a carga ou replanejando a disposição e utilização das vias a fim de otimizar a sua vazão. Independente de qual a abordagem feita, para todas é crítico o conhecimento de qual o comportamento dos fluxos urbanos, identificando quais rotas e vias se encontram em sobrecarga e utilizando desta informação para traçar alvos de ação mais bem definidos e métricas de qualidade mais direcionadas.

A obtenção de tais dados é usualmente feita por pesquisas de campo, um processo caro e lento. Uma alternativa que vem crescendo é a coleta e processamento computacional de dados diversos para construir matrizes de origem-destino, como dados

de localização de *smartphones* [Junior et al. 2018] ou bilhetes eletrônicos de embarque [Pelletier et al. 2011], em muitos casos facilitando o processo e obtendo resultados comparáveis com a pesquisa de campo.

Outra via a ser considerada na questão de dados de tráfego é substituir a implementação de processos de obtenção de informações como os já citados por uma extração em dados já coletados por aplicações e serviços de mobilidade urbana como *Waze* e *Google Maps*, que utilizam dados de uma grande quantidade de usuários para alimentar seus cálculos. Tais dados de tráfego gratuitos possuem poucos detalhes, gerando uma necessidade de maior pré-processamento antes de seu uso [Mostafi and Elgazzar 2021], o que não impede que ferramentas desenvolvidas tratem esses dados não estruturados para gerar informações úteis [Churi 2021].

Para o escopo deste trabalho, a ferramenta produzida processará as informações coletadas em uma cadeia de Markov, com suas probabilidades de transição devidamente populadas, facilitando que trabalhos de *Road Network Design Problem* [Salman 2018] possam ser mais facilmente executados em qualquer região que tenha acesso à aplicações de cálculos de rota como o *Google Maps API* e aplicações de informações de geoprocessamento como o *Openstreetmap API*(OSM), ambas ferramentas que serão utilizadas para o desenvolvimento da aplicação deste estudo.

1.1. Estrutura do Texto

Este trabalho está estruturado em cinco seções.

A Seção 1, Introdução, apresentou uma apanhado geral quanto a objetivos, motivações e metodologia de desenvolvimento do trabalho.

Na Seção 2, Trabalhos Relacionados, trabalhos com temáticas ou assuntos relacionados com o objeto central deste trabalho são analisados e discutidos.

Na Seção 3, Métodos, são apresentadas as ferramentas utilizadas para a confecção da aplicação deste trabalho e descrito todo o processo de processamento de dados implementado na aplicação.

Na Seção 4, Resultados, são apresentados os experimentos utilizados para validar o funcionamento da ferramenta e as análises realizadas sob os mesmos.

Na Seção 5, Conclusão, são apresentadas as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Para a elaboração deste capítulo foi conduzida uma pesquisa por trabalhos focados em coleta de dados de tráfego, processamento de dados de tráfego e relação entre as diferentes variáveis de tráfego. Vários dos trabalhos encontrados constroem as referências e fundamentação para este trabalho. Dentre eles, dois foram escolhidos para discutir mais profundamente.

2.1. Alleviating Road Network Congestion: Traffic Pattern Optimization Using Markov Chain Traffic Assignment

Salman e Alaswad buscam neste trabalho [Salman 2018] realizar um experimento com métodos para reduzir os acúmulos de densidade de tráfego em vias urbanas, através de apenas alterações no sentido de uso das vias.

Os autores desenvolvem seu experimento representando a malha viária por meio de cadeias de Markov e utilizando métodos de algoritmos genéticos para chegar em uma solução ótima. Para construção da cadeia de Markov representativa são necessárias diversas informações sobre a região do experimento. Tais dados, como de velocidade, número de faixas e estrutura da malha viária, foram coletados com a *Openstreet Map*.

Após definir a região do experimento como a parte de ilha da cidade de Abu Dhabi, EAU, foram identificados e coletados os dados de 360 ruas inclusas na região. Também foi estimado que aproximadamente 10000 veículos populam as vias. Na solução proposta, as técnicas de algoritmo genético tentariam minimizar a densidade máxima em qualquer via da malha, considerando um individuo uma configuração da malha completa e o genoma do indivíduo a informação quanto a inversão das vias. Além de minimizar a densidade máxima, o algoritmo dos autores também priorizava indivíduos com menor número de inversões.

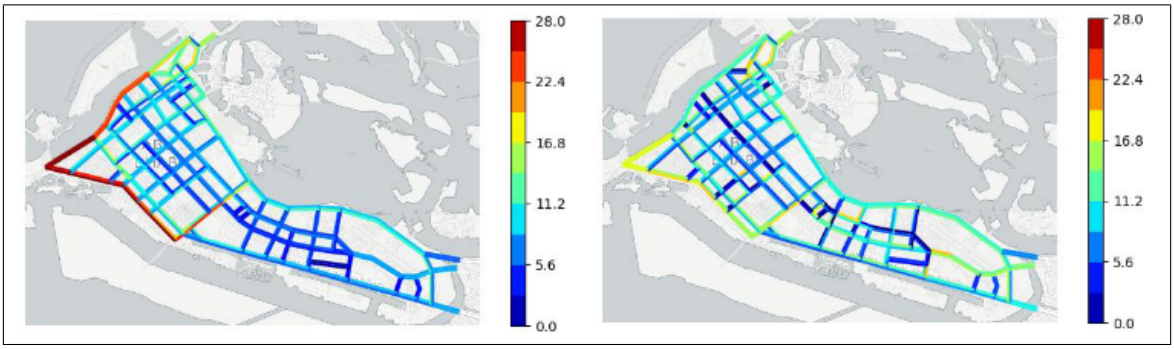

Figura 1. Resultados de experimento. À esquerda, mapa de calor de densidade de tráfego com a disposição atual das vias; À direita, mapa de calor de densidade de tráfego com a disposição de vias sugerida [Salman 2018]

Na 1 é apresentada uma comparação entre a densidade das vias no estado inicial da malha viária e após a aplicação do modelo proposto pelos autores. Com apenas 24 inversões a solução encontrada reduziria a densidade máxima encontra na região de 34,2 veículos/km/faixa para 19,8 veículos/km/faixa.

Com resultados interessantes, o trabalho de Salman e Alaswad demonstra um grande potencial a ser explorado no uso de cadeias de Markov para aplicações na área de mobilidade e planejamento de tráfego. Partindo desse potencial, o trabalho aqui proposto procura explorar formas obter de dados de tráfego com maior representatividade do fluxos reais, fornecendo fundação para que outros autores e pesquisadores sigam progredindo este campo de pesquisa.

2.2. An Open Source Tool to Extract Traffic Data from Google Maps: Limitations and Challenges

Logo na introdução da publicação é mencionado como motivação do trabalho a falta de ferramentas de qualidade confiável para extração de dados de tráfego urbano para pesquisa científica [Mostafi and Elgazzar 2021], algo que seria então facilitado pelas novas *APIs* da *Google* citadas na seção anterior deste capítulo. Além disso, os autores mencionam a existência de ferramentas pagas disponibilizadas pela *Google* que poderiam ser

utilizadas para tal fim mas buscavam maximizar a utilidade dos dados já disponibilizados gratuitamente pela plataforma.

Após definidas as motivações e problemas abordados no estudo são levantadas discussões sobre a confiabilidade dos dados de estimativa de tempo de viagem fornecidos pela ferramenta do *Google Maps*, algo crucial para o trabalho aqui proposto logo que toda a metodologia de desenvolvimento da aplicação final depende da qualidade de tais dados para ponderar o seu sucesso e alcance de seus objetivos. Dentre essas discussões são elencados alguns pontos sobre a plataforma do *Google Maps* e sobre os dados nela obtidos tais como:

- Os dados para realizar as estimativas são alimentados por *crowdsourcing* de telefones que utilizam o sistema da Android. Além disso, o uso desse tipo de dado fornece bons resultados para realizar tais estimativas de tráfego [Tahmasseby 2015];
- Em estudos de caso diferentes em Hong Kong [He et al. 2019] e Paris [Rezzouqi et al. 2019] ambos demonstraram alta qualidade nas estimativas realizadas pela plataforma, sendo a acurácia de 95,8% para situações de trânsito fluido no segundo experimento;
- Em uma pesquisa [Caiza et al. 2018] é demonstrada a capacidade de extrair informações suficientes para determinar regiões de congestionamento de tráfego, utilizando apenas das imagens de mapa fornecidas pela plataforma.

Em seguida são apresentadas as características da ferramenta proposta pelos autores, incluindo seu fluxo de execução e a interface que será fornecida para os usuários. O usuário teria como entrada sua localização, coordenadas ou selecionaria a posição diretamente na ferramenta do *Google Maps* para definir a origem do trajeto de interesse. Em seguida faria o mesmo tipo de entrada para definir o destino do trajeto de interesse, selecionaria entre opção de analisar por horário de saída ou de chegada no destino e informar o número de dias de observação.

A aplicação então faria a coleta de informações de estimativa de tempo de viagem em intervalos de 15 em 15 minutos durante o número de dias requisitados e estruturaria um compilado dos resultados em um arquivo de formato CSV e também na forma de gráficos onde são apresentados valores de velocidade média estimada.

An Open Source Tool to Extract Traffic Data from Google Maps: Limitations and Challenges desenvolve uma ferramenta de caráter parecido com a necessária para o presente trabalho e fornece análises interessantes acerca da mesma, que apesar de intrínsecas ao processo de desenvolvimento e do objetivo da pesquisa dos autores são cruciais para orientar o planejamento e execução do trabalho proposto.

3. Método Proposto

3.1. Ferramentas utilizadas

Sendo o objeto principal deste trabalho uma produção de *software*, algumas bibliotecas de uso livre foram utilizadas durante o processo de desenvolvimento da aplicação. Nesta seção, tais bibliotecas são brevemente apresentadas.

3.2. OSMnx

A OSMnx [Boeing 2024] é uma biblioteca de Python que acessa dados da *OpenStreetMap* para facilitar o processo de aquisição, modelagem, análise e utilização de dados de malhas viárias urbanas.

Dentre as diversas funcionalidades oferecidas pela biblioteca poucas foram utilizadas neste trabalho, sendo elas:

- Funcionalidades para obter um grafo representativo de uma região, com nodos representando interseções e arestas as ruas. Neste grafo são inclusas informações sobre as vias, como nodo origem, nodo destino, número de faixas, velocidade máxima e tipo de rodovia. Também estão inclusos as coordenadas geográficas de cada nodo;
- Funcionalidades para obter um grafo representativo de uma região a um raio de distância de uma coordenada geográfica, com detalhamento de informações obtidas idênticos ao obtido por região.
- Funcionalidades para criar visualizações de grafos gerados com a biblioteca.

A aplicação utiliza essa biblioteca para obter as informações necessárias para construir a cadeia de Markov bem como para realizar estimativas de densidade baseadas na velocidade média das vias.

3.2.1. google_maps

A *google_maps* é uma biblioteca disponível para Rust que encapsula vários métodos de cliente da *Google Maps Platform*, além de incluir estruturas que facilitam o uso dos serviços da *Google*. Neste trabalho a biblioteca é utilizada para facilitar a autenticação de cliente e realização de requisições de tempo de viagem dado através da *Directions API*.

Vale mencionar aqui que os serviços de rota fornecidos pela *Google Maps Platform* não são gratuitos, requerem a contratação dos serviços na plataforma e possuem custo por requisição feita atráves da *Directions API*, com valores variando de R$0,004 a R$0.005 por requisição [Google 2024]. A plataforma também disponibiliza US$ 300 de crédito no momento de contratação e US$ 200 de crédito mensalmente, sendo suficiente para cobrir quaisquer gastos provenientes dos testes durante o desenvolvimento deste trabalho.

3.3. Implementação

Nesta seção é apresentado brevemente a estrutura da implementação feita neste trabalho, focando em alguns detalhes de cálculos e algoritmos utilizados. A implementação da ferramenta proposta neste trabalho foi divida em duas partes bem definidas:

- *Scripts* em *Python* para obter dados da OSM e para gerar visualizações para análise;
- Aplicação em *Rust* para obter dados da *Directions API* e para processar juntamente os dados da OSM em cadeias de Markov.

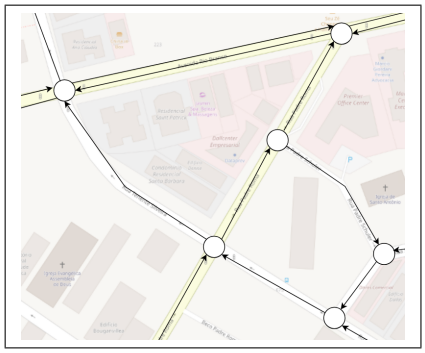

Figura 2. Grafo representativo dos dados provenientes da OSMnx.

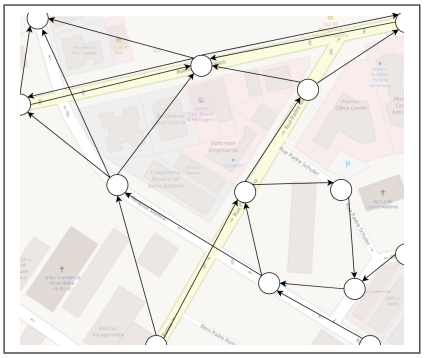

Figura 3. Grafo representativo da cadeia de Markov.

3.3.1. Pré-processamento de dados

A partir do nome de uma região geográfica (como o nome de uma cidade ou bairro) a aplicação obtém, através da OSMnx, um grafo direcionado representante da região populado com as informações sobre cada uma das vias. Então são realizados alguns pré-processamentos dos dados: conferência de que todos os campos necessários para a aplicação estão presente; remoção de informações que não serão úteis para a aplicação; reorganização dos dados para facilitar processamentos posteriores. Este processo é feito para os elementos de arestas do grafo, com as informações das vias, bem como para os elementos de nodo, com as informações das interseções.

Após tratados e com garantia de qualidade suficiente para uso, os dados de arestas e nodos são salvos separadamente em formato JSON com uma estrutura bem definida para posterior leitura.

3.3.2. Estruturação da cadeia de Markov

Após a fase de pré-processamento de dados é estruturado um novo grafo com todos os dados já disponíveis, no formato de uma cadeia de Markov. Neste novo grafo os nodos representam as vias e as arestas uma transição entre vias, diferente dos dados obtidos anteriormente pela OSMnx onde as arestas que representam as vias.

Por conta dessa diferença as informações das vias passam a ser referentes aos nodos do grafo e as arestas passam a possuir apenas informações sobre movimentos possíveis a partir de uma via.

Neste passo também foi optado por alterar as estruturas de dados que representam o grafo. Ao invés de possuir uma lista para nodos e para arestas separados, as cadeias de Markov são estruturadas com apenas uma lista de nodos, cada qual contendo uma lista de transições possíveis. Essa abordagem elimina a necessidade de pesquisas pelas transições ou de manutenção de ponteiros para as transições.

É então atribuído um novo identificador único para cada nodo (iniciando em 0), utilizado os dados das interseções (antigos nodos) para determinar o início e o fim de cada via e populado a lista de transições possíveis para cada nodo. Até então, dados sobre a

probabilidade de transição, densidade, velocidade média e tempo de travessia de cada via não foram estimados e são desconhecidos.

3.3.3. Cálculo de velocidade média e tempo de travessia

Para estimar os dados faltantes na cadeia de Markov são primeiramente necessárias informações sobre velocidade média e tempo de travessia de cada via em cada via. A aplicação desenvolvida possui duas opções de estimativa de tais informações dependendo da fonte de dados utilizada para tal.

Caso a base de estimativa sejam apenas os dados da OSMnx a aplicação considera que a velocidade média do usuário da via é sempre o valor de velocidade máxima da via. A partir dessa suposição é possível calcular o tempo de travessia:

$$tt = \frac{l}{v_e} \tag{1}$$

Sendo tt o tempo de travessia estimado em segundos, v_e a velocidade estimada em km/h e l o tamanho da via em quilômetros.

A aplicação também pode utilizar dados da *Google Maps* como base de estimativa dessas informações, sendo o ponto central da proposta deste trabalho. A principal motivação de utilizar tais dados para essa estimativa de velocidade média baseia-se em buscar informações com maior aderência com a realidade, em vista de que por influência de questões geométricas das vias e do volume de indivíduos na malha, a velocidade máxima da via nem sempre é a melhor suposição de velocidade média para os indivíduos.

Para isso é utilizado a coordenada de início e de fim de cada via como pontos de origem e destino, respectivamente, em consulta pelo *Directions API*, atráves das funcionalidades fornecidas pela biblioteca do *google_maps* discutida anteriormente, para obter a estimativa de tempo de travessia. Com o tempo de travessia é possível utilizar a equação 1 para determinar a velocidade média estimada.

3.3.4. Cálculo de transições

Para o cálculo das probabilidades de transição é primeiramente calculada a probabilidade de um indivíduo permanecer no estado em que se encontra.

Uma transição em uma cadeia de Markov representativa de uma malha viária corresponde à indivíduos saindo de um trecho de via e se posicionando em outro trecho. Para tal o indivíduo necessita percorrer completamente a via onde se encontra, relacionando então o tempo de travessia com os passos do processo da cadeia de Markov. Normalizando os tempos de travessia na via com base no menor tempo da malha, a probabilidade de um indivíduo permanecer no mesmo estado após um passo do processo é dada pela seguinte equação [Salman 2018]:

$$p_{ii} = \frac{tt_n - 1}{tt_n} \tag{2}$$

Sendo p_{ii} a probabilidade do indivíduo permanecer no mesmo estado e tt_n o tempo de travessia normalizado.

Essa probabilidade é também chamada probabilidade auto transição, interpretada de modo que o indivíduo que realiza a auto transição permanece na mesma via e mesmo sentido no próximo passo do processo. Para as demais transições é suposto que o indivíduo escolhe aleatoriamente qualquer uma das opções. Dessa forma, a probabilidade de transição dada por:

$$p_{ij} = \frac{1 - pii}{n_p - 1} \tag{3}$$

Sendo p_{ij} a probabilidade do indivíduo transitar do estado i para o estado j, p_{ii} a probabilidade do indivíduo permanecer no mesmo estado e n_p o número de transições possíveis (auto transição inclusa).

Com os dados de transição calculados, os dados são organizados em uma matriz de transição e salvos. Nesta matriz é utilizado o novo identificador atribuído para cada via durante a construção da estrutura da cadeia de Markov como índice, de modo que o elemento p_{ij} da matriz é a probabilidade de transição partindo da via de identificador i para a via de identificador j.

3.3.5. Cálculo de densidades

A densidade de veículos em cada via é calculada da mesma forma que em trabalhos relacionados [Salman 2018]:

$$D_i = \sum_{\forall j | p_{ji} > 0} \frac{V p_{ji}}{l_i N_i} \tag{4}$$

Sendo D_i a densidade de veículos na via i em veículos/km/faixa, V o número total de veículos na malha, p_{ji} a probabilidade do indivíduo transitar do estado j para o estado i, l_i a extensão da via em quilômetros e N_i o número de faixas na via.

Durante esse cálculo o número de faixas N_i é dividido pela metade para vias que são de mão duplas, considerando que cada sentido da via pode utilizar metade da sua capacidade.

Neste momento também é necessário supor o número total de veículos na malha V. Para tal é considerado que todas as vias inicialmente possuem um número de veículos suficiente para se encontrarem ainda em fluxo livre, sendo 7 veículos/km/faixa uma boa estimativa na área [Zegeer et al. 2008]. Pode-se obter então o valor suposto de veículos que populam a cadeia de Markov pela seguinte equação.

$$V = \sum_{i=0}^{n} 7 N_i l_i \tag{5}$$

Por fim, são obtidos os resultados do processamento da ferramenta, seja utilizado

os valores de velocidade máxima da OSM como velocidade média ou utilizando o tempo de travessia estimado proveniente da *Directions API*.

4. Resultados

Neste capítulo são apresentados a metodologia de teste e análise da ferramenta desenvolvida, os resultados obtidos de seu uso e discutido sobre a qualidade dos resultados obtidos.

5. Experimentos

Para os experimentos compilou-se os códigos-fonte em Rust e a partir do binário resultante e dos *scripts* em Python, foram realizadas execuções da aplicação para posterior análise. O equipamento utilizado para os experimentos possui as seguintes configurações:

- Processador: Ryzen 5 4650g
- Memória RAM: 2x8GB DDR4
- Sistema Operacional: Arch Linux

Pela falta de dados quantitativos de cadeias de Markov representativa de malha viária ou de perfil de densidades de tráfego optou-se por uma análise majoritariamente qualitativa dos resultados, justapondo os resultados obtidos utilizando a velocidade média como a velocidade máxima obtida pelo OSM com os resultados obtidos através do cálculo de velocidade média com base no tempo de travessia obtido pela *Directions API*.

Para testes durante o desenvolvimento foi escolhido o bairro José Mendes, Florianópolis, com apenas 81 segmentos de via. Nos experimentos foram realizadas execuções aproximadamente as 9 horas e as 18 horas de uma segunda-feira com o Centro de Florianópolis, contendo 920 segmentos de via. Também foram realizadas experimentos aproximadamente as 17 horas de uma terça-feira com o bairro do Brás, São Paulo, contendo 656 segmentos de via.

6. Discussão de resultados

O primeiro teste realizado foi feito com o Centro de Florianópolis. Após estimada a densidade de tráfego para a malha com ambas as fontes de dados previamente descritas, foi obtido um mapa de calor com as densidades seguindo uma separação pelos níveis de serviço segundo densidade segundo [Zegeer et al. 2008].

Pode-se observar que existe semelhança entre os resultados obtidos por ambas as fontes. Contudo, a partir da visualização das figuras 4 e 5 é notável alguns trechos de via possuem fortes diferenças entre as duas fontes. Para facilitar a visualização e análise foi utilizado um mapa de calor onde a densidade obtida com os dados da OSM é subtraída da densidade encontrada com dados da *Google Maps*.

Na Figura 6, de acordo com a legenda na imagem, tons de azul correspondem a trechos onde a densidade estimada utilizando apenas os dados do OSM possuem valores maiores que os encontrados utilizando dados da *Google Maps*. Tons em vermelho correspondem a trechos onde o valor de densidade estimada utilizando dados da *Google Maps* foram maiores que os valores obtidos utilizando apenas os dados da OSM. Por exemplo, se um trecho estimasse uma densidade de 6 veiculos/km/faixa utilizando apenas o

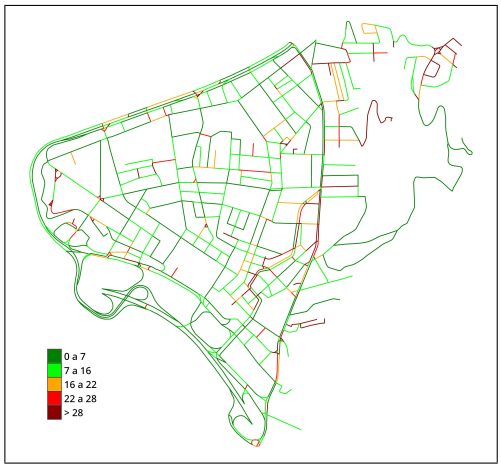

Figura 4. Mapa de calor de densidade estimada baseado em dados da OSM. Legenda em veiculos/km/faixa

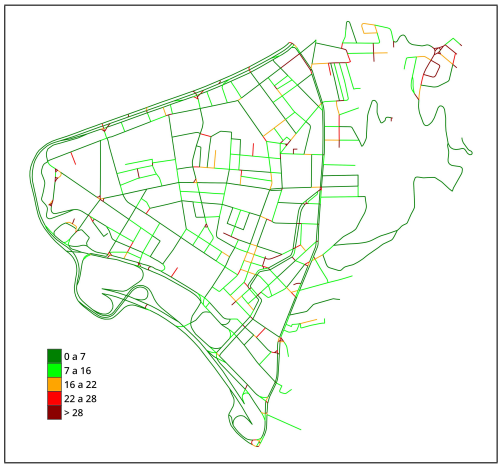

Figura 5. Mapa de calor de densidade estimada baseado em dados da *Google Maps*. Legenda em veiculos/km/faixa

OSM e estimasse 28 veiculos/km/faixa utilizando dados da *Google Maps*, o trecho estaria colorido com o tom mais vivo de vermelho.

Ao analisar a Figura 6 não é possível identificar padrão onde a OSM possui leituras de densidade maiores. Pelas características do cálculo de densidades nesse tipo de representação de cadeias de Markov (Equações 2, 3 e 4) e levando em consideração que vias coletoras tendem a possuir velocidades de tráfego maiores que arteriais [Reilly 1997], algumas dessas vias de caráter mais coletor podem ter leituras de densidade menores justamente pelas vias arteriais conectadas a ela possuírem velocidade média menor. Como hipótese, a estimativa fornecida atráves do uso da *Google Maps* representa melhor essa diferença em velocidade de uso da via, uma vez que a OSM considera que os usuários da rede utilizariam vias arteriais no limite de velocidade permitido.

Já os trechos onde os dados da *Google Maps* levam a acumular densidade recorrentemente se encontra em trechos pequenos de cruzamentos. Supondo que os resultados da *Google Maps* representam melhor a realidade que os da OSM, uma hipótese que pode ser levantada para tal efeito nos cruzamentos é que por conta de cruzamento exigirem dos indivíduos redução de velocidade, seja por conta de sinalizações ou para realizar curvas, os dados de *crowdsourcing* acumulam dados notavelmente menores de velocidade do que o máximo permitido na via.

Considerando que os serviços da *Google Maps* fornecem dados baseado no tráfego em tempo real foram feitas tentativas de encontrar variações no comportamento da malha viária em diferentes horários. Sendo a Figura 5 relativa a uma coleta de dados realizada em torno das 9 horas de uma terça-feira, também foram executados testes em torno das 18 horas do mesmo dias. Visualmente a figura obtida na leitura das 9 horas é idêntica à Figura 5. Apesar disso, leituras mais profundas nos dados obtidos mostram que em fato são resultados diferentes.

Por fim, sobre o custo financeiro do uso do serviço da *Directions API* neste trabalho, todos os gastos foram coberto pro créditos gratuitos oferecidos pela plataforma, sendo

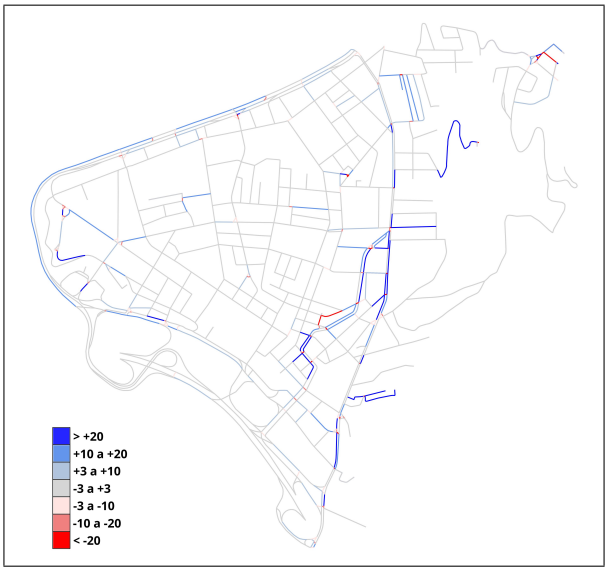

Figura 6. Diferença de densidades entre resultados da OSM vs *Google Maps*

um total de R$ 246,31 de créditos utilizados durante testes na fase de desenvolvimento e R$ 431,12 de créditos utilizados durante a execução dos experimentos.

7. Conclusão

Buscando explorar melhores alternativas para a estimativa de densidade de tráfego, o trabalho presente explora as possibilidades trazidas por serviços bem consolidados de estimativa de tempo de viagem como o *Google Maps*. Com resultados que representem melhor a realidade da malha viária em estudo, pesquisas e projetos de mobilidade urbana beneficiam-se ao partir de tais dados para seus experimentos.

A partir das análises realizadas pode-se concluir que a implementação da ferramenta é bem sucedida em processar dados em estimativas úteis para outros trabalhos que usam de cadeias de Markov para representação de tráfego. Apesar disso, fica evidente também de que necessita-se de mais dados para definir ou quantificar a melhoria proporcionada pelo uso de dados além dos provenientes do *OpenStreetMap* nesse tipo de estimativa.

Por fim, agradecemos o apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq, 405247/2023-0).

7.1. Trabalhos futuros

Considerando o trabalho desenvolvido, abaixo são listados alguns dos possíveis tópicos para trabalhos futuros:

- Utilização da *Routes API* ao invés da *Directions API* para requisições de tempo de viagem;
- Implementação de variação de número de veículos na malha de estudo;
- Implementação de novas metodologias de análise e validação de resultados;
- Revisitar trabalhos que utilizam técnicas similares de cadeias de Markov a partir do OSM e comparar resultados utilizando os dados da *Google Maps*;
- Implementação de conexão com *softwares* GIS.

Referências

Boeing, G. (2024). Modeling and analyzing urban networks and amenities with osmnx.

Caiza, L. J., Alvarez, R., Urquiza-Aguiar, L., Calderón-Hinojosa, X., and Zambrano, A. (2018). Vtm: Vehicular traffic monitor via images processing of googlemaps. In *Proceedings of the 15th ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks*, pages 40–46.

Churi, A. D. (2021). Google map traffic data scraping and mining. *no*, 4:891–895.

de Trânsito Senatran, S. N. (2022). Frota de veículos - 2022.

Galindo, Ernesto Pereira e Lima Neto, V. C. (2019). A mobilidade urbana no brasil: Percepções de sua população.

Google (2024). Directions api pricing.

He, Z., Chow, C.-Y., and Zhang, J.-D. (2019). A comparative analysis of journey time from google maps and intelligent transport system in hong kong. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 2610–2617. IEEE.

Junior, M. R., Medrano, R. A., and Cruvinel, K. (2018). O uso de sinais wi-fi para estimação de pares origem destino de usuários do transporte público em ônibus. In *XX Congreso Latinoamericano de Transporte Público y Urbano, At Medellín, Colombia*.

Mostafi, S. and Elgazzar, K. (2021). An open source tool to extract traffic data from google maps: Limitations and challenges. In *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–8.

Pelletier, M.-P., Trépanier, M., and Morency, C. (2011). Smart card data use in public transit: A literature review. *Transportation Research Part C: Emerging Technologies*, 19(4):557–568.

Reilly, W. (1997). *Highway capacity manual 2000*. Number 193.

Rezzouqi, H., Gryech, I., Sbihi, N., Ghogho, M., and Benbrahim, H. (2019). Analyzing the accuracy of historical average for urban traffic forecasting using google maps. In *Intelligent Systems and Applications: Proceedings of the 2018 Intelligent Systems Conference (IntelliSys) Volume 1*, pages 1145–1156. Springer.

Salman, Sinan; Alaswad, S. (2018). Alleviating road network congestion: Traffic pattern optimization using markov chain traffic assignment. *Computers & Operations Research*, 99.

Tahmasseby, S. (2015). Traffic data: Bluetooth sensors vs. crowdsourcing—a comparative study to calculate travel time reliability in calgary, alberta, canada. *Journal of Traffic and Transportation Engineering*, 3(2):63–79.

Zegeer, J. D., Blogg, M., Nguyen, K., and Vandehey, M. (2008). Default values for highway capacity and level-of-service analyses. *Transportation Research Record*, 2071(1):35–43.