Alandesson Linhares de Carvalho

Geração Automática de Artefatos para Suporte a Sistemas de Transporte Inteligentes

Aracaju/SE

2018, dezembro

Alandesson Linhares de Carvalho

Geração Automática de Artefatos para Suporte a Sistemas de Transporte Inteligentes

Trabalho de conclusão de curso apresentado na Universidade Tiradentes - UNIT, como requisito para a obtenção do título de Bacharel em Ciência da Computação sob Prof. Msc. Adolfo Guimarães.

Universidade Tiradentes - UNIT

Curso de Ciência da Computação

Programa de Graduação

Orientador: Adolfo Guimarães

Aracaju/SE

2018, dezembro

Alandesson Linhares de Carvalho
Geração Automática de Artefatos para Suporte a Sistemas de Transporte Inteligentes/ Alandesson Linhares de Carvalho. – Aracaju/SE, 2018, dezembro-
55 p. : il. (algumas color.) ; 30 cm.

Orientador: Adolfo Guimarães

Projeto de Conclusão de Curso – Universidade Tiradentes - UNIT
Curso de Ciência da Computação
Programa de Graduação, 2018, dezembro.

1. Semáforo Inteligente. 2. Simulação de Tráfego. 3. Visualização de cidades em 3D. 4. Processamento e Coleta de Mapas. I. MSc. Adolfo Guimarães. II. Universidade Tiradentes. III. Departamento de Computação. IV. Graduação

Alandesson Linhares de Carvalho

Geração Automática de Artefatos para Suporte a Sistemas de Transporte Inteligentes

Trabalho de conclusão de curso apresentado na Universidade Tiradentes - UNIT, como requisito para a obtenção do título de Bacharel em Ciência da Computação sob Prof. Msc. Adolfo Guimarães.

Trabalho aprovado. Aracaju/SE, 11 de dezembro de 2018:

Adolfo Guimarães
Orientador

Professor
Convidado 1

Professor
Convidado 2

Aracaju/SE
2018, dezembro

Agradecimentos

Agradeço aos meus pais Valda e Jorge, que não me deixaram ser vencido pelo cansaço. Sou profundamente grato ao meu irmão Alisson e minha irmã Aline pela assistência, paciência e motivação. Obrigado a todos os professores que contribuíram com a minha trajetória acadêmica, especialmente: ao professor Adolfo, responsável pela orientação do meu projeto; ao professor Virgílio, pela sua atenção, incentivos e ensinamentos; e ao professor Thiago, por toda ajuda e chances dadas durante minha graduação. Também agradeço meus amigos Leonardo, Raphael, Breno, Félix, Ézio, Daniel e todos que me deram apoio e incentivo para concluir este projeto.

"They say the universe is expanding.
That should help with the traffic.
(Steven Wright)

Resumo

O crescente número de veículos é uma preocupação das empresas de gerenciamento de tráfego. Esta preocupação pode ser justificada pela correlação do número de veículos com o aumento de acidentes rodoviários e congestionamentos. Infelizmente, muitas dessas empresas ainda usam métodos ultrapassados para gerenciar o trânsito, recorrendo a medições manuais e equipamentos não atuados.

Com o objetivo de melhorar o gerenciamento de tráfego, desenvolvemos um *software* modular multiplataforma, capaz de identificar rotas congestionadas e melhorar os ciclos do semáforo. O *software* foi implementado em Python e C++ usando as bibliotecas Urllib3, OpenCV e Scikit-Image para coletar e processar automaticamente dados de sistemas de informações geográficas (SIG). Com base nos dados interpretados, criamos um grafo, um *height map* e diversos *heat maps*. Além disso, através do motor gráfico da Unity, construímos um visualizador para renderizar a região escolhida em 3D.

Como estudo de caso, escolhemos a Farolândia, um bairro da cidade de Aracaju, para exibir as funcionalidade do nosso *software*. Esta região possui cerca de 9 km^2 de área, 3 cruzamentos com semáforos, vias de sentido único e duplo, uma rótula e diversos pontos de congestionamento. De acordo com nossos experimentos, nossa ferramenta demorou cerca de 33 minutos para gerar um *heat maps* de toda região. Portanto, acreditamos que nosso projeto poderá simplificar o trabalho de empresas de gerenciamento de trânsito, uma vez que somos capazes de detectar e visualizar pontos de congestionamento em qualquer horário, de forma automática e sem custos adicionais.

Palavras-chave: sistema inteligente de tráfego, semáforo inteligente, visualização de cidades em 3D, processamento e coleta de mapas.

Abstract

The rising number of vehicles is a concern of traffic management companies. This concern can be justified by the correlation of the number of vehicles with the increase of road accidents and congestion. Unfortunately, many of these companies still use outdated methods to manage traffic, using manual measurements and not actuated equipments.

In order to improve traffic management, we have developed a modular crossplatform software capable of identifying congested routes and improving traffic light's cycles. The software was implemented in Python and C++ using the Urllib3, OpenCV and Scikit-Image libraries to automatically collect and process geographic information systems (GIS) data. Based on the interpreted data, we created a graph, a height map and several heat maps. In addition, through the Unity graphics engine, we built a viewer to render the chosen region in 3D.

As a case study, we chose Farolândia, a neighborhood in the city of Aracaju, to test the functionality of our software. This region has about 9 km^2 of area, 3 intersections with traffic lights, one and two way routes, a roundabout and several points of congestion. According to our experiments, our tool took about 33 minutes to generate a heat map for the entire region. Therefore, we believe that our project can simplify the work of traffic management companies, since we are able to detect and view congestion points at any time, automatically and without additional cost.

Keywords: intelligent transportation system, smart traffic lights, 3D city view, map extraction and processing.

Lista de ilustrações

Lista de tabelas

Lista de algoritmos

Sumário

Introdução

Trânsito é um dos grandes problemas encontrados na maioria dos centros urbanos. Por exemplo, no Brasil, São Paulo é infamemente conhecida por seus diversos engarrafamentos. Segundo o Jornal El País (CORCOBADO, 2017), no dia 23 de maio de 2014, São Paulo entrou na quarta posição da lista dos piores engarrafamentos da história. Neste dia registrou-se um recorde de 344 quilômetros de vias congestionadas simultaneamente em toda a cidade.

Em 2017, na cidade de São Paulo, a CET-SP (Companhia de Engenharia de tráfego de São Paulo) registrou 13.483 acidentes e 16.252 vítimas feridas ou mortas(CETSP, 2017b). Analisando as estáticas de tráfego, podemos constatar que as vias com maior quantidade de acidentes são as mais engarrafadas. A Marginal Tietê, por exemplo, foi a via com maior número de acidentes. Em horários de pico, o tempo médio para a realização do percurso da Marginal Tietê, no sentido Castelo Branco - Ayrton Senna, chega a ser 2,26 vezes mais lento, demorando 21 minutos 49 segundos para percorrer 7,95 km (CETSP, 2017a).

Assim como em São Paulo, diversas cidades tem sofrido com problemas relacionados ao engarrafamento. Este problema motivou o desenvolvimento de diversos ramos de pesquisa, que objetivam melhorar a qualidade do trânsito. Um exemplo é área de sistemas de transporte inteligente, que tenta reduzir engarrafamentos, utilizando inteligência artificial em conjunto com sensores, câmeras, GPS, entre outras ferramentas. Esses sistemas de transporte inteligentes possuem diversas aplicações, incluindo sistemas de navegação, controle de semáforos, direção automática e avisos de segurança.

O uso de métodos de controle de fluxo no trânsito traz diversos benefícios, que aumentam a segurança do motorista e pedestres. Além de reduzir o congestionamento, identificar os obstáculos e colisões, sistemas de controle de fluxo podem ser utilizados para facilitar circulação de veículos de emergência, como ambulâncias e viaturas policiais.

Neste trabalho, apresentamos um sistema modularizado capaz de obter dados de fluxo com um baixo custo, que pode ser rapidamente implementado em qualquer região mapeada, sem adição de sensores. Essa aplicação utiliza informações de geolocalização públicas, para auxiliar métodos de otimizar o controle de semáforos. Dentre as principais vantagens do nosso método, temos: a capacidade de operar de forma independente ou em conjunto com outros métodos de controle de fluxo; a coleta e processamento é feito de forma automática; possibilita a reconstrução tridimensional dos prédios da área coletada e a simulação do fluxo de veículos, através de um visualizador de mapas; e por fim, a mais importante, podemos executar os módulos individualmente.

Os dados de geolocalização foram obtidos através de um sistema de informações geográficas (SIG). SIGs foram projetados para capturar, armazenar, manipular, analisar, gerenciar e disponibilizar diferentes tipos de informações geográficas. Bing Maps (MICROSOFT, 2018), Google Maps (GOOGLE, 2018a) e OpenStreetMap (HAKLAY; WEBER, 2008) são aplicações *web*, que permitem através de uma *Application Programming Interface* (API) acesso a essas informações. Mapas *web* oferecem mapa de ruas, imagens aéreas/satélite, geocodificação, buscas de endereços, busca de rotas e informações sobre engarrafamentos (TOSTES et al., 2013).

Este trabalho está dividido em três partes. Na Parte I discutimos alguns dos conceitos fundamentais e os trabalhos relacionados. Na Parte II discutimos a arquitetura do projeto, incluindo detalhes de implementação. Por fim, na Parte III, discutimos os resultados e estatísticas obtidas no projeto.

Parte I

Referenciais Teóricos

1 Gerenciamento de Tráfego

O tráfego urbano é um problema que cresce em complexidade diariamente. Portanto, técnicas para gerenciar diversos tipos de veículos e pedestres é algo necessário para o funcionamento da nossa sociedade. Neste capítulo, discutimos o uso e gerenciamento de semáforos para o controle do trânsito.

1.1 Tipos de Semáforos

Em 1913, James Hoge inventou o primeiro semáforo elétrico. Esta invenção se popularizou na década de 20 e aparece como sendo a origem do semáforo de três cores (vermelho, amarelo e verde)(HOMBURGER et al., 1992). Estes semáforos possuem tempo fixo e são utilizados para regular o trânsito de veículos nas interseções de vias e o fluxo de pedestres. Existem também semáforos conhecidos como intermitentes, que são utilizados para sinalizar perigo em lugares especiais, como saídas de garagem, vias que cruzam linhas de trem, etc.

Os primeiros semáforos com tempo variado, também conhecidos como semáforos atuados, só apareceram em 1928 em algumas cidades dos Estados Unidos, como New Haven, East Norwalk e Baltimore (HOMBURGER et al., 1992). Semáforos atuados podem possuir sensores de presença, botões para pedestres e outros dispositivos para regular o fluxo do trânsito. Este tipo de semáforo é o mais importante para o projeto, visto que precisamos que o semáforo permita a alteração do seu ciclo de acordo com o fluxo do trânsito.

1.2 Controle de Semáforos

O controle de semáforos de tempo fixo depende de fórmulas matemáticas para a busca de configurações ótimas. Estas fórmulas utilizam fluxo de carros por minuto, para estabelecer o tempo de cada semáforo. Esse tempo geralmente é fixado de acordo com o congestionamento da via de maior fluxo.

Diferente dos semáforos fixos, os semáforos atuados permitem alterar dinamicamente a duração dos seus ciclos. Dessa forma, diversas técnicas com base em redes neurais (LIU; LIU; CHEN, 2017), algoritmos genéticos (GHANIM; ABU-LEBDEH, 2015) e teoria dos jogos (KHANJARY, 2013) tem sido sugeridas como forma de otimizar o seu funcionamento.

Na tentativa de otimizar o ciclo de semáforo preventivamente, com base em cálculos de fluxos mais precisos, métodos de extração de informações de geolocalização de redes

veiculares e dispositivos, como celulares e GPSs, foram empregados.

Além das técnicas para otimização de semáforo e melhoria dos dados de fluxo. Existe uma técnica conhecida como sincronização[1], que pode ser aplicada em semáforos de tempo fixo e variado. Está técnica consiste em determinar ciclos ótimos para um conjunto de semáforo, permitindo que motoristas encontrem um maior número de sinais verdes, reduzindo a espera e evitando congestionamento (STANCIU; MOISE; NEMTOI, 2012). A coordenação de semáforos pode ser implementada a partir de Centros de Gerenciamento Tráfego (CGT). Nova York, Toronto e Singapura são exemplos de cidades que possuem CGTs, onde funcionários analisam e controlam diversos semáforos da cidade com base em observações obtidas através de câmeras, sensores e sugestões realizadas por *softwares* de otimização de trânsito.

[1] Também conhecida como coordenação

2 Processamento de Imagem

Nesta seção, apresentaremos as principais teorias para a implementação do módulo de processamento de imagem.

2.1 Segmentação

Segmentação é o processo que divide uma imagem em múltiplas regiões, a fim de destacar objetos facilitando o processamento da imagem. Quando está tarefa é realizada, os objetos separados passam até *pixels* similares aos do seu grupo.

Existem diversos métodos de se aplicar uma segmentação, como: *clustering*, que agrupa *pixels* com valores similares de acordo com uma métrica; histograma, que utiliza seus picos e vales para gerar grupos; e métodos matemáticos baseados em equações diferenciais parciais. No entanto, para este projeto escolhemos o método mais simples, conhecido como *thresholding*. O motivo desta escolha é esclarecido no Capitulo II.

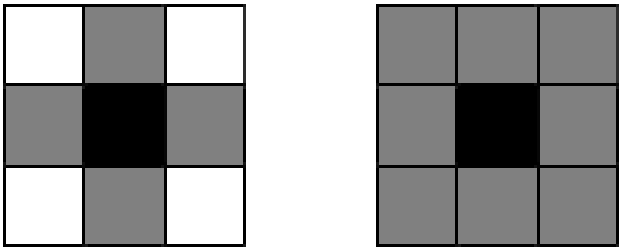
(a) Imagem de Lena Söderberg. (b) Imagem após *thresholding*.

Figura 1 – Exemplo de segmentação.

Como pode ser observado na Figura 1, o método *thresholding* binariza uma imagem com base em um valor limite. Este valor limite pode ser determinado de diversas maneiras, dentre as técnicas, mais comuns temos *clustering*, entropia e algoritmos genéticos. E em certos casos o valor pode até mesmo ser determinado arbitrariamente.

2.2 Vizinhança

Em processamento de imagem é comum utilizar as redondezas de um *pixel* para obter informações. Normalmente utiliza-se dois tipos de vizinhança denominadas de vizinhança-4 e vizinhança-8. Podemos ver na Figura 2 uma representação dos dois tipos de vizinhança. Onde preto representa o *pixel* e cinza os vizinhos.



(a) Vizinhança-4. (b) Vizinhança-8.

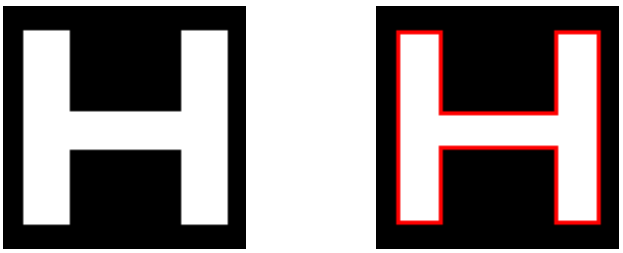
Figura 2 – Tipos de Vizinhança.

Neste projeto o tipo de vizinhança mais utilizado foi a vizinhança-8, pois para o problema em questão todos os *pixels* ao redor precisavam ser checados.

2.3 Bordas

Detecção de bordas é uma tarefa comum em projetos de processamento de imagem. No entanto, detectar bordas não é um problema simples em diversos casos. Para solucionar este problema diversas técnicas podem ser aplicadas, por exemplo: aplicação de gradientes, cálculos de taxa de variância *pixels*. No entanto este problema é simples para o nosso projeto, pois a imagem utilizada é uma imagem segmentada.

A detecção de bordas em imagens segmentadas é simples, pois a segmentação agrupa *pixels* tornando eles similares e facilitando distinção dos grupos.



(a) Carácter H. (b) Borda do carácter H.

Figura 3 – Exemplo de Borda.

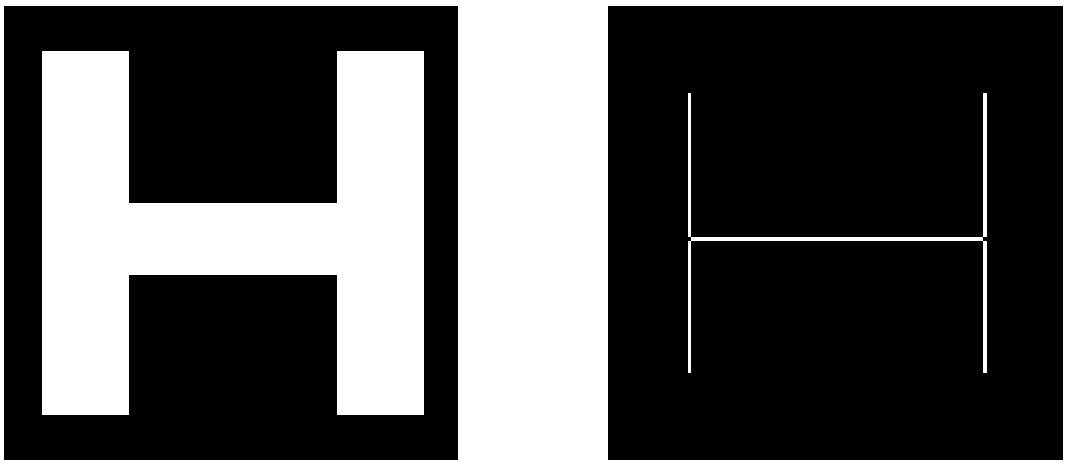
Na Figura 3 a borda pode ser facilmente identificada, pois a imagem utilizada é binária. Logo para encontrar a borda destacada em vermelho na Figura 3b, basta varrer a imagem verificando se a vizinhança contém um elemento, que não é similar ao *pixel* em análise.

2.4 Emagrecimento

Algoritmos de emagrecimento são comumente aplicado em programas de desenho e problemas de análise de tracejados. Este algoritmo auxilia técnicas de reconhecimento de

padrões possibilitando reconhecer objetos, caracteres e números através de um esqueleto representativo. (ZHANG; SUEN, 1984)

Um esqueleto representativo é obtido a partir da remoção de *pixels* da borda da imagem iterativamente, até que nenhum ponto possa ser removido sem alterar a conectividade do objeto. A remoção dos pontos podem ser realizadas de diferentes formas desde que a conectividade e estrutura do objeto seja mantida (ZHANG; SUEN, 1984).



(a) Antes do emagrecimento. (b) Após emagrecimento.

Figura 4 – Exemplo do funcionamento de um algoritmo de emagrecimento.

Na Figura 4 podemos ver o resultado do procedimento de emagrecimento aplicado no carácter H. Note que, um esqueleto gerado por este procedimento possui, na maioria dos casos, uma espessura de 1 *pixel*. Além disso, estas estruturas contam com três propriedades: preservação de conectividade; invariância em relação a operação de rotação e redimensionamento; e preservação de informação, que possibilita a reconstrução do objeto (VERNON, 1991). Por causa destas propriedades é possível garantir, que a utilização desta técnica irá simplificar a imagem sem causar prejuízos a solução de um dado problema.

3 Trabalhos Relacionados

Nesta seção, apresentaremos algumas da soluções mais populares. Assim como, alguns projetos de pesquisas e soluções que estão sendo implementadas atualmente no Brasil.

3.1 Soluções via Sensores

Diferente do nosso trabalho, os sistemas adaptativo de controle de tráfego mais utilizados dependem em grande parte de equipamentos como sensores de presença, câmeras e transmissores a rádio. Isso dificulta análises de trânsito, visto que muitas vezes só controlamos o fluxo nos pontos de congestionamento.

Entre os sistemas adaptativos mais populares, podemos citar o Split Cycle Offset Optimisation Technique (SCOOT)(TRL software, 2018), Sydney Coordinated Adaptive Traffic System (SCATS)(SCATS, 2012) e InSync(Rhythm Engeenering, 2017).

O SCOOT é uma solução para gerenciamento de tráfego urbano, desenvolvida em 1980 no Reino Unido, pela empresa TRL (Transport Research Laboratory). Este sistema, instalado em mais de 6.000 cruzamentos, utiliza sensores de indução para otimizar e sincronizar semáforos em tempo real. Em testes realizados, o SCOOT reduziu a duração de percursos em 15%, quando comparado com sistemas de tempo fixos em diversas cidades como São Paulo, Toronto e Londres(TRL software, 2018).

Similar ao SCOOT, o SCATS é um sistema de gerenciamento de tráfego por área em tempo real, desenvolvido em 1970 na Austrália, que ajusta e sincroniza ciclos de semáforos de acordo com variações no fluxo do tráfego - detectadas por sensores de indução em laço. Este sistema inteligente de trânsito, presente em aproximadamente 42.000 intersecções, demonstrou uma redução de 28% na duração de percursos de e 25% no tempo de espera em semáforos (SCATS, 2012).

Já o sistema de controle adaptativo InSync, desenvolvido pela empresa Rhythm Engineering, controla semáforos em tempo real com base no trânsito. Este sistema utiliza dados obtidos por diversos sensores como câmeras, sensores de presença e sensores de indução em laço, para realizar previsões precisas. Atualmente, o InSync possui um dos melhores resultados do mercado reduzindo a duração de percursos em 37% e o tempo de espera em semáforos em 68% (Rhythm Engeenering, 2017).

(LIU; LIU; CHEN, 2017) apresentou um projeto de otimização de semáforos, através de sensores de presença e câmeras, utilizando Multi-agent Q Learning. Está técnica de rede neural foi treinada para otimizar semáforos atuados, com base no fluxo de veículos,

pedestres e outros semáforos próximos. Este trabalho apresentou uma melhor performance que métodos similares, reduzindo filas de espera de pedestres e veículos. Entretanto, o projeto só leva em consideração o fluxo em semáforos, impedindo previsões mais precisas.

3.2 Soluções via SIG

Devido a popularização de dispositivos móveis e GPS, ferramentas que utilizam um SIG como fonte de dados ganharam notoriedade. Uma dos primeiros sistemas de controle de trânsito em tempo real através de SIG foi o Osiris.

Osiris é um protótipo, sugerido em 1999 e validado pelo Centro de Controle de Tráfego de Nottingham. Ele melhora diversas funções do sistema de gerenciamento de trânsito, especialmente visualização e funções relacionadas as condições presentes do tráfego de toda uma região. Além disso, devido ao conhecimento mais completo sobre os fluxos de uma região, o Osiris simplifica o gerenciamento e planejamento do tráfego (CLARAMUNT; PEYTCHEV; BARGIELA, 1999).

Similar ao nosso trabalho, (TOSTES et al., 2013) usou informações de controle de fluxo extraídas de um sistema de informação geográficas. No entanto, eles optaram por utilizar o sistema Bing Maps como fonte de dados.

Outra diferença foi a limitação da região, nesse trabalho os dados foram coletados a partir de rotas manualmente determinadas, limitando a previsão a somente as vias escolhidas pelo autor. Contudo, a técnica de regressão logística desenvolvida, mesmo com várias limitações, foi capaz de prever a situação do tráfego, com uma acurácia de 98%.

4 Ferramentas utilizadas

Nesta seção, descrevemos brevemente as principais bibliotecas e ferramentas, dando ênfase as vantagens e motivos da sua utilização ao longo do projeto.

4.1 Urllib3

A Urllib3(URLLIB3, 2006) é uma biblioteca disponível para Python, que permite realizar conexões HTTP e HTTPS de forma prática. Esta ferramenta, patrocinada pela plataforma Google Cloud e a empresa Hewlett Packard (HP), possui diversas vantagens como: *threads* seguras, verificação SSL/TLS, *uploads* em múltiplas partes com codificação, suporte de *proxy* para HTTP e SOCKS, tratamento para redirecionamento de requisições e diversos recursos críticos para aplicações com conexão *web*.

Utilizamos a Urllib3 versão 1.23 para realizar as requisições HTTPS feitas para as APIs usadas nos processos de coleta. Está ferramenta está disponível em: <https://github.com/urllib3/urllib3>.

4.2 Scikit-Image

O scikit-image(WALT et al., 2014) é uma biblioteca de processamento de imagem para Python, que implementa ferramentas e algoritmos utilizados para pesquisa, educação e aplicações industriais. O scikit e distribuído gratuitamente, sem nenhuma restrição, sob a licença *Modified BSD*.

Está biblioteca é mantida por um grupo de colaboradores ativos, que realizam melhorias continuas. A versão do scikit-image utilizada neste projeto foi a 0.14.1, disponibilizada em outubro de 2018 em: <https://github.com/scikit-image>.

4.3 OpenCV

OpenCV *(Open Source Computer Vision Library)*(OPENCV, 2000) é uma biblioteca escrita e otimizada em C++, disponibilizada sob a licença *BSD*. Ela é compatível com as linguagens C++, Python e Java e suporta aplicações desenvolvidas para Windows, Linux, Mac OS, iOS e Android.

Essa biblioteca foi projetada com foco em aplicações em tempo real. Logo, ela é extremamente eficiente, conseguindo se beneficiar de arquiteturas *multi-core*. Além disso,

ela permite o uso de OpenCL, para melhorar o desempenho de aplicações através de aceleração por *hardware*.

Neste projeto utilizamos a OpenCV em conjunto com Python e C++, para ler e processar imagens com grandes resoluções de forma eficiente. A versão utilizada foi 3.4.3 que está disponível em: <https://github.com/opencv/opencv/releases/tag/3.4.3>.

4.4 Unity

A Unity é um motor gráfico com múltiplas funcionalidades, que permite uma rápida edição e desenvolvimento de aplicações 2D e 3D. Esta ferramenta está disponível para Windows e Mac, e tem como foco criar aplicações gráficas multiplataformas (UNITY, 2018).

Utilizamos a Unity versão 2018.2.0f2, para simular o relevo e prédios de uma região, a fim de tornar a experiencia de visualizações do tráfego mais interativa. A Unity está disponível em: <https://store.unity.com>

Parte II

Arquitetura Proposta

5 Organização do Projeto

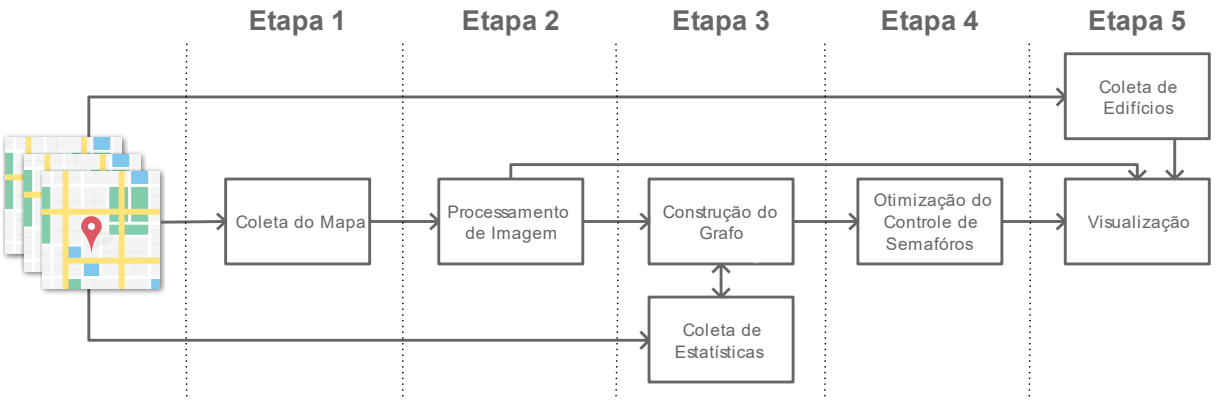


Figura 5 – Representação simplificada do funcionamento do projeto.

A Figura 5 descreve a arquitetura e funcionamento do projeto em etapas. Além disso, ela foi utilizada como base para organização deste capítulo.

Este capítulo foi divido em coleta de dados, processamento de imagem, construção do grafo, otimização do controle de semáforos e visualização. A seção de coleta de dados descreve todos os módulos de coleta vistos na Figura 5. As seções restantes descrevem os módulos com seus respectivos nomes.

5.1 Coleta de Dados

A coleta de dados é primeira etapa do projeto. Ela depende da extração de dados cartográficos de uma região escolhida pelo usuário. Nesta seção, discutiremos os diferentes tipos de coleta realizados ao longo do projeto.

5.1.1 Coleta do Mapa

O sucesso do desenvolvimento desse projeto depende da qualidade do mapa coletado. Qualquer ferramenta capaz de obter mapas com um nível adequado de detalhes pode ser utilizada. Dentre as ferramentas de coleta de mapas mais populares, podemos citar: Maps Static (GOOGLE, 2018b), Mapbox GL JS (MAPBOX, 2018), Bing Maps (MICROSOFT, 2018) e Yandex Maps (YANDEX, 2018).

Apesar de algumas limitações em relação ao número máximo de requisições e tamanho da imagem, escolhemos utilizar a Static Map API. Os principais motivos por trás dessa decisão, foram: a possibilidade de obter imagens com uma visualização mais próxima da região; cálculo de rotas mais precisas; ter uma melhor documentação; e o

mais importante, possui informações suficientes para converter *pixels* em coordenadas geográficas e vice-versa.

O Static Map suporta diversas linguagens de programação. Dentre as possíveis opções, optamos por Python. Escolhemos esta linguagem por simplificar a elaboração de requisições em HTTPS, com o uso da biblioteca Urllib3(URLLIB3, 2006). Além disso, o Python é compatível com a biblioteca scikit-image (WALT et al., 2014), que é uma ferramenta de processamento de imagem robusta [1].

Durante a construção do pedido, a fim de elevar a precisão da coleta, utilizamos um *zoom* de 20. O incremento deste valor garantiu a integridade da localização e a existência de todas as vias da região escolhida. No entanto, quando elevamos o *zoom*, diminuímos a área de visualização, pois o Static Map API limita o tamanho da imagem em 640 por 640 *pixels*. Como a resolução é limitada, acabamos obtendo apenas uma pequena parte da região desejada. Para contornar este problema, implementamos um algoritmo de conexão de imagens, com o objetivo de criar um mapa da região selecionada. A seguir, temos o pseudocódigo do algoritmo utilizado.

```
1  função requitarMapaGrande(lng,lat,zoomInicial,zoomFinal)
2      escala ← 2
3      divisões ← escala^(zoomFinal - zoomInical)
4
5      tamanhoLogo ← 22
6      larguraImagem ← 640
7      adicional ← tamanhoLogo/larguraImagem*escala^(zoomFinal - zoomInicial)
8
9      para i de -divisões/2 até divisões/2
10         para j de -divisões/2 até divisões/2 + adicional
11             novaLng ← converter pixel para longitude (lng,i*640)
12             novaLat ← converter pixel para latitude (lat,j*618)
13             imagem ← requesitar imagem de (novaLng,novaLat)
14             anexar imagem ao mapa
15     retornar mapa
```

Algoritmo 1 – Algoritmo de requisição do mapa

Observando o Algoritmo 1, temos os parâmetros que determinam um ponto inicial para a coleta das imagens: a latitude e a longitude do centro da região; um *zoom* inicial, suficiente para observar toda região desejada; e um *zoom* final, suficiente para visualizar as vias com maior nitidez.

A primeira ação do algoritmo é determinar o número de subimagens que formam o mapa. Com base na documentação do Google Maps (GOOGLE, 2018b), utilizamos um valor multiplicativo igual a 2 para a escala. Assim, podemos determinar que o número de

[1] Na seção 5.2 descrevemos seu uso com mais detalhes

partes necessárias para construir o eixo horizontal e vertical do mapa é $2^{zoomFinal-zoomInicial}$ - visto que a resolução dos eixos é a mesma.

É importante notar que houve um adicional de $\lceil 22/640 * 2^{zoomFinal-zoomInicial} \rceil$ ao número de imagens verticais. Este valor foi usado para remover as marcas d'água deixadas pela API do Google em cada requisição de uma subimagem - pois a logo atrapalharia o processamento do mapa.

Na Figura 6, podemos observar o funcionamento do algoritmo descrito em 1. Coletamos as imagens horizontalmente, pulando na coordenada longitudinal o equivalente a 640 *pixels*. Ao coletar a primeira linha de imagens, incrementamos o equivalente a 618 *pixels* na coordenada latitudinal. Precisamos usar o valor de 618, ao invés de 640, porque cobrimos 22 *pixels* da parte inferior de cada subimagem, para esconder a logo do Google.

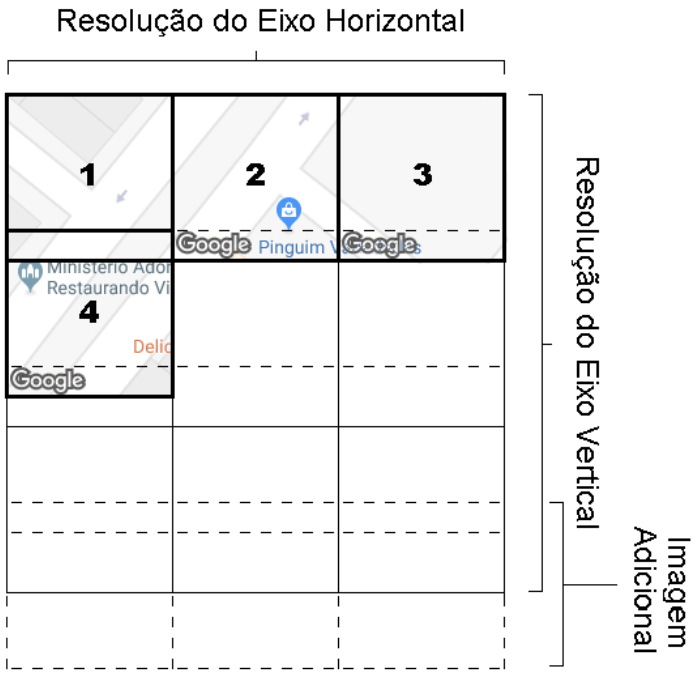


Figura 6 – Geração do mapa.

5.1.2 Coleta de Edifícios

Com o objetivo de projetar um cenário tridimensional, estudamos diversas ferramentas capazes de extrair informações das edificações. Dentre as mais importantes, podemos citar: Mapbox GL JS (MAPBOX, 2018), Mapme (MAPME, 2018) e Osmbuildings (OPENSTREETMAPS, 2018).

Estas ferramentas, apresentam edificações extremamente detalhadas. No entanto, elas estão disponíveis somente em grandes cidades dos Estados Unidos e em algumas cidades famosas da Europa. Como pretendíamos produzir uma solução genérica, atendendo múltiplas regiões, preferimos utilizar a API Maps Static.

Diferente das outras ferramentas, o Maps Static não possui informações de altura. Contudo, ele possui uma maior quantidade informações de largura, comprimento e posições. Dessa forma, optamos por estimar a altura e as informações não são disponibilizadas pela API.

Primeiramente, aplicamos um filtro para remover todos os elementos da imagem, com exceção dos edifícios. Em seguida, extraímos a localização de prédios e casas da região. Com base nessas informações, a área dos edifícios é calcula usando uma função de contorno, conhecida como Border Following (SUZUKI; ABE, 1985), implementada na OpenCV3 (OPENCV, 2000). Está função recebe como parâmetro uma imagem binária e gera como saída uma lista de contornos, de onde podemos obter a área em pixels.

A altura dos prédios foi calculada usando uma função geradora, que recebe como parâmetro a área das edificações. Contudo, a fim de obter um cenário mais diversificado, adicionamos ao resultado da função geradora um valor aleatório - dentro de uma margem configurável. Na seção 5.5, discutimos a ferramenta de visualização com mais detalhes.

5.1.3 Coleta de Estatísticas

Nesta etapa, coletamos estatísticas de tempo e distância, utilizando a API Distance Matrix (GOOGLE, 2018a). Esta ferramenta, disponibilizada pelo Google, trabalha a partir de requisições HTTPS. Similar ao Static Map, a Distance Matrix API suporta diversas linguagens de programação. Dessa forma, optamos por manter o uso da linguagem Python.

Os motivos que nos levou a escolher Python, foram: a simplicidade oferecida na realização de requisições HTTPS; a facilidade de armazenar e ler estruturas de dados complexas, por meio do uso da biblioteca pickle e das funções nativas do Python; e não menos importante, manter a padronização com a implementação da coleta do mapa.

Assim como a maioria das APIs do Google, a Distance Matrix possui limites de requisições. Com o intuito de fazer um melhor uso da cota disponibilizada, garantindo a diminuição do número de chamadas da API, tomamos a precaução de colocar em cada requisição um ponto de origem e todos seus destinos.

Cada chamada da API retorna um arquivo no formato json. Este arquivo possui campos contendo a distância e a duração dos percussos, entre cada ponto inicial até todos os pontos finais. Estes valores serão utilizados pelos algoritmos de otimização, discutidos na seção 5.4.

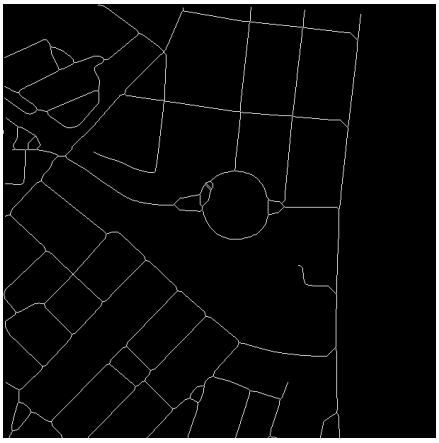
Os pontos utilizados nesta coleta foram determinados durante a construção do grafo. Portanto, a explicação de como foram obtidos será descrito na seção 5.3.

5.2 Processamento de Imagem

(a) Mapa Original.

(b) Mapa após Edição.

(c) Mapa após Binarização.



(d) Mapa após Emagrecimento.

Figura 7 – Etapas do processamento da imagem coletada em 5.1.1.

Durante o processamento do mapa utilizado no projeto, aplicou-se três métodos importantes. Primeiramente, o mapa passou por um processo de edição, em que todo conteúdo irrelevante foi removido, como pode ser visualizado na Figura 7b. Nessa etapa, removemos: os nomes das ruas, avenidas e rodovias; marcadores; nomes de prédios; representação de edifícios, parques, rios e outros elementos que atrapalhavam a visualização das vias. Essa remoção foi realizada com a API Maps Static através da função de estilização do mapa.

Na segunda etapa, que pode ser visualizada na Figura 7c, aplicamos uma segmentação com um limite 250 no canal verde do RGB. O método de segmentação utilizado resulta em uma binarização, no qual o branco representa as vias e o preto o restante da imagem. O limite foi calculado com base na coloração das ruas, usando uma pequena margem de erro para garantir a conectividade das vias.

Por fim, a última etapa pode ser visualizada na Figura 7d. Nesta etapa, aplicamos

um método paralelo de emagrecimento, disponibilizado pela scikit-image (WALT et al., 2014). Esse método é uma operação iterativa aplicada na vizinhança de um objeto de uma imagem segmentada e resulta em um esqueleto representativo (ZHANG; SUEN, 1984). O esqueleto obtido através deste processo é adequado para representar um objeto, visto que ele exibe três importantes propriedades: preservação de conectividade; invariância em relação a operação de rotação e redimensionamento; e preservação de informação, que possibilita a reconstrução do objeto (VERNON, 1991).

A aplicação do algoritmo de emagrecimento ofereceu uma melhor representação do mapa, mantendo a sua forma. Este processo foi necessário para simplificar a identificação de interseções de vias, auxiliando a construção do grafo.

5.3 Construção do Grafo

O grafo é construído com base em uma imagem pré-processada de um mapa. Uma região pode conter avenidas, rotatórias, ruas sem saída e vias com mão única e dupla. Logo, para caracterizar a região selecionada, optamos por uma representação de um grafo conexo e direcionado, com uma estrutura de lista de adjacência. Escolhemos esta estrutura, pois a lista de adjacência apresenta um menor consumo de memória, quando um grafo contém uma grande quantidade de vértices e poucas arestas.

Como Python foi utilizado em etapas anterior, mantivemos o seu uso para leitura e escrita de dados, remoção dos elementos desconexos e formação do grafo. Infelizmente, a tarefa de busca de todos os vértices e arestas do grafo demonstrou-se bastante ineficiente, chegando a demorar várias horas, ao ser realizada em Python. Esta circunstância pode ser atribuída ao fato de Python ser uma linguagem interpretada e por não haver uma biblioteca otimizada capaz de realizar a tarefa de detecção dos vértices e arestas, compatíveis com o mapa criado.

Para acelerar a detecção dos vértices e arestas, criamos em Python um subprocesso que executa um programa escrito em C++ - previamente compilado utilizando O3 para uma melhor otimização. A este subprocesso, delegamos a tarefa de analisar o mapa. Portanto, passamos como parâmetro de entrada, uma cadeia de caracteres descrevendo o caminho para um arquivo contendo a imagem que será analisada.

O subprocesso faz uma varredura em busca de regiões brancas na imagem. Para cada ponto encontrado, calculamos o total de pontos brancos contidos na sua vizinhança-8. A vizinhança-8 consiste em todos os pontos ao redor do pixel em análise, incluindo as diagonais. Se o valor obtido nessa operação for superior a 2, sabemos que aquele ponto é um vértice do grafo. É importante notar, que este resultado só é possível graças ao algoritmo de emagrecimento, pois ele faz com que ruas sejam representadas com a espessura de 1 pixel. Logo, se um ponto qualquer possuí mais de 2 *pixels* brancos ao seu redor, significa

que este ponto está localizado em uma intersecção de vias.

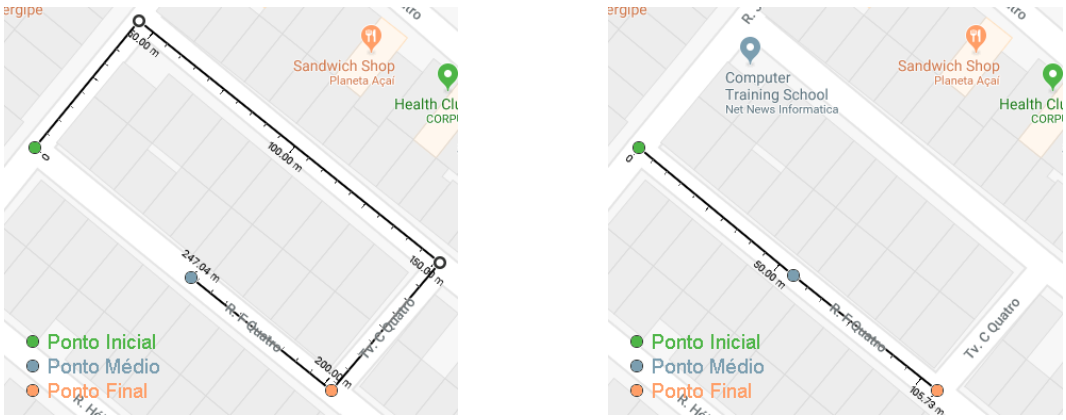
Após encontrarmos todos os vértices, partimos para o processo de determinação das arestas. Esse processo é dividido em duas etapas:

1. Determinar as possíveis arestas, através da imagem utilizando C++;
2. Verificar a existência de cada aresta encontrada, utilizando Python e a API Distance Matrix;

Na primeira etapa, para cada elemento da lista de vértice, navegamos na imagem seguindo somente as vias conectadas ao vértice em análise, até encontrar um outro elemento contido na lista. Durante essa navegação, empilhamos o caminho. Assim, quando encontramos um ponto contido na lista de vértices, podemos extrair o início, meio e o fim de cada rota. Ao final desta etapa, armazenamos em disco a lista de vértices e arestas em um caminho predeterminado.

O início e o fim é tudo que precisamos para determinar a ligação de uma aresta. No entanto, um ponto médio foi adicionado. Este ponto é importante para validar a existência da rota, visto que a conexão entre dois vértices nem sempre é válida, já que existem vias com mão única.

Para validar aresta, verificamos se a distância do ponto inicial até o ponto médio é menor do que a distância do ponto inicial até o ponto final. O uso do ponto médio pode ser melhor compreendido através da imagem 8.



(a) Aresta não existe. (b) Aresta existe.

Figura 8 – Possíveis percursos para o ponto médio.

Na segunda etapa, lemos os dados que foram armazenados em disco pelo subprocesso e recriamos as listas de arestas e vértices. Em seguida, realizamos a coleta de tempo e distância como descrito na seção 5.1.3, utilizando os pontos da lista de arestas. Concluímos o processo ao realizar a validação das arestas, com a técnica do ponto médio descrita anteriormente.

O tempo médio para a construção do grafo é um dado interessante a ser comentado. Após implementação do subprocesso em C++, obtivemos uma melhoria em performance em cerca de 200 vezes, com um tempo médio de execução reduzido para cerca de 5 minutos.

5.4 Controle de Semáforo

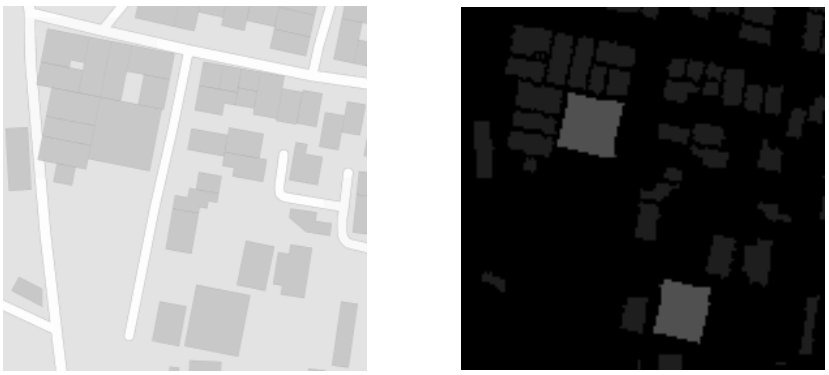
Com o objetivo de verificar a possibilidade de controlar um semáforo com base na ferramente desenvolvida um método simples de controle de semáforos foi implementado, este método necessita que o usuário determine o par de aresta que possuem os semáforos. O fluxo deste par de aresta é utilizado para determinar o tempo dos ciclos do semáforo, dentro de um intervalo de 20 a 100 segundos.

Para determinar o tempo, utilizamos a soma do fluxo de cada aresta para determinar um fluxo total. Em seguida, calculamos a porcentagem do fluxo de cada aresta em relação ao fluxo total. E finalmente, multiplicamos cada porcentagem por 80 e adicionamos 20, para obter o tempo de cada ciclo dentro do intervalo escolhido.

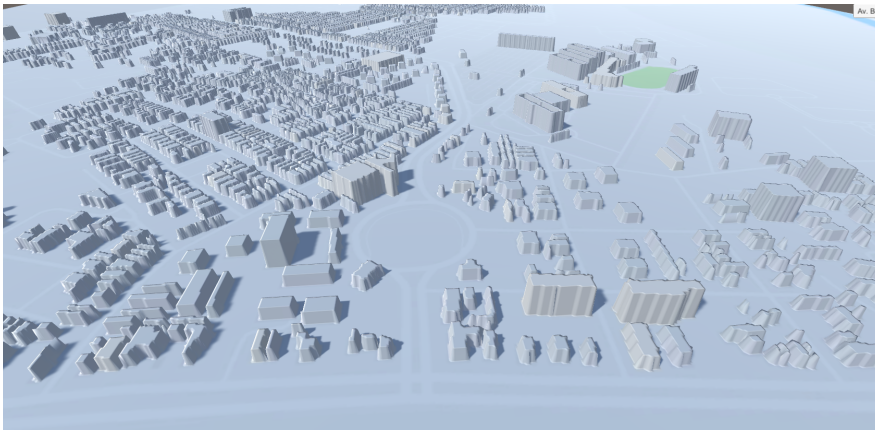
5.5 Visualização

Com o objetivo de visualizar a região e a aplicação dos algoritmos, sobre grafo coletado. Desenvolvemos um simulador que renderiza tridimensionalmente os principais edifícios da região. Esse simulador foi desenvolvido usando a Unity3D(UNITY, 2018) e recebe como parâmetros de entrada a imagem coletada em 5.1.2 e o grafo otimizado em 5.4.

Escolhemos a Unity3D, pois ela possuí nativamente uma função de geração de relevo. Esta função utiliza uma imagem em escala de cinza, conhecida como heightmap, para gerar terrenos tridimensionais. Dessa forma, convertemos a imagem obtida na etapa de coleta de edifícios em um heightmap compatível com Unity3D.



(a) Imagem Original. (b) *Height Map.*



(c) Edifícios rederizados pela Unity3D.

Figura 9 – Etapas de renderização 3D.

Na Figura 9a podemos observar uma região do estado de Sergipe. Em 9b podemos ver o heightmap, desta mesma região após o processamento. Em 9c utilizamos o heightmap apresentado em 9b para renderizar os prédios.

Para navegar dentro do visualizador, utilizamos uma câmera que sobrevoa a região mapeada. Essa câmera pode se mover de forma guiada, saltando entre pontos importantes listados no visualizador; ou livre, utilizando comandos do usuário. Durante a navegação, poderemos ver três marcadores ao longo das rotas. Os marcadores representaram o fluxo de veículos, em que vermelho simboliza alto, amarelo médio e verde baixo.

Parte III

Resultados

6 Estudo de Caso

Para testar o funcionamento do projeto, escolhemos uma área de aproximadamente 9 km², com centro nas coordenadas -10,9709826 e -37,0651128, que inclui a Farolândia e suas proximidades. A imagem resultante tem a resolução de 20480 por 20480 e possui vias de mão única e dupla, assim como diversos semáforos.

Todo processo de extração, processamento, construção e execução demora em torno de 30 minutos. Contudo, esse tempo pode variar dependendo das configurações de cada máquina. Além disso, é importante notar que o tempo de extração depende da velocidade da conexão utilizada.

Etapa		**Tempo Médio (s)**	**Consumo de Memória (MB)**	**Consumo do Disco**
Extração		535.26514	1711	10MB
Processamento		105.34288	1320	3.7MB
Construção	**C++**	3,48645	1205	103KB
	Python	1333,99407	32	66KB por Coleta

Tabela 1 – Resultado da execução da aplicação.

A Tabela 1 descreve o tempo médio, o consumo de memória e o consumo de disco de cada etapa do processo. Esses resultados foram obtidos usando um Notebook Lenovo G40, com as seguintes configurações: sistema operacional Windows 10 Pro; 8GB de memória RAM DDR3M, com frequência de 1600Mhz; processador Intel I5-5200U com *clock* de 2.2GHz; HD de 1TB com 8MB de cache e 5400 RPM; placa de vídeo AMD Radeon R5 M230 com 2GB de memória DDR3; e uma conexão de 35MB provida pela NET.

O funcionamento do projeto também foi verificado na região do centro da cidade de Aracaju, nas coordenadas -10.9126921 e -37.0544503. Obtivemos imagens do mapa e o grafo da região, com uma resolução de 5120 por 5120. Abaixo podemos a Figura 10 com uma imagem do mapa depois do processo de segmentação.

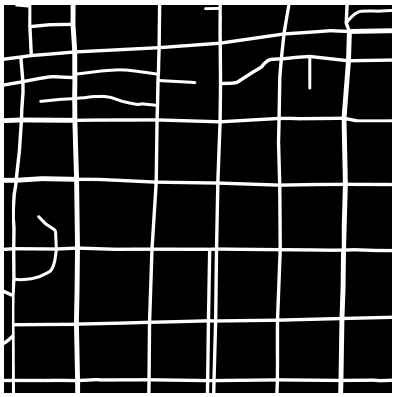


Figura 10 – Mapa do centro da cidade de Aracaju segmentada.

6.1 Contribuições

No escopo deste projeto, podemos destacar três grandes contribuições: a primeira foi a implementação de uma ferramenta para construção de grafos a partir da imagem de um mapa. Com esta ferramenta, grafos de cidades podem ser construídos em alguns minutos; a segunda contribuição foi a geração de um mapa de aquecimento, que informa a variação do congestionamento de todas as vias de uma região; por fim, a nossa terceira contribuição é um método para otimizar semáforos utilizando estatísticas de fluxo. Nas próximas seções discutiremos essas contribuições com mais detalhes.

6.1.1 Grafo da Cidade

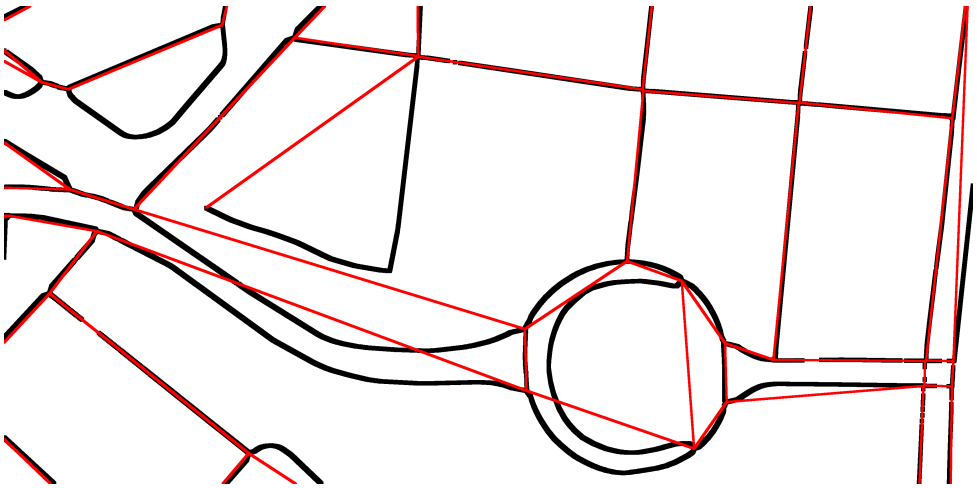


Figura 11 – Visualização de arestas do Grafo.

No estudo de caso, geramos um grafo de toda a região da Farolândia. Este grafo possuí 963 vértices e 2290 arestas. Quando plotada, a imagem resultante tem uma resolução de 20480 por 20480 *pixels* e consome aproximadamente 3.5MB de disco.

Na Figura 11 podemos visualizar o grafo, plotado em vermelho, sobre um segmento da região. O grafo é armazenado em disco de forma serializada e pode ser facilmente distribuído e recuperado. Além disso, os grafos gerados podem ser utilizados para diferentes estudos, como: busca de rotas, estimativa de fluxos, caracterização das vias e cruzamentos, etc.

6.1.2 Mapa de Aquecimento

O mapa de aquecimento é uma forma de visualizar o congestionamento de vias. Com a implementação dessa ferramenta, podemos visualizar o aquecimento de todas as vias de uma região com informações atuais.

Para coletar as estimativas de tempo de uma via, utilizamos um horário informado pelo usuário, o tráfego atual e o histórico do tráfego na via. A estimativa é realizada de tal

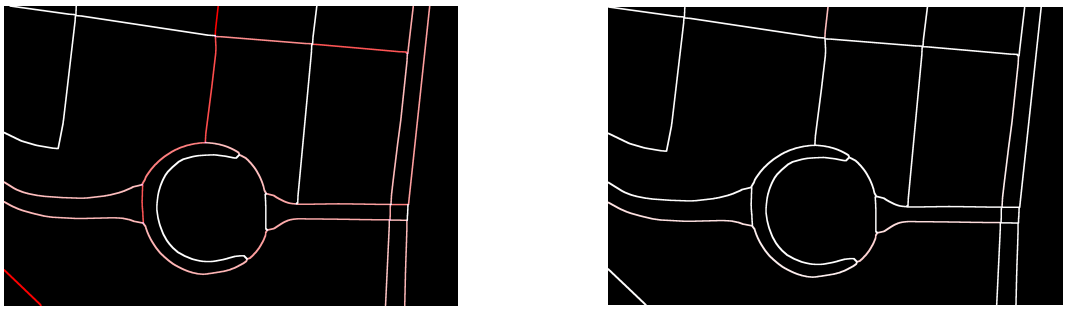
forma que a relevância do histórico do tráfego é reduzida quanto mais próximo o horário informado é do horário real.

Para determinar o nível de aquecimento, utilizamos três tipos estimativas de fluxo fornecidas pela API Distance Matrix (GOOGLE, 2018a): otimista (Optimistic), pessimista (Pessimistic) e a melhor suposição (Best guess). A estimativa otimista normalmente retorna um valor menor que o real. A melhor suposição tenta se aproximar da realidade. Já a estimativa pessimista retorna um valor de tempo alto, mas que pode ser superado em caso de um congestionamento do trânsito.

Escolhemos representar o aquecimento no mapa variando da cor branca à vermelha. Para calcular o valor da escala, adotamos o tempo T do percurso como $\min(pe, \max(ot, ms))$, onde ot é o tempo otimista, ms o tempo da melhor suposição e pe o tempo pessimista. Abaixo, temos equação utilizada para determinar o valor da escala de vermelho:

$$EscalaVermelho = \frac{254 * (T - ot) + pe - T}{pe - ot} \tag{6.1}$$

Note que escala vária entre 1 e 254, pois durante a implementação reservamos os valores 0 e 255. Além disso, podemos constatar que quanto mais distante T está do ótimo maior será a escala de vermelho. Na Figura 12 é possível visualizar o resultado do congestionamento na rotatória da Farolândia, em uma terça-feira às 18:50h. Já Na Figura 12 podemos ver o mesmo trecho, em um domingo, às 23:59h.



(a) Aquecimento de uma terça-feira às 18:50h. (b) Aquecimento de um domingo às 23:59h.

Figura 12 – Visualização do aquecimento.

As linhas em vermelho, como observadas na Figura 12a, representam o nível de engarrafamento no horário da coleta. As linhas mais brancas, como as vistas na Figura 12b, demonstram uma baixa alteração no fluxo de veículos.

7 Conclusão

Neste trabalho apresentamos uma infraestrutura para otimização, visualização e detecção de congestionamentos. Nosso *design* modular permite a reutilização das ferramentas em diversas aplicações, incluindo geração de grafos de cidades, busca de caminhos, criação de cenários para jogos, controle de tráfego, otimização de vias, etc.

Dentre as contribuições, podemos destacar a capacidade de gerar de mapas de aquecimento. Esta funcionalidade poderá automatizar os trabalhos realizados por empresas de controle de tráfego. Uma vez que diversas empresas no Brasil ainda realizam coletas manuais - como é o caso da CET-SP em que os pesquisadores coletam dados através de equipamentos de GPS, trafegando nas vias em determinados horários (CETSP, 2017a).

Diferente das principais alternativas disponíveis no mercado. A nossa metodologia utiliza bases de dados públicas, reduzindo os custos de implementação. Além disso, nossa infraestrutura pode ser integrada com mecanismos mais sofisticados de coleta, incluindo sensores e ferramentas com estatísticas em tempo real como o Waze (GOOGLE, 2018c).

Todos os detalhes do projeto, incluindo as imagens geradas, os módulos da aplicação, as listas de nós e arestas, o grafo serializado com a biblioteca Pickle e a ferramenta de visualização, podem ser encontrado em: <https://www.github.com/alandesson/its>.

7.1 Trabalhos Futuros

No decorrer deste projeto, percebemos diversas aplicações para os módulos implementados. Entre as mais importantes, podemos citar:

1. **Previsão de Fluxo**: utilizar as estatísticas obtidas para o treinamento de uma rede neural, que poderá prever fluxo do tráfego em qualquer momento do dia.
2. **Sugestão de Posição de Semáforos**: utilizar o grau dos vértices e estimativas de fluxo, para sugerir pontos, onde semáforos poderiam melhorar o fluxo do trânsito.
3. **Detecção Automática de Semáforos**: realizar a disposição dos semáforos automaticamente, com base na API Open Street Maps (HAKLAY; WEBER, 2008) e a gerar a logica de conexões dos semáforos sem entrada manual.
4. **Otimizar Funcionamento de Semáforos Atuados**: utilizar métodos de teoria dos jogos para otimizar os semáforos em intersecções (KHANJARY, 2013).
5. **Melhorar as Estatísticas em Tempo Real**: podemos combinar os resultados do mapa de aquecimento e do grafo com outras fontes de estatísticas em tempo real,

como o Waze (GOOGLE, 2018c), a Traffic API da empresa Here(HERE, 2018) ou o Bing Maps (MICROSOFT, 2018). Dessa forma, podemos melhorar a detecção de anomalias no trânsito, alterando o comportamento dos semáforos em tempo real.

6. **Simular o Trânsito**: Importar os dados obtidos pela ferramenta desenvolvida, no simulador de trafego urbano SUMO(KRAJZEWICZ et al., 2012).

Referências

CETSP. Pesquisa de monitoração da mobilidade. *Companhia de Engenharia de Tráfego*, 2017. Disponível em: <http://www.cetsp.com.br/media/714822/msvp-2017-volume-e-velocidade.pdf>. Acesso em: 13 out. 2018. Citado 2 vezes nas páginas 19 e 51.

CETSP. Relatório anual de acidentes de trânsito. *Companhia de Engenharia de Tráfego*, 2017. Disponível em: <www.cetsp.com.br/media/646657/relatorioanualacidentestransito-2017.pdf>. Acesso em: 13 out. 2018. Citado na página 19.

CLARAMUNT, C.; PEYTCHEV, E.; BARGIELA, A. A real-time gis for the analysis of a traffic system. v. 1, p. 15–20, Set 1999. Disponível em: <https://ieeexplore.ieee.org/document/820669>. Citado na página 30.

CORCOBADO, M. Trânsito de são paulo está na lista dos cinco maiores congestionamentos da história. *El País*, Maio 2017. Disponível em: <https://brasil.elpais.com/brasil/2017/05/08/internacional/1494262753_775936.html>. Citado na página 19.

GHANIM, M. S.; ABU-LEBDEH, G. Real-time dynamic transit signal priority optimization for coordinated traffic networks using genetic algorithms and artificial neural networks. *Journal of Intelligent Transportation Systems*, Taylor Francis, v. 19, n. 4, p. 327–338, 2015. Disponível em: <https://doi.org/10.1080/15472450.2014.936292>. Citado na página 23.

GOOGLE. *Distance Matrix API*. 2018. Disponível em: <https://developers.google.com/maps/documentation/distance-matrix/start>. Acesso em: 10 out. 2018. Citado 3 vezes nas páginas 20, 38 e 49.

GOOGLE. *Maps Static API*. 2018. Disponível em: <https://developers.google.com/maps/documentation/maps-static/intro>. Acesso em: 10 out. 2018. Citado 2 vezes nas páginas 35 e 36.

GOOGLE. *Waze Sdk*. 2018. Disponível em: <https://www.waze.com/sdk>. Acesso em: 24 out. 2018. Citado 2 vezes nas páginas 51 e 52.

HAKLAY, M. M.; WEBER, P. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 7, n. 4, p. 12–18, out. 2008. ISSN 1536-1268. Disponível em: <https://doi.org/10.1109/MPRV.2008.80>. Citado 2 vezes nas páginas 20 e 51.

HERE. *Traffic API*. 2018. Disponível em: <https://developer.here.com/documentation/traffic/topics/incident-data.html>. Acesso em: 24 out. 2018. Citado na página 52.

HOMBURGER, W. et al. *Fundamentals of Traffic Engineering*. Institute of Transportation Studies, University of California, Berkeley, 1992. (Course notes). Disponível em: <https://books.google.com.br/books?id=wXonAQAAMAAJ>. Citado na página 23.

KHANJARY, M. Using game theory to optimize traffic light of an intersection. IEEE, p. 249–253, Novembro 2013. Disponível em: <https://doi.org/10.1109/CINTI.2013.6705201>. Citado 2 vezes nas páginas 23 e 51.

KRAJZEWICZ, D. et al. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, v. 5, n. 3&4, p. 128–138, December 2012. Citado na página 52.

LIU, Y.; LIU, L.; CHEN, W.-P. Intelligent traffic light control using distributed multi-agent q learning. *2017 IEEE 20th International Conference on Intelligent Transportation Systems*, nov 2017. Disponível em: <https://arxiv.org/pdf/1711.10941.pdf>. Citado 2 vezes nas páginas 23 e 29.

MAPBOX. *Mapbox GL JS API*. 2018. Disponível em: <https://www.mapbox.com/mapbox-gl-js/api/>. Acesso em: 10 out. 2018. Citado 2 vezes nas páginas 35 e 37.

MAPME. *Mapme API*. 2018. Disponível em: <https://mapme.com/support/knowledgebase/>. Acesso em: 10 out. 2018. Citado na página 37.

MICROSOFT. *Bing Maps API*. 2018. Disponível em: <https://www.microsoft.com/en-us/maps/documentation>. Acesso em: 10 out. 2018. Citado 3 vezes nas páginas 20, 35 e 52.

OPENCV. *OpenCV Library*. 2000. <https://opencv.org/>. Acesso em 11 out. 2018. Citado 2 vezes nas páginas 31 e 38.

OPENSTREETMAPS. *OSM Buildings API*. 2018. Disponível em: <https://osmbuildings.org/documentation/>. Acesso em: 10 out. 2018. Citado na página 37.

Rhythm Engeenering. *InSync: The Most Widely-Deployed Adaptive Traffic Control Signal System in the U.S.* 2017. Disponível em: <https://rhythmtraffic.com/insync-2/>. Acesso em: 3 nov. 2018. Citado na página 29.

SCATS. *SCATS 6 Functional Description*. [S.l.], 2012. Disponível em: <https://www.scats.com.au/files/an_introduction_to_scats_6.pdf>. Acesso em: 1 nov. 2018. Citado na página 29.

STANCIU, E. A.; MOISE, I. M.; NEMTOI, L. M. Optimization of urban road traffic in intelligent transport systems. *Annalen der Physik*, IEEE, p. 4, 2012. Citado na página 24.

SUZUKI, S.; ABE, K. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, v. 30, n. 1, p. 32–46, 1985. Disponível em: <http://dblp.uni-trier.de/db/journals/cvgip/cvgip30.html#SuzukiA85>. Citado na página 38.

TOSTES, A. I. J. et al. From data to knowledge: City-wide traffic flows analysis and prediction using bing maps. In: *Proceedings of the 2Nd ACM SIGKDD International Workshop on Urban Computing*. New York, NY, USA: ACM, 2013. (UrbComp '13), p. 12:1–12:8. ISBN 978-1-4503-2331-4. Disponível em: <http://doi.acm.org/10.1145/2505821.2505831>. Citado 2 vezes nas páginas 20 e 30.

TRL software. *SCOOT: Adaptive Traffic Control System*. Transport Research Laboratory, 2018. Disponível em: <https://trlsoftware.com/wp-content/uploads/2018/08/SCOOT.pdf>. Acesso em: 3 nov. 2018. Citado na página 29.

UNITY. *Unity User Manual (2018.2)*. 2018. Acesso em 3 nov. 2018. Disponível em: <https://docs.unity3d.com/Manual/index.html>. Citado 2 vezes nas páginas 32 e 42.

URLLIB3. *HTTP Client Urllib3*. 2006. Acesso em 29 out. 2018. Disponível em: <https://urllib3.readthedocs.io/en/latest/>. Citado 2 vezes nas páginas 31 e 36.

VERNON, D. *Machine vision: Automated Visual Inspection and Robot Vision*. Prentice Hall, 1991. Disponível em: <http://homepages.inf.ed.ac.uk/rbf/BOOKS/VERNON/vernon.htm>. Citado 2 vezes nas páginas 27 e 40.

WALT, S. van der et al. scikit-image: image processing in Python. *PeerJ*, v. 2, p. e453, 6 2014. ISSN 2167-8359. Disponível em: <http://dx.doi.org/10.7717/peerj.453>. Citado 3 vezes nas páginas 31, 36 e 40.

YANDEX. *Yandex.Maps JavaScript API*. 2018. Disponível em: <https://tech.yandex.com/maps/doc/jsapi/2.1/quick-start/index-docpage/>. Acesso em: 10 out. 2018. Citado na página 35.

ZHANG, T. Y.; SUEN, C. Y. A fast parallel algorithm for thinning digital patterns. *Commun. ACM*, ACM, New York, NY, USA, v. 27, n. 3, p. 236–239, mar. 1984. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/357994.358023>. Citado 2 vezes nas páginas 27 e 40.