CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GESTÃO DE CONGESTIONAMENTOS E VEÍCULOS DE EMERGÊNCIA UTILIZANDO KNN EM SEMÁFOROS INTELIGENTES

MARINA DOS SANTOS BERETTA

ORIENTADOR: PROF. ME. RODOLFO BARROS CHIARAMONTE

Marília - SP
Dezembro/2017

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GESTÃO DE CONGESTIONAMENTOS E VEÍCULOS DE EMERGÊNCIA UTILIZANDO KNN EM SEMÁFOROS INTELIGENTES

MARINA DOS SANTOS BERETTA

Monografia apresentada ao Centro Universitário Eurípides de Marília como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.
Orientador: Prof. Me. Rodolfo Barros Chiaramonte

Marília - SP
Dezembro/2017

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
MANTIDO PELA FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Marina dos Santos Beretta

Gestão de congestionamentos e veículos de emergência utilizando KNN em semáforos inteligentes

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 10,0 (Dez)

Orientador:Rodolfo Barros Chiaramonte

1º.Examinador:Mauricio Duarte

2º.Examinador:Luan Cardoso dos Santos

Marília, 28 de novembro de 2017.

RESUMO

A má administração do trânsito é uma das maiores causas de acidentes, atrasos, gasto de combustível e emissão de carbono, devido à grande quantidade de veículos existentes. Todos esses pontos resultam em prejuízos nas áreas da saúde e de finanças. A criação de novas vias, pelo contrário do que muitos pensam, pode resultar no aumento desse tráfego, como mostra o paradoxo de Braess. Com o objetivo de minimizar esses problemas, o projeto desenvolvido otimiza essa administração através do controle dinâmico das fases dos semáforos, isto é, a fase de cada semáforo é definida de acordo com o número de veículos, suas prioridades, para veículos de emergência, por exemplo, e tempo de espera. Utilizando o algoritmo KNN para realização deste controle, o projeto obteve melhora nos tempos de percurso dos veículos em 100% dos testes realizados, ainda que com a priorização dos veículos de emergência. Para execução dos testes e validação do algoritmo desenvolvido, foram utilizadas as ferramentas *OpenStreetMap* para captação dos dados e SUMO para simulação do trânsito. Foi utilizado um cenário real e muito movimentado do centro da cidade de Marília, no interior do estado de São Paulo, Brasil, garantindo, assim, a veracidade dos resultados e a capacidade do projeto.

Palavras-chaves: Gestão de Trânsito, Semáforo Inteligente, Prioridade de veículos, veículos de emergência, Congestionamento, KNN, SUMO.

ABSTRACT

The bad traffic management is one of the biggest causes of accidents, delays, fuel expense and carbon emissions due to the large number of existing vehicles. All of these points result in losses in health and finance. The creation of new avenues may result in increased traffic, as shown by the Braess paradox. In order to minimize these problems, the developed project optimizes this management through the dynamic control of the traffic lights' phases, i.e., the phase of each traffic light is defined according to the number of vehicles, their priorities, for emergency vehicles for example, and the travel time. Using the KNN algorithm to perform this control, the project obtained an improvement in the travel times of the vehicles in 100% of the performed tests, even with the prioritization of emergency vehicles. To perform the tests and validation of the developed algorithm, it was used the tools OpenStreetMap for data capture and SUMO for road traffic simulation. It was used a real and very busy scenario of the center of the city of Marília, inside the state of São Paulo, Brazil, thus guaranteeing the veracity of the results and the capacity of the project.

***Keywords**: Traffic Management, Inteligent Traffic Lights, Vehicle Priority, Emergency Vehicles, Traffic Jam, KNN, SUMO.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CETRAN	Conselho Estadual de Trânsito
CONTRAN	COnselho Nacional de TRÂnsito
CTB	Código de Trânsito Brasileiro
DENATRAN	DEpartamento NAcional de TRÂNsito
DETRAN	Departamento Estadual de Trânsito
GPS	Sistema de Posicionamento Global
IVT	*Institute for Transport Planning and Systems*
KNN	*K-Nearest Neighbors*
MANET	*Mobile Ad hoc NETwork*
MATSim	*Multi-Agent Transport Simulation*
OMNet++	*Objective Modular Network testbed in C++*
OSM	*OpenStreetMap*
OSMF	*OpenStreetMap Foundation*
RFID	*Radio-Frequency IDentification*
SNT	Sistema Nacional de Trânsito
SO	Sistemas Operacionais
SUMO	*Simulation of Urban MObility*
TCP	Protocolo de Controle de Transmissão
TraCI	Interface de Controle de Tráfego
VANET	*Veicular Ad hoc NETworks*
WSN	*Wireless Sensor Networks*

SUMÁRIO

Capítulo 1
INTRODUÇÃO

Este capítulo abordará os detalhes sobre a elaboração deste projeto, apresentando itens como a sua importância para a sociedade, para futuros trabalhos na área e os passos de seu desenvolvimento.

1.1 Motivação

Pode-se perceber que o número de veículos tem crescido constantemente. Segundo DETRAN (2016), nos últimos 10 anos, o número de carros nas grandes cidades brasileiras aumentou 150%, aproximadamente. Esse grande fluxo de automóveis gera longos congestionamentos, que, de modo controverso, impossibilita o uso de apenas um veículo por família, pois cada pessoa tomará um rumo diferente e perderá muito tempo se desviar de seu caminho, aumentando ainda mais esse número.

Grande parte do tempo dos condutores no trânsito é gasto na espera pela abertura dos semáforos, muitas vezes com tempos que não correspondem à necessidade do fluxo de veículos na determinada hora. O estresse causado por esta demora pode resultar em problemas de saúde, pela ansiedade dos condutores para continuar seu caminho, e financeiros, pelo atraso desses em seus compromissos.

Uma opção para desviar deste problema é utilizar meios alternativos de transporte, como metrôs, ônibus e trens. Porém, eles não suportam o número de pessoas que optam por ele e acabam ficando superlotados.

Outra saída, muito requisitada pela população, é a criação de novas vias para distribuir o fluxo de veículos. Porém, como mostra ERIC e SHARI (1997), em pouco tempo essas vias ficarão tomadas por veículos e não conseguirão escoá-los para as vias antigas, que já estavam congestionadas. Isso pois os condutores procurarão sempre o caminho mais rápido para eles, não se preocuparão com a distribuição geral de carros, gerando um novo congestionamento.

Sendo assim, é preciso focar em gerar novos e melhores meios de gerenciar as vias já existentes, tirando o melhor proveito delas e economizando fortunas que seriam utilizadas em obras e sinalizações, além de aproveitar melhor o tempo e preservar a paciência do condutor.

A fim de focar nesses objetivos, este projeto pretende controlar os semáforos de modo que os tempos correspondam ao necessitado nos respectivos horário e local, economizando, ao máximo, o tempo de espera dos condutores e possibilitando que estes usem seu tempo para suas atividades, ao invés de esperarem parados no trânsito.

Além disso, o semáforo será capaz de gerenciar automaticamente os tempos, eliminando as grandes centrais de monitoramento de sinais e sobretudo, os erros humanos, sempre possíveis de acontecer.

Sendo direcionado também para os veículos de emergência, o projeto melhorará o atendimento desses a seus chamados. As viaturas terão preferência aos demais automóveis, permitindo que cheguem ao local necessário a tempo de realizar o atendimento.

Com os pontos citados, conclui-se que a proposta vai além do simples controle dos semáforos, sendo focada na melhoria da qualidade de vida da população. Uma vez que haverá economia no tempo gasto em transporte, diminuição no estresse causado pelo trânsito e rapidez no atendimento das emergências.

1.2 Problema de Pesquisa

Com o desenvolvimento cada vez mais acelerado das cidades, diversos problemas são criados pelo fluxo de pessoas superior ao suportado por suas estruturas. Um desses problemas é o congestionamento, que atrasa a vida da maioria da população, gerando um desconforto que muitas vezes pode ser evitado ou, ao menos, minimizado.

Além disso, há situações em que a má administração do trânsito pode causar consequências piores, como emergências (de incêndios, acidentes, roubos) não atendidas a tempo, agravando os problemas de saúde, violência, financeiros e outros.

Conforme dados divulgados pelo DETRAN (Departamento Estadual de Trânsito) de São Paulo[1], entre os fatores que contribuem para o aumento de acidentes de trânsito estão a inexistência de semáforo e o desrespeito dos condutores à ele.

Na mesma divulgação, segundo dados do Ministério da Saúde, no Brasil morrem aproximadamente 43 mil pessoas por ano nesses acidentes, tornando essa uma

[1] <https://www.sp-detran.org/estatisticas-acidentes-no-transito-em-sao-paulo/>

das principais causas de morte no país. Em 2014, o custo de cada acidente foi de R$ 72.705,31 e, para acidentes envolvendo vítimas fatais, esse valor sube para R$ 646.762,94.

O projeto desenvolvido realiza o controle dos semáforos de forma eficiente, ou seja, de acordo com a necessidade do fluxo de veículos, resultando na diminuição de acidentes e, por consequência, do número de óbitos do país.

1.3 Justificativa

A mobilidade urbana vem se tornando um assunto muito preocupante e frequentemente abordado pelos governos e pela população em grandes eventos por todo o mundo. Isto porque os problemas de trânsito crescem cada vez mais rápidos e não são simples de serem resolvidos por impactarem muitas pessoas com diferentes comportamentos nesse ambiente.

Buscando respeitar esses costumes, as medidas a serem tomadas precisam ser dinâmicas e se moldar à necessidade de cada população. Esse é o foco desse trabalho, que se preocupa em oferecer a melhor gerência do fluxo de veículos existentes nas cidades.

1.4 Objetivos Gerais e Específicos

O objetivo desse trabalho é gerar uma solução de controle dos semáforos, a fim de melhorar, ao máximo, o gerenciamento de veículos, evitando tempos de espera desnecessários. Além disso, é capaz de identificar os veículos de emergência e dar-lhes prioridade nos cruzamentos, para que trafeguem com mais rapidez e segurança.

Para tal fim, será desenvolvido um algoritmo capaz de analisar o fluxo de cada via, a prioridade e o tempo de espera de cada veículo e, assim, definir as fases do semáforo que melhor gerencie esses dados. Sendo os objetivos específicos:

- Determinar e simular um cenário real de trânsito;
- Analisar o comportamento do tráfego no cenário;
- Desenvolver um algoritmo de controle das fases do semáforo;
- Integrar esse algoritmo à simulação do cenário.

1.5 Metodologia de Desenvolvimento

O processo de desenvolvimento do projeto foi dividido em etapas, para maior organização e controle de seu avanço, sendo a ordem e definição das etapas:

I Estudar trabalhos relacionados ao tema, levantando suas principais características;

II Estudar as ferramentas à serem utilizadas *OpenStreetMap*, SUMO (*Simulation of Urban MObility*) e MATSim (*Multi-Agent Transport Simulation*);

III Escolher ferramenta para simulação do tráfego (SUMO ou MATSim);

IV Definir o cenário real a ser estudado e simulado;

V Retirar as informações sobre o cenário e incluí-las na ferramenta de simulação;

VI Estudar os algoritmos de aprendizado de máquina e escolher o melhor a ser utilizado;

VII Implementar o algoritmo para realizar o controle dos semáforos, analisando as informações do cenário;

VIII Simular o controle de semáforos utilizando o algoritmo desenvolvido.

1.6 Organização do trabalho

Este projeto está organizado em capítulos conforme divisão das suas fases de estudo e elaboração. A ordenação e o conteúdo presente em cada capítulo são:

O presente capítulo apresenta uma introdução ao projeto, descrevendo sua importância para a sociedade, as intenções de seu desenvolvimento e os passos seguidos em sua realização.

O Capítulo 2 apresenta os conceitos estudados para fundamentar o planejamento do projeto. Além das ferramentas e dos principais algoritmos estudados e avaliados para escolha do que melhor auxilia a construção deste.

O Capítulo 3 exibe trabalhos de outros autores cujos temas se relacionam com a proposta deste, descrevendo seu funcionamento e destacando seus pontos positivos e negativos.

O Capítulo 4 possui detalhes sobre o processo de definição do cenário utilizado para validação do projeto, além dos passos seguidos para sua preparação e simulação.

No Capítulo 5 encontra-se os passos seguidos para implementação do projeto, além de explicações mais detalhadas sobre o algoritmo utilizado no controle das fases dos semáforos.

Os resultados obtidos na validação do projeto estão no Capítulo 6, com detalhes sobre cada simulação e a melhoria atingida no gerenciamento dos veículos. Além disso, é apresentada a análise desses resultados, para comprovação da eficiência do algoritmo desenvolvido.

A conclusão do projeto, presente no Capítulo 7, apresenta os pontos mais marcantes do desenvolvimento deste projeto, juntamente com os possíveis trabalhos a serem desenvolvidos a partir deste.

Capítulo 2

CONCEITOS E FERRAMENTAS PARA SIMULAÇÃO DE TRÂNSITO

Neste capítulo, são apresentados e discutidos os principais assuntos sobre o tema deste projeto, estudados e avaliados para a fundamentação de seu planejamento, e as ferramentas a serem utilizadas, para a captação dos dados e simulação do cenário determinado.

2.1 Conceitos básicos

Nesta seção serão apresentados alguns itens básicos para garantir o completo entendimento desse projeto.

2.1.1 Código de Trânsito Brasileiro

O Código de Trânsito Brasileiro (CTB) é descrito pela Lei 9.503, de 23/09/97, que entrou em vigor no dia 22 de janeiro de 1998 e se aplica a quaisquer vias terrestres do território brasileiro. Essa lei é a garantia da segurança no trânsito, pois é de responsabilidade das entidades e órgãos do Sistema Nacional de Trânsito (SNT) defender a vida, incluindo preservar a saúde e o meio-ambiente.

> "Considera-se trânsito a utilização das vias por pessoas, veículos e animais, isolados ou em grupos, conduzidos ou não, para fins de circulação, parada, estacionamento e operação de carga ou descarga.", (Art. 1º, § 1º)

2.1.2 Semáforos de tempo fixo

Os chamados semáforos de tempo fixos são os mais comuns nas ruas do Brasil. Esses possuem tempos de duração pré-programados para a abertura de cada

fase (verde, amarelo e vermelho), o que não permite que os tempos se adequem ao fluxo de veículos da via.

Um meio de melhorar os tempos deste semáforo para que gerencie o trânsito de forma mais eficiente é observar o fluxo de veículos do determinado cruzamento por um período de tempo grande e definir manualmente os novos tempos que se satisfaçam melhor a necessidade da área. Porém, este método ainda não permite que o semáforo funcione de maneira dinâmica, ou seja, com tempos de fases diferentes para cada fluxo.

Caso haja um tempo desnecessário de espera do veículo no semáforo, por não haver veículos transitando no cruzamento neste momento, é possível que o veículo à espera avance o sinal vermelho por impaciência ou descrença da necessidade do semáforo nesta situação. Essa imprudência gera uma situação de risco, pois pode surgir um automóvel na via transversal sem que aquele veículo perceba sua passagem e freie a tempo de não atingí-lo.

2.1.3 Semáforos Inteligentes

Os semáforos inteligentes são desenvolvidos para resolver o problema citado na subseção 2.1.2. Eles controlam a duração de cada fase do semáforo de acordo com a necessidade da via, ou seja, conforme informações captadas sobre o local.

Pode-se avaliar quais dados sobre as vias, o cruzamento, os veículos, o semáforo ou qualquer outro objeto que possa influenciar o tempo necessário para o controle correto do semáforo. Assim, é possível obter diversos modelos de semáforos inteligentes, sendo mais fácil de encontrar o que melhor satisfaça o cruzamento à ser implementado.

Utilizando um semáforo deste tipo, o tempo gasto à espera da fase verde do semáforo será o menor possível. Isso diminuirá o gasto desnecessário de combustível, o trânsito formado nos semáforos, o stress causado por isso, os desrespeitos ao semáforo, os acidentes causados por eles, e muitos outros fatores que interferem na economia e na qualidade de vida da população.

O projeto desenvolvido se encaixa neste tipo de aplicação, já que utiliza as informações reais dos veículos em tempo real para determinar a fase a ser acionada no semáforo.

2.1.4 Mapeamento urbano

O mapeamento urbano é a ação de criar mapas digitais sobre uma determinada área urbana com informações sobre o local. Essas informações podem ser de todos os níveis e sobre qualquer objeto que componha o cenário, como velocidade

média dos veículos, tipos de vegetação, últimos edifícios construídos, quantidade de pedestres, poluição do ar, profundidade dos buracos de uma via e muitos outros.

Para captar essas informações podem ser usados sensores, imagens de satélites ou imagens de câmeras. Pode-se também recolher informações de forma manual, distribuindo pessoas por toda a área para observar, analisar e anotar as características que se pretende obter.

Thapa e Murayama (2009) destacam diversas possíveis aplicações para essa prática, como medir o crescimento urbano, como o solo é usado, estimativa da população, monitoramento da temperatura da superfície terrestre, qualidade do ar e vegetação, mapeamento topográfico e monitorar as atividades humanas na superfície.

2.2 Captação de informações do cenário

Para o desenvolver de um semáforo inteligente é necessário obter informações sobre o cenário determinado para a implantação deste. Elas podem ser captadas da forma mais conveniente para o implementador do controle do semáforo.

Algumas das formas existentes de captação de informações sobre o trânsito serão apresentadas nesta seção, além de seus pontos positivos e negativos.

2.2.1 Sensores

É possível efetuar a contagem de carros presentes na via de diversas formas, uma delas é utilizando sensores. Porém, existem vários tipos de sensores que efetuam essa função.

Os sensores térmicos detectam a temperatura dos objetos, não ficando sensíveis à emissão de luzes direta. Sendo assim, funcionam muito bem com uma forte luz solar, faróis altos, objetos localizados em sombras e no período noturno.

Outra opção seria utilizar os sensores magnéticos, que detectam a presença de objetos metálicos ao seu redor, captando o campo magnético gerado pela proximidade desses, estando o objeto em movimento ou não. Para detectar a presença dos veículos, pode-se implantar esses sensores debaixo do asfalto, assim eles ficariam seguros contra roubos e ainda captariam os automóveis.

Há ainda o RFID, um termo que define qualquer tecnologia que utilize frequência de rádio para passagem de informações. Um sistema RFID é composto por um transponder (ou RF Tag), onde estão guardadas as informações a serem transmitidas, e um leitor, que receberá essas informações e que também pode ser um transmissor. Quando o transponder entra na zona de alcance do leitor e os dois estão na mesma frequência, esse detecta e emite um sinal àquele para que transfira suas informações.

Então, o transponder envia seus dados através de campos magnéticos, que serão capturados e decodificados pelo leitor e então enviados a um dispositivo que fará o processamento deles.

Além dessas opções, há o Sistema de Posicionamento Global (GPS), composto por vários satélites que orbitam a Terra emitindo sinais de rádio que são detectados pelos sensores GPS instalados nos diversos dispositivos conhecidos, como celulares e veículos. Esses recebem os sinais dos satélites mais próximos e, sabendo a posição e a área de alcance de cada um, calcula a área de intersecção entre elas, que será onde se localiza o dispositivo. Assim, se cada veículo possuir um sensor, é possível ter conhecimento da posição de cada um nas vias.

A utilização dos sensores é incentivada pela necessidade de manutenção ser quase nula. Porém, seu custo é elevado e dependendo do tamanho da área em que será feita o controle de veículos, seu custo-benefício torna seu uso inviável.

2.2.2 *Veicular Ad Hoc Network*

A VANET (*Veicular Ad Hoc Network*) é uma subcategoria de MANET (*Mobile Ad hoc NETwork*). Essa por sua vez é definida por Boukerche (2008) como um conjunto de estações (nodos) comunicando através de canais sem fio, sem qualquer suporte principal fixo. Nela, os dispositivos servem como roteadores e transmitem as informações de um para o outro, até que chegue à seu destino.

Por serem destinadas à veículos, as VANETs funcionam para conexão tanto entre veículos como entre veículo e via. Assim, podem transmitir dados sobre a situação do carro e seus componentes como das vias e possíveis acidentes, por exemplo.

O alerta de acidentes e a prevenção deles são as principais aplicações que utilizam essa rede. Isso porque nas estradas não há muitos métodos disponíveis de comunicação, pela falta de estrutura nessa área. Além disso, a transmissão das informações diretamente entre os veículos e as vias é ideal para a resposta rápida dos condutores ou dos próprios veículos, caso possuam algum sistema de inteligência.

A facilidade de distribuição rápida das informações feita por esse mecanismo ainda não garante total segurança contra usuários mal intencionados que tentem invadir a rede. Esses acessos podem ser usados, por exemplo, para implantar informações falsas sobre o trânsito e causar tomadas de decisões erradas dos condutores, que resultarão em acidentes reais.

2.2.3 Câmeras

A utilização de câmeras para controle do trânsito vêm crescendo muito nos últimos anos. Já em 18/12/13, o Conselho Nacional de Trânsito (CONTRAN) elaborou

a Resolução nº 471, que regulamenta a utilização de câmeras para aplicar multas de trânsito em condutores imprudentes.

As imagens captadas por elas precisam ser processadas para distinguir os veículos dos demais objetos presentes. A qualidade desse processamento depende de fatores como uma boa condição de luz captada, por exemplo, da luz solar e dos faróis dos carros. A luz, tanto natural quanto artificial, emitida diretamente em direção à lente da câmera causa uma luminosidade na imagem que os sensores ópticos, por exemplo, seriam ofuscados. Essa luminosidade atrapalha o processamento das informações da imagem, gerando uma avaliação do trânsito muito diferentes da situação real.

Por outro lado, esta opção possui melhor custo-benefício e fácil implementação e manutenção. As pessoas estão mais acostumadas com este dispositivo, o que causaria menos estranhamento e resistência da população a eles. Também pelo costume com as câmeras, há várias empresas que oferecem instalação e manutenção de forma rápida, prática, segura e confiável.

2.2.4 *Simulation of Urban MObility* (SUMO)

SUMO[1] é um programa de código aberto e sem custo, capaz de simular veículos, pedestres e transporte público. Seu principal objetivo é a simulação de trafego urbano, por isso ele será apresentado com mais detalhes na seção 2.3. Porém, ele foi utilizado neste projeto para a captação das informações sobre os veículos presentes nas vias.

Este programa fornece diversos dados[2] sobre os veículos, as vias, os semáforos, os pedrestres e até sobre as rotas dos veículos. Sua praticidade o tornou o melhor meio de captação no desenvolvimento, pois este projeto não visa definir uma única forma de realizar a coleta dessas informações, mas sim contribui para que o controle dos semáforos desenvolvido possa servir para qualquer projeto em quaisquer circunstâncias, de maneira prática.

Sendo assim, os códigos de captação dessas informações estão totalmente separados dos códigos de processamento delas, para que o segundo possa ser integrado à outros projetos de forma independente.

2.2.5 *OpenStreetMap*

O OpenStreetMap[3] (OSM) é um projeto que reúne informações de todo o mundo sobre estradas, plantações, estações ferroviárias, entre outras.

[2] <http://www.sumo.dlr.de/pydoc/traci.html>
[3] <http://www.openstreetmap.org>

Os dados são inseridos e revisados por uma comunidade de mapeadores voluntários, que buscam informações em fontes abertas, como imagens de satélites e dados de GPS, ou mesmo pela observação pessoal. O projeto também possui muitas parcerias com universidades, empresas de hospedagem, servidores, provedores de internet, centros de pesquisas de diversas partes do planeta que contribuem com hospedagem, *hardware* ou tempo para ajudar a manter o projeto atualizado e confiável.

Steve Coast iniciou o projeto em Julho de 2004 e logo em Abril de 2006, criou-se uma fundação, a OpenStreetMap Foundation (OSMF) para incentivar a geração de dados abertos sobre mapeamento geoespacial e a distribuição livre desses para quem quiser utilizá-los. Em Julho de 2007 ocorreu a 1ª *The State of the Map*, uma conferência internacional anual, que acontece até os anos atuais, onde discutiu-se esses assuntos através de palestras e *workshops*.

Esta ferramenta foi utilizada no projeto para obter informações sobre as vias terrestres do cenário determinado, como a própria existência e localização das vias, cruzamentos e semáforos presentes na área.

2.3 Ferramentas de simulação de tráfego urbano

Existem diversos *softwares* que auxiliam a execução e validação de uma simulação de trânsito. Para escolher a mais adequada ao projeto, é necessário avaliar as necessidades deste e a ferramenta que melhor as atende.

Este tópico apresenta as ferramentas avaliadas para este projeto, suas características e a escolhida para efetuar a simulação do cenário e algoritmo desenvolvido.

2.3.1 *Simulation of Urban MObility* (SUMO)

Como vagamente apresentado no subseção 2.2.4, o SUMO é um simulador de trafego urbano de código aberto e sem custo, capaz de simular veículos, pedestres e transporte público em escala microscópica, isto é, controlando individualmente cada componente. Além disso, ele possui ferramentas para auxiliar o cálculo de rotas, visualização da simulação, importação e exportação de dados.

Seu desenvolvimento começou em 2000, com o intuito de facilitar as pesquisas na área, permitindo que os usuários implementassem e avaliassem seus próprios algoritmos. Disponível para o público desde 2001, essa facilidade de interação através de APIs fez com que o SUMO se tornasse o programa mais utilizado entre estudantes e pesquisadores de todo o mundo.

Para utilizá-lo é preciso baixar o *software* disponível no site[4], efetuar a instalação, e ao executar uma simulação é possível visualizar em tempo real o que acontece a cada passo. Há também uma barra de ferramentas, que possibilita interagir e intervir, caso haja necessidade.

No site também estão disponíveis a documentação do *software*, com manuais que explicam o seu funcionamento, tutoriais de suas principais funcionalidades, e exemplos que podem ser executados para melhor visualização de sua capacidade. Além disso, há links para uma página onde estão registrados os erros encontrados no programa e para uma página Wiki sobre ele, onde usuários cadastrados e aprovados pela equipe do SUMO podem contribuir com explicações e códigos para resolver esses erros, criar alguma funcionalidade nova ou incrementar uma já existente.

2.3.2 *Multi-Agent Transport Simulation* (MATSim)

O MATSim[5] é um framework de código aberto que realiza simulações de trânsito em larga escala baseadas em multiagentes, tratando cada agente como um componente individual capaz de se conectar com os demais.

Consiste em um conjunto de módulos para diversas áreas de estudo, podendo esses serem implementados individualmente, em conjunto ou mesmo serem substituídos por um módulo de autoria própria.

O projeto possui o incentivo de alguns institutos, como o *Institute for Transport Planning and Systems* (IVT) e possui grupos de pesquisa para discutir o desenvolvimento de novos módulos ou a melhoria de um já existente.

Em seu site, há documentações, tutoriais, exemplos prontos e exemplos de pessoas ao redor de todo o mundo que utilizaram a ferramenta, permitindo que o usuário saiba como foi o desenvolvimento de cada um desses projetos. Além disso, também há uma lista de extensões que podem ser usadas para diversas finalidades, como cálculo de rota, análise e visualização da simulação.

O fato do próprio módulo de visualização precisar ser instalado além da instalação principal, deixa o MATSim em desvantagem ao SUMO, pois gera maior dificuldade para o usuário neste primeiro momento de uso da ferramenta.

Devido ao SUMO obter maior reconhecimento mundial e utilização já avançada no desenvolvimento de diversos projetos de trânsito, além da maior facilidade deste em implementar algoritmos novos e visualizar a simulação que está sendo executada, esta foi a ferramenta escolhida para implementação desse projeto.

[4] <http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/>
[5] <http://matsim.org/>

2.4 Algoritmos utilizados na simulação

Essa seção citará alguns algoritmos usados para auxiliar ou incrementar a execução da simulação. Todos eles foram aprovados pela equipe da ferramenta de simulação a ser utilizada (SUMO) e podem ser encontrados, juntamente com sua documentação, na página Wiki da ferramenta[6].

2.4.1 Mapeamento das vias

NETCONVERT é um algoritmo com a função de efetuar a conversão dos dados do mapeamento retirados de diversas fontes, como o *OpenStreetMap*, para um formato que pode ser importado em várias ferramentas de simulação, como o SUMO e o MATSim.

É possível gerar dados abstratos, que não correspondam à uma rede real, utilizando o NETGENERATE. Em sua utilização, é possível informar as características da rede a ser gerada, como o comprimento das ruas, a quantidade total de ruas ou restringir quantidade vertical e horizontal. Ambos foram desenvolvidos em C++ para os sistemas operacionais GNU/Linux e Windows.

O algoritmo utilizado neste projeto foi o NETCONVERT, pois foram usados dados de mapeamento de um cenário real, não necessitando do algoritmo de geração.

2.4.2 Cálculo de rota

Os algoritmos de criação de rotas utilizados foram o RandomTrips e o DUAROUTER.

O primeiro, como o próprio nome sugere, gera rotas aleatórias para veículos por toda a simulação, a partir de pontos de saída e destino também escolhidos aleatoriamente ou modicados pelo usuário.

O segundo tem a finalidade de calcular a rota que cada carro presente na simulação deve fazer para que gaste o menor tempo possível de deslocamento até seu destino. Para isso, precisa receber a rede de vias, gerada pelo NETCONVERT, e um modelo de demanda. Esse modelo é um conjunto de informações sobre os veículos, que serão usados para geração das rotas.

Este projeto utilizou o RandomTrips para gerar as rotas aleatórias dos veículos comuns e o DUAROUTER para gerar as rotas dos veículos de emergência, pois estes precisam de informações mais precisas sobre seu comportamento para garantir a melhor validação do projeto.

[6] <http://sumo.dlr.de/wiki>

2.4.3 Visualização do cenário

O algoritmo POLYCONVERT tem a finalidade de converter formas geométricas (polígonos) em uma representação gráfica do objeto, que poderá ser visualizado na simulação. Foi desenvolvido em linguagem C++ para os SOs GNU/Linux e Windows.

Para sua utilização deve-se importar arquivos que contenham os polígonos a serem representados, alguns formatos possíveis são *osm* (gerados pelo *OpenStreetMap*), *shp* (*shapefile*) e *xml*. É possível determinar a configuração da projeção, como zona UTM, projeção inversa e modelo de coordenadas, e filtrar os polígonos convertidos, por nome ou tipo de polígono.

Este algoritmo foi utilizado para a visualização do cenário simulado de forma mais realista e detalhada.

2.4.4 TraCI (Interface de Controle de Tráfego)

A interface padrão do SUMO para controle da simulação executada é o TraCI (*Traffic Control Interface*). Segundo Yang, Hao e Cai (2016), ele permite adaptar os comportamentos dos veículos e afetar seus movimentos dinamicamente durante a execução da simulação.

Bajpai e Mathew (2011) explica melhor o funcionamento dessa interface. Segundo ele, o SUMO atua como um servidor que é iniciado através da linha de comando. O TraCI usa uma arquitetura cliente-servidor baseada em TCP (Protocolo de Controle de Transmissão) para criar uma conexão com o SUMO. A aplicação cliente envia comandos ao SUMO para controlar a simulação ou pedir informações sobre ela. O SUMO, então, retorna as informações ao TraCI, que atuará como a parte visual da simulação.

Esse componente é muito aplicado em projetos que utilizam VANETs, pois atua na comunicação desses dispositivos, garantindo que a rede de informações não se quebre.

Sua utilização foi fundamental para o desenvolvimento do projeto, pois foi este componente que realizou a comunicação entre o código desenvolvido e a simulação realizada, possibilitando a captação das informações sobre o cenário e o controle dos semáforos em tempo real.

2.4.5 Veins

Para entender melhor o funcionamento e o potencial do TraCI, pode-se observar seu comportamento de comunicação entre os simuladores SUMO e Veins.

O Veins[7] é um framework de código aberto composto de vários módulos que auxiliam a simulação veicular, podendo ser utilizados na configuração, execução e monitoramento da simulação.

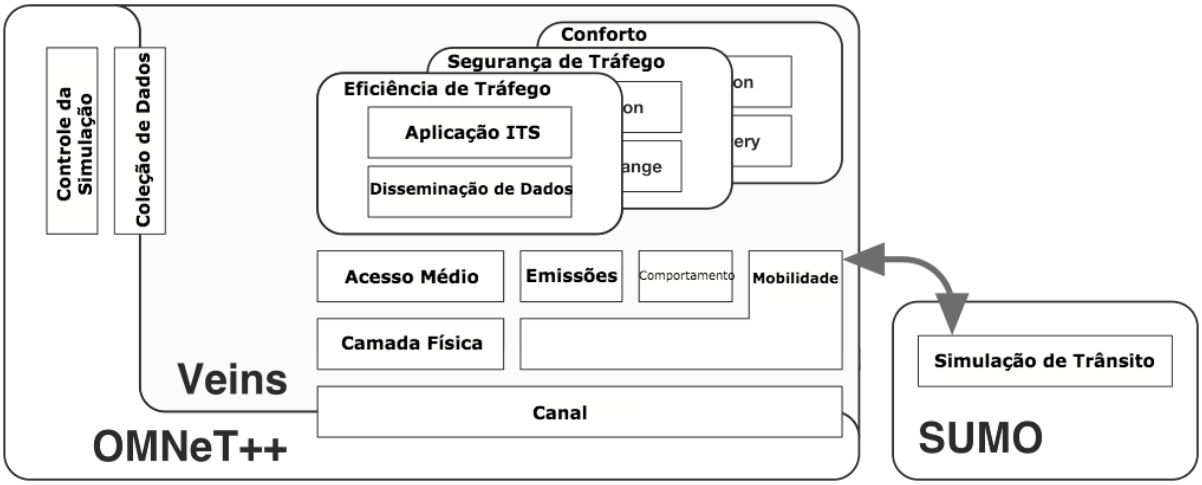

Figura 2.1 – Arquitetura do framework Veins, adaptado e traduzido de (SOMMER, 2016)

Como apresentado na Figura 2.1, a arquitetura da simulação que utiliza o Veins como base. Nela, o SUMO é executado paralelamente ao OMNet++ (*Objective Modular Network testbed in C++*), um simulador de rede, que interpreta os movimentos dos veículos na via como o movimento de um nodo. Utilizando o TraCI, se estabelece um protocolo que permite a troca de informações entre esses dois simuladores.

O Veins utiliza o OMNet++ como base para a execução de seus módulos. Os dados dos veículos e das vias da simulação executada no SUMO são transferidas para o Veins através do TraCI, que retorna o resultado do processamento desses dados, realizado de acordo com a finalidade do módulo usado. Assim, é possível ampliar ainda mais as aplicações da simulação urbana.

[7] <http://veins.car2x.org/>

Capítulo 3

TRABALHOS CORRELATOS

Este capítulo aborda projetos de outros autores, relacionados com o assunto do presente projeto, embasando a importância desse e levantando pontos de discussões sobre seu desenvolvimento.

3.1 Diminuição do fluxo de veículos baseado na distância faltante

Ahmad et al. (2016) propõe o cálculo do tempo para cada semáforo a partir da distância faltante para que o veículo complete seu percurso. Essa abordagem resulta na redução dos carros presentes no trânsito, tendo a intenção final de diminuir congestionamentos.

Utiliza-se, para isso, RFIDs (*Radio-Frequency IDentifications*) que comunicam cada veículo presente na via à dispositivos posicionados ao lado dela, conforme mostra a Figura 3.1.

Cada leitor RFID se comunica em uma frequência diferente, para evitar que haja interferência entre eles, e cada carro deverá ser equipado com um RFID *tag*, para se comunicar com o leitor. Porém essas *tags* estão configuradas em uma frequência padrão, pois se comunicam com leitores diferentes em cada via que passa. É na função de direcionar cada *tag* para o devido leitor que atuarão os módulos de sincronização.

Os módulos de sincronização enviam um sinal à cada veículo em seu alcance, informando a frequência do leitor responsável pela respectiva via. Então o RFID instalado em cada automóvel receberá essa informação e mudará sua frequência para a mesma do leitor, podendo, assim, passar suas informações sobre o veículo, que serão capturadas pelo leitor e enviadas para processamento.

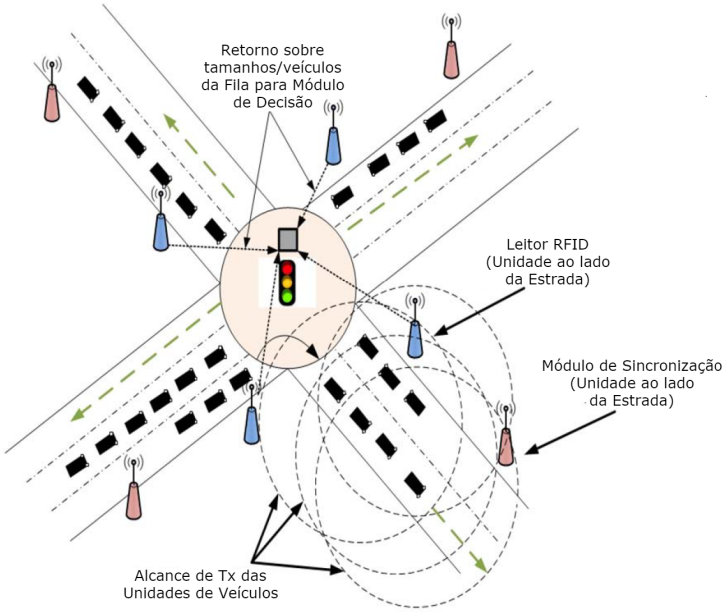

Figura 3.1 – Arquitetura genérica proposta, traduzido de (AHMAD et al., 2016).

Esse processamento será, justamente, o cálculo da distância total faltante para que os veículos da via cheguem à seus destinos. E, então, a via com a menor distância faltante terá seu semáforo aberto e seus carros liberados para que completem seu percurso o mais cedo possível e liberem as vias para passagem de outros carros, evitando congestionamentos.

Porém, para que este projeto atinja um ótimo desempenho, é preciso que todos os veículos possuam um RFID para se conectar com os dispositivos das vias e, esses, devem ser instalados em uma posição que permita um longo alcance de seu sinal e estarem em bom estado de conservação.

3.2 Livre passagem dos veículos de emergência pelos semáforos

A fim de reduzir o tempo de resposta dos veículos de emergência até o local onde ocorreu o incidente, Chowdhury (2016) propõe a criação de uma onda verde para essas viaturas, ou seja, todos os semáforos presentes no caminho do veículo passam para o estado aberto (verde), possibilitando a livre transição desses pela via em todo seu percurso.

Ao receber a ocorrência de um acidente/incidente, a central de emergência recolherá todas as informações e as enviará ao sistema de controle dos semáforos as informações sobre a emergência detectada, como mostra a Figura 3.2.

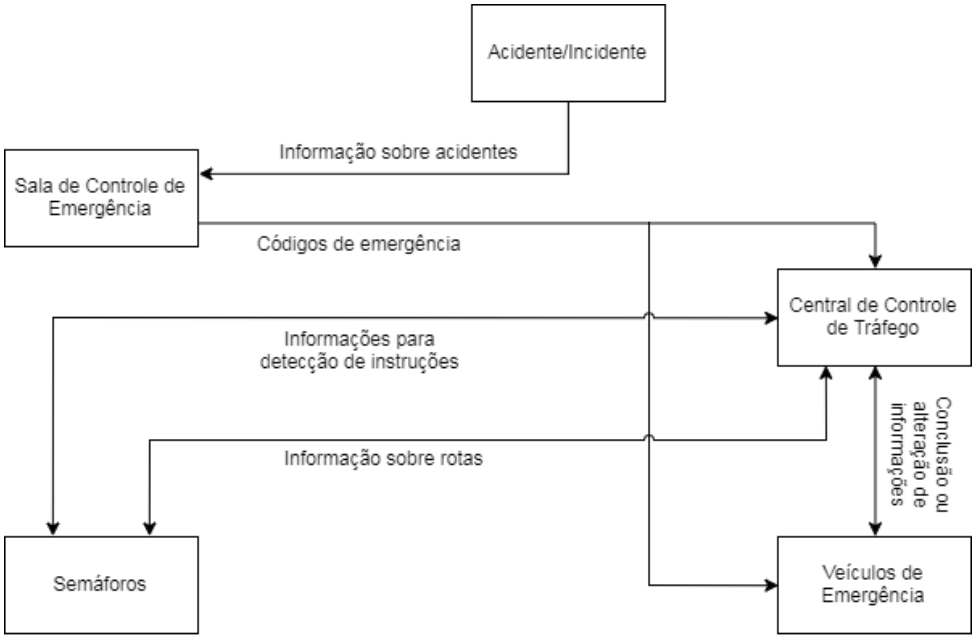

Figura 3.2 – Fluxo das informações, adaptado e traduzido de (CHOWDHURY, 2016).

Assim que o veículo de emergência sair para o atendimento, o controle dos semáforos mudará todos os semáforos presentes em seu percurso para o sinal verde e bloqueará os carros que queiram entrar nessa via, tanto vindos da esquerda, quanto da direita. Desse modo, os carros que estiverem nos semáforos à frente da viatura poderão trafegar livremente e liberarão as vias conforme virarem em outra direção e nenhum outro veículo entrará no percurso, evitando que a viatura precise frear e permitindo que ela chegue à seu destino no menor tempo possível.

Em contrapartida à esse ponto, o bloqueio dos semáforos fora de seu percurso pode gerar grande congestionamento nas vias paralelas. E, além disso, se o percurso for muito longo, os carros bloqueados terão de esperar por muito tempo até que a viatura passe pela via e libere o sinal. Isso pode gerar muita impaciência desses condutores, fazendo com que eles passem pelo sinal vermelho, desacreditados da necessidade de sua espera ser tão longa.

3.3 Alteração dos tempos dos semáforos baseados em intervalos de valores

Para calcular a quantidade de veículos numa via, não se pode levar em conta apenas o espaço total ocupado por eles, deve-se considerar o espaço deixado entre um veículo e outro. É preciso que haja uma distância mínima de segurança exigida, mas não muito além disso. Por isso, Jain e Sethi (2012) utiliza sensores para captar essas duas distâncias para definir o tempo de abertura dos semáforos necessário para

que todos os automóveis consigam passar por ele.

Os valores oferecidos pelos sensores são classificados entre 4 (quatro) categorias, apresentadas nas primeira e segunda colunas da Tabela 3.1. Cada relação entre essas categorias determina uma das categorias de duração, apresentadas na última coluna, para o sinal verde do semáforo.

Tabela 3.1 – Classificação das variáveis, traduzido de (JAIN; SETHI, 2012).

Distância	Espaço	Tempo do semáforo
Muito perto	Muito pequeno	Muito pequeno
Perto	Pequeno	Pequeno
Longe	Médio	Médio
Muito Longe	Grande	Grande

Assim, apesar do tempo estipulado para o semáforo mudar de acordo com as características do trânsito, ele estará sempre entre um intervalo pré-determinado, podendo haver uma margem de erro entre o tempo determinado e o real necessário para o fluxo. Por outro lado, a classificação desse tempo ocorre de forma muito mais rápida que um cálculo totalmente dinâmico, diminuindo o atraso de resposta do semáforo.

3.4 Sinais de alerta/bloqueio para evitar congestionamentos

Este projeto foi desenvolvido por Qi, Zhou e Luan (2016) para uma rede retangular de vias de mão dupla a fim de evitar a criação de congestionamentos nessas vias em caso de uma ocorrência de acidente entre elas.

O semáforo utilizado para implementação do sistema proposto é ilustrado na Figura 3.3. Nele, há uma luz vermelha, que indica quando o veículo não pode avançar, uma luz amarela, que orienta o motorista à frear, pois a luz vermelha será acionada, e três luzes verdes, que permitem o motorista virar à esquerda, seguir em frente ou virar à direita, respectivamente.

Além disso, há seis luzes que indicam a proximidade do condutor à uma via onde ocorreu um acidente. Três delas, (d), (e) e (f), manifestam o sinal de alerta para uma possível formação de congestionamento caso o motorista vire à esquerda, siga em frente ou vire à direita, respectivamente. As outras três, (a), (b) e (c), bloqueiam o avanço do motorista nessas mesmas direções, em mesma ordem, pois é certa a formação de um congestionamento nessa área.

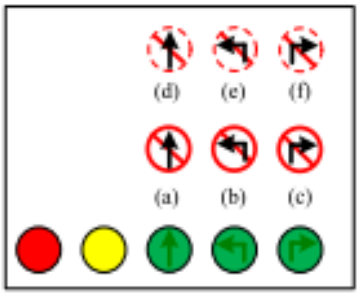

Figura 3.3 – Semáforo utilizado para sinalização das vias (QI; ZHOU; LUAN, 2016).

Ao ocorrer um acidente, o algoritmo proposto calcula as vias próximas onde há possibilidade de se formar um congestionamento. Então, os sinais de bloqueio são acionados nas vias mais próximas para as direções que levam os motoristas ao local do incidente. Já os sinais de alerta são acionados nas direções em que o congestionamento será menor, mas ainda possível. E, por último, as vias que ficarão livres do congestionamento seguirão com a sinalização normal.

3.5 Rede de sensores *wireless* para controle de semáforos

Para controlar o tempo dos semáforos a fim de reduzir o congestionamento formado nas vias, Zhou (2013) utiliza uma rede de sensores *wireless* (WSN - *Wireless Sensor Networks*) para captar as informações do trânsito em tempo real e as transmite instantaneamente através de uma comunicação *wireless*.

Os sensores são distribuídos nas vias de acordo com a Figura 3.4 e informam, basicamente, o número de veículos que chegam e saem das vias. Com esses dados, é possível determinar ainda a densidade de veículos de cada via.

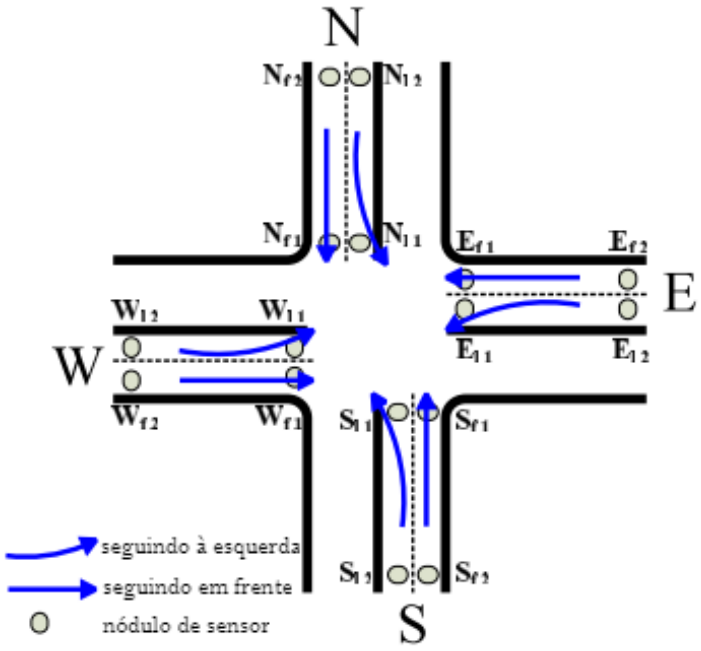

Figura 3.4 – Distribuição dos sensores, traduzido de (ZHOU, 2013).

Analisando outros fatores como o tempo médio de espera, o peso do volume do trafego e circunstâncias especiais, é definida a ordem de prioridade das vias, ou seja, as vias que serão liberadas primeiro e as que precisarão esperar mais. E então, é calculado o tempo da abertura de cada uma delas, que deve durar o suficiente para que todos os veículos presentes consigam passar pelo semáforo.

3.6 Conclusão

Para um melhor entendimento dos trabalhos apresentados e melhor comparação entre suas principais características, a Tabela 3.2 apresenta um sumário de características de cada projeto.

Tabela 3.2 – Comparação entre os trabalhos citados. Os trabalhos 1, 2, 3, 4 e 5 se referem, respectivamente aos projetos de Ahmad et al. (2016), Chowdhury (2016), Jain e Sethi (2012), Qi, Zhou e Luan (2016) e Zhou (2013).

Trabalhos	Trab. 1	Trab. 2	Trab. 3	Trab. 4	Trab. 5
Captação de informações	RFIDs	Manual	Sensores	N/I	WSN
Controle de semáforos (sem emergência)	Sim	Não	Sim	Sim	Sim
Controle de semáforos (com emergência)	Não	Sim	Não	Não	Não
Controle de congestionamento	Não	Não	Não	Sim	Sim

Para comparação desses trabalhos, foram analisadas as características:

1. Captação de informações, que diz respeito ao método utilizado para coletar as informações sobre os veículos e/ou o trânsito;
2. Controle de semáforos (sem emergência), que indica se o projeto proposto faz o controle dos semáforos para o trafego comum, sem a existência de viaturas;
3. Controle de semáforos (com emergência), que indica se o projeto faz o controle dos semáforos priorizando os veículos de emergência;
4. Controle de congestionamento, que revela se o projeto realizou o controle das vias ao redor do semáforo, evitando que a abertura desse resultasse em engarrafamento.

Analisando a Tabela 3.2, é possível perceber que os trabalhos 1 e 3 não abordam nem a questão da prioridade dos veículos de emergência, nem a criação de congestionamentos nas vias próximas ao semáforo. O trabalho 2 chama a atenção por ser o único citado que realiza o controle de emergências, porém este é seu único foco. Já os trabalhos 4 e 5 realizam o controle dos semáforos para evitar a ocorrência de congestionamentos na área controlada, sem o controle de viaturas.

Também é possível verificar que cada trabalho possui uma forma de captação diferente dos dados sobre o cenário analisado. Isso mostra como não há um método ideal para essa área e, por isso, não há a definição de um único a ser usado para o presente projeto.

O projeto desenvolvido aborda todos os itens analisados, ou seja, controla os semáforos para o trânsito comum, mas respeitando a prioridade dos veículos de emergência, caso exista, e, ainda, se preocupa que o fluxo de veículos gerado não cause congestionamentos na área.

Capítulo 4

CENÁRIO DE SIMULAÇÃO

Neste capítulo serão descritos todos os passos realizados para a preparação do cenário usado no desenvolvimento do projeto e em sua validação. O cenário é a base do projeto, pois ele dita os limites a serem observados em seu planejamento e considerados em sua validação.

4.1 Definição do cenário

O primeiro passo para o desenvolvimento do controle dos semáforos foi a definição do cenário, para conhecer os problemas que ele traria e direcionar o desenvolvimento, principalmente, para a solução destes problemas.

Foi definido então um cenário que possui cruzamentos simples entre duas ruas, reduzindo os diferentes tipos de cruzamentos a serem considerados. As ruas em questão são todas de mão única, simplificando os dados a serem analisados. Os cruzamentos possuem sentido apenas para seguir em frente, sem conversões à direita ou à esquerda, ou seja, os veículos presentes nas ruas verticais só poderão seguir na direção vertical e os veículos presentes nas ruas horizontais poderão seguir apenas na direção horizontal. Assim, haverá menos cruzamentos entre as vias existentes e, então, o controle dos semáforos possuirá menos fases, agilizando a análise dos dados e, consequentemente, a definição da fase a ser acionada.

O ambiente utilizado foi a cidade de Marília, localizada no interior do estado brasileiro de São Paulo, por maior conhecimento sobre o comportamento do trânsito nesta região, possibilitando uma análise mais realista. Sendo assim, o cenário definido foi o centro da cidade, contendo os cruzamentos das ruas Paes Leme, Prudente de Moraes e Nove de Julho com as ruas Quinze de Novembro, São Luiz e Quatro de Abril, como ilustrado na Figura 4.1.

Figura 4.1 – Cenário do projeto, no centro de Marília-SP, constituído por 9 cruzamentos e 5 semáforos.

4.2 Preparação do cenário

Para ser possível a utilização desse cenário pela ferramenta de simulação do projeto foi preciso adicionar as informações dos semáforos existentes na área. Para isso foi utilizada a ferramenta OpenStreetMap (OSM), apresentada na subseção 2.2.5.

Ao possuir uma conta no site[1], é possível editar as informações contidas no mapa, como ruas, edifícios, pontos turísticos, ferrovias e muito mais. As informações adicionadas para este projeto foram os semáforos existentes nos cruzamentos do cenário, pois não havia nenhum cadastrado nessa área.

Para realizar a adição dos semáforos é preciso seguir os passos:

1. Acessar o site[1] da ferramenta;
2. Criar uma conta do OSM ou se identificar através da conta de terceiros e efetuar o login. As disponíveis, segundo o site, são OpenID, Google, Facebook, Windows Live, GitHub e Wikipedia;
3. Centralizar o mapa na área sobre a qual deseja adicionar as informações e clicar em "Editar", no canto superior esquerdo;
4. Selecionar o ponto referente ao cruzamento que receberá o semáforo. Em cruzamentos de 2 ou mais vias, selecionar cada ponto;

[1] <www.openstreetmap.org>

5. À esquerda, nas opções do elemento, adicionar uma etiqueta do tipo "*highway*"e valor "*traffic_signals*";

4.3 Simulação do cenário

Nos próximos passos será utilizada a expressão LOCAL_DO_ARQUIVO para se referir ao local onde estarão salvos os arquivos do projeto, podendo ser o local de sua preferência.

A retirada das informações sobre o cenário, tanto as ruas existentes e seus cruzamentos, quanto os semáforos adicionados, é feita por meio do próprio site do OpenStreetMap. Ao realizar a exportação de uma determinada área nesta ferramenta, todas as informações sobre todos os elementos existentes nessa região são exportados juntamente com as já citadas.

Para realizar a exportação das informações sobre o cenário definido é necessário, estando já logado no site da ferramenta (referente aos passos 1 e 2 da seção 4.2):

1. Centralizar o mapa na área sobre a qual deseja adicionar as informações e clicar em "Exportar", no canto superior esquerdo;
2. Selecionar a opção "Selecionar outra área manualmente", caso não queira exportar todo o mapa exibido;
3. À esquerda do mapa, clicar no botão "Exportar". O *download* deve ser iniciado em seu navegador;
4. Salvar o arquivo exportado com o nome LOCAL_DO_PROJETO/map.osm;

Em seguida, é preciso abrir um console/terminal na pasta LOCAL_DO_SUMO/bin para executar as linhas de comando, sendo LOCAL_DO_SUMO a pasta da ferramenta SUMO, baixada para sua instalação. Então, é feita a conversão o arquivo exportado para o formato exigido pelo SUMO, executando o código:

```
1 netconvert --osm-files LOCAL_DO_PROJETO/map.osm -o LOCAL_DO_PROJETO/map.net
    .xml
```

É preciso obter também os desenhos dos polígonos presentes no cenário, para que a visualização da simulação seja mais real e fácil de ser entendida. Para isto, pode-se realizar o *download* das informações pelo próprio site da ferramenta[2] e salvá-las no arquivo LOCAL_DO_PROJETO/typemap.xml.

[2] <http://sumo.dlr.de/wiki/Networks/Import/OpenStreetMap#Importing_additional_Polygons_.28Buildings.2C_Water.2C_etc..29>

Para converter este arquivo para o formato aceito pelo SUMO, é executado, ainda na mesma pasta, o código:

```
polyconvert --net-file LOCAL_DO_PROJETO/map.net.xml --osm-files
    LOCAL_DO_PROJETO/map.osm --type-file LOCAL_DO_PROJETO/typemap.xml -o
    LOCAL_DO_PROJETO/map.poly.xml
```

A configuração da simulação é salva no arquivo LOCAL_DO_PROJETO/map.sumo.cfg, com o conteúdo:

```
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="http://sumo.sf.net/xsd/sumoConfiguration.xsd
    ">
  <input>
    <net-file value="LOCAL_DO_PROJETO/map.net.xml"/>
    <additional-files value="LOCAL_DO_PROJETO/map.poly.xml"/>
  </input>

  <time>
    <begin value="0"/>
    <end value="10000"/>
    <step-length value="0.1"/>
  </time>
</configuration>
```

Finalmente, para executar a simulação do cenário é executado o comando:

```
sumo-gui -c LOCAL_DO_PROJETO/map.sumo.cfg -S
```

Uma nova janela do SUMO deve abrir e a simulação começará a ser executada automaticamente.

Capítulo 5

CONTROLE DOS SEMÁFOROS

Esse capítulo contém todos os detalhes sobre o ponto central desse projeto: o desenvolvimento do algoritmo que realiza o controle dos semáforos, definindo suas fases de acordo com o trânsito em que se insere. São descritos nele, o algoritmo utilizado para aprendizado do projeto sobre o cenário e a aplicação deste utilizando dados da simulação em tempo real.

5.1 K-Nearest Neighbors (KNN)

O algoritmo utilizado para o controle dos semáforos é o chamado KNN (K-Nearest Neighbors), destacado por (SHARMA, 2017) como entre os dez algoritmos mais utilizados para reconhecimento de padrões, classificação de texto, etc.

O KNN é principalmente utilizado para classificações, porém também pode ser utilizado para estimações e previsões, como aponta (LAROSE, 2005). O autor cita a simplicidade da classificação de um novo dado, já que a base de dados de treinamento já está armazenada no algoritmo.

Ele determina a classificação de um dado de acordo com os seus *k* vizinhos mais próximos, sendo *k* um número inteiro definido na implantação do algoritmo.

Para efetuar o treinamento do algoritmo é necessário, primeiramente, determinar as informações relevantes para a definição de seus padrões. Informações além do necessário podem distorcer o entendimento desse padrão e gerar classificações incorretas.

Então, é definido a porcentagem desses dados de teste, geralmente a maioria, para ser utilizada na criação do padrão entre eles e o restante é utilizado para validação desse padrão, tornando possível calcular a porcentagem de acerto na classificação dos novos dados. É definido também o valor de *k* para a definição da classe do novo dado, sendo este o número de classes mais próximas, entre os dados de

treinamento, utilizados para essa definição.

Ao iniciar a fase de aplicação, ou seja, a classificação do novo dado, o algoritmo calcula a distância entre este e cada um dos dados do treinamento. Essa distância pode ser medida de diversas formas, sendo a distância euclidiana, definida pela seguinte equação, a utilizada neste projeto.

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Para este método de cálculo da distância, primeiro é calculada a diferença entre cada variável do novo dado e a variável correspondente no dado do treinamento avaliado. Em seguida, eleva-se cada diferença encontrada ao quadrado, soma-se todos estes valores e, por fim, retira-se a raiz quadrada. O resultado será a distância euclidiana entre o dado a ser classificado e o dado de treinamento utilizado. Esta operação deve ser efetuada para cada dado de treinamento.

Em seguida, o conjunto de todas as distâncias euclidianas é ordenada em ordem crescente e seleciona-se apenas os *k* primeiros elementos ordenados para a classificação. É, então, contada a quantidade de cada tipo de classe presente nestes elementos e a classe com mais aparições será a determinada para o novo dado.

O valor definido para *k* deve sempre ser ímpar, pois caso contrário, poderá haver empate na quantidade de classes vizinhas existentes e, assim, a classificação do novo dado poderá se tornar incorreta.

5.2 Criação do treinamento

Para efetuar o treinamento do algoritmo é necessário determinar as informações relevantes para a definição de seus padrões. Para o projeto desenvolvido, os dados observados foram:

a Quantidade de ambulâncias presentes em cada quadra;

b Quantidade de carros, incluindo ambulâncias, presentes em cada quadra;

c Maior tempo de espera, em segundos, entre os carros presentes em cada quadra.

Para criação dos dados de treinamento, foram admitidos, para os dados utilizados, os intervalos de valores:

- De 0 a 2 ambulâncias presentes por quadra, sendo 90%, 5% e 5% a probabilidade de ocorrência de 0, 1 e 2 ambulâncias, respectivamente;

- De 0 a 12 carros, incluindo ambulâncias, presentes por quadra, sendo este limite máximo a quantidade de veículos que preenchem uma quadra na simulação;
- De 5 a 30 segundos de espera para abertura do semáforo, sendo este limite mínimo o necessário para que um veículo atravesse o cruzamento.

Distribuindo as informações apresentadas no cenário determinado, a disposição das variáveis a serem utilizadas na etapa de treinamento do algoritmo é apresentada na Figura 5.1.

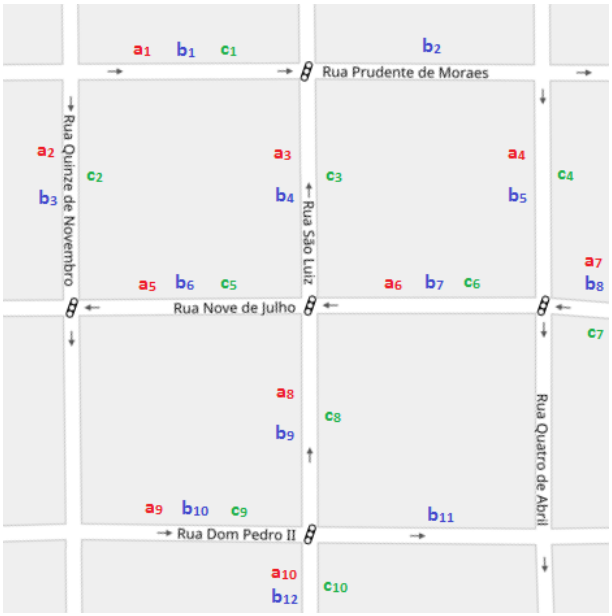

Figura 5.1 – Variáveis para treinamento. (a=quantidade de ambulancias. b=quantidade de carros. c=tempo de espera)

Analisando as diferentes situações capazes de ocorrer no cenário, foram definidos os testes para que possam gerar um padrão de dados bem definido. Neles estão contidos os valores de cada variável em cada possível situação e a fase do semáforo que melhor gerencia os veículos nesse momento.

Os valores definidos para as classes que determinam a fase do semáforo a ser ativada são:

- 0 para liberar os carros que seguem na vertical (direção Norte-Sul);
- 1 para liberar os carros que seguem na horizontal (direção Oeste-Leste).

De maneira geral, quanto mais situações forem descritas no treinamento, mais preciso ficará o resultado exibido pelo algoritmo e menor a probabilidade de ocorrer *overfitting*, isto é, a memorização da classificação dos dados, sem gerar um padrão

para ser seguido. Assim como a utilização de variáveis dispensáveis na definição do padrão pode aumentar esse efeito.

Porém, como a classificação do novo dado é definida pela classe mais ocorrente entre os *k* vizinhos mais próximos, o desequilíbrio da quantidade de testes existentes para cada classe pode gerar uma classificação injusta. Sendo assim, conclui-se que a criação dos testes de treinamento deve ser feita com muita atenção e cuidado, pois esta é a parte mais sensível do projeto.

Após a criação dos treinamentos, foi realizada a normalização dos dados, dividindo cada variável pelo respectivo valor limite máximo, a fim de deixá-los entre os valores 0 e 1. Assim, as variáveis possuirão pesos diferentes no cálculo da distância entre os dados e, por consequência, garantirá maior eficiência na priorização dos veículos de emergência, já que esta variável não possui a mesma importância dos demais no processo de classificação.

5.2.1 Implementação do algoritmo

A classe de implementação do algoritmo KNN obtém validações dos dados de treinamento, para garantir que dados passados de maneira incorreta não resultem em erros de classificação de um dado e que os erros disparados sejam de fácil e correto entendimento.

Um exemplo dessa validação é a substituição das *Strings* contidas entre os dados de treinamento, que devem conter apenas valores reais, pelo valor -99999. Este valor é utilizado para que este dado seja interpretado como não sendo parte do padrão de classificação e assim, não seja preciso descartar todo o dado de treinamento por não estar no formato esperado.

O controle dos semáforos, ou seja, os comandos responsáveis por alterar suas fases foram desenvolvidos separadamente do algoritmo KNN, para que este possa ser utilizado de maneira independente ao cenário deste projeto. Também para sua independência, a classe do KNN recebe, por parâmetros em sua inicialização, todas as configurações necessárias para seu funcionamento.

A linguagem de programação utilizada na elaboração da primeira classe foi *Python* 2.7, pois é de fácil entendimento e implementação. A segunda classe deve ser desenvolvida para cada caso de utilização da primeira e deve, portando, ser elaborada com uma linguagem compatível com esta, para que seja possível a integração entre elas.

5.2.2 Fase de treinamento

Para iniciar a fase de treinamento do algoritmo desenvolvido é necessário, primeiramente, criar uma instância da classe desenvolvida, por meio do código:

```
TLC = TLControl(FILE, QTD_VARIAVEIS, QTD_CLASSIFICACOES, QTD_CLASSES,
    PORCENTAGEM_TESTE)
```

Neste código, TLC é a variável que armazenada a instância criada. São passados, para a criação desta instância, os parâmetros necessários para configuração do algoritmo KNN, sendo: FILE uma *string* com o nome do arquivo que possui os testes para treinamento; QTD_VARIAVEIS um valor inteiro referente à quantidade de variáveis que serão observadas no cenário; QTD_CLASSIFICACOES um valor inteiro referente à quantidade de classes retornadas para o novo dado; QTD_CLASSES um valor inteiro referente à quantidade de classes possíveis para o dado; PORCENTAGEM_TESTE um valor inteiro referente à porcentagem dos dados de treinamento que serão utilizados para determinar a taxa de acerto do algoritmo, sendo o complemento dessa porcentagem os dados utilizados para definição do padrão.

Para o cenário definido, os valores utilizados foram os seguintes, sendo tcl_cima, tlc_esquerda, tlc_centro, tlc_direita e tlc_baixo os controles dos semáforos localizados, respectivamente, no cruzamento entre as ruas São Luiz e Prudente de Moraes, Quinze de Novembro e Nove de Julho, São Luiz e Nove de Julho, Quatro de Abril e Nove de Julho e Dom Pedro II e São Luiz:

```
tlc_cima = TLControl('testes1.data', 7, 1, 2, 30)
tlc_esquerda = TLControl('testes2.data', 6, 1, 2, 30)
tlc_centro = TLControl('testes3.data', 8, 1, 2, 30)
tlc_direita = TLControl('testes4.data', 7, 1, 2, 30)
tlc_baixo = TLControl('testes5.data', 8, 1, 2, 30)
```

Em seguida, para iniciar a fase de treinamento do algoritmo, que utilizará os parâmetros passados no código anterior, é executado o seguinte comando, sendo TLC a variável que armazena a instância criada:

```
TLC.treino()
```

É possível realizar a exportação e importação dos padrões de classificação gerados, para maior facilidade na implementação do projeto. Para isto, utiliza-se os comandos:

```
TLC.exportar(FILE_PADROES)
TLC.importar(FILE_PADROES)
PADROES = TLC.exportar()
TLC.importar(None, PADROES)
```

O primeiro realiza a exportação dos padrões para o arquivo definido pela variável FILE_PADROES, que deve ser uma *string*. O segundo comando executa a importação desses padrões localizados no arquivo FILE_PADROES, que também deve ser uma *string*. Já o terceiro comando é responsável por retornar a lista de padrões, que serão armazenados na variável PADROES. O último comando realiza a importação da lista de padrões não através de um arquivo, mas sim da variável PADROES informada.

5.2.3 Fase de aplicação

A execução da fase de aplicação também ocorre de forma simples. Para isto, é preciso que todos os dados de treinamento sejam coletados. Neste caso, a captação foi feita através do TraCI, que retorna diversas informações sobre a simulação sendo executada na ferramenta SUMO. Porém, os dados podem ser coletados da maneira mais conveniente para quem realizar a integração o projeto.

Para iniciar esta fase, basta executar o seguinte comando, ainda utilizando a variável que armazena a instância do algoritmo no lugar de TLC:

```
1 CLASSE = TLC.aplicacao(DADOS, K)
```

Neste comando, a variável CLASSE se refere ao resultado da classificação do novo dado, que será uma lista com o tamanho de QTD_CLASSIFICACOES informada na instanciação do algoritmo e conterá valores inteiros dentro do limite de classificações estabelecido pela variável QTD_CLASSES, também informada na criação da instância. DADOS é uma lista contendo os valores do novo dado à ser classificado e K é um inteiro referente ao número de classes vizinhas utilizadas na definição da nova classe.

Para o cenário utilizado, foi definido valor 3 para a variável *k* nos testes de todos os semáforos. Foi escolhido um número pequeno, pois a classificação do novo dado não deve depender de muitos vizinhos, já que o ambiente urbano possui informações que, ao serem levemente alteradas, podem resultar em uma classificação totalmente diferente.

Com o resultado retornado para a variável CLASSE, é definida a classificação do novo dado. Para o cenário do projeto, essa classificação representa a fase do semáforo que deve ser acionada para que garanta o melhor gerenciamento dos veículos presentes nas vias.

Capítulo 6
RESULTADOS

Os resultados obtidos, assim como a elaboração da análise desses dados serão detalhados neste capítulo. Este processo concluirá a validação da melhoria causada pelo projeto produzido no controle dos semáforos e, consequentemente, no gerenciamento dos veículos das vias envolvidas, ressaltando o controle dos veículos de emergência e do congestionamento gerado no cenário.

6.1 Resultados obtidos

Para validar a melhoria do controle das fases dos semáforos foram executadas 10 simulações para cada quantidade de ambulância e controle dos semáforos utilizado. A quantidade de ambulância variou entre os valores 0 e 4 e os controles dos semáforos disponíveis foram os tempos fixos definidos no cenário no momento do desenvolvimento deste projeto e o algoritmo KNN desenvolvido.

A quantidade e as rotas dos veículos presentes nas simulações foram geradas de modo aleatório e se alteram entre as simulações com diferentes números de ambulância, porém se mantêm entre a utilização dos tempos fixos e do KNN para controle dos semáforos.

Em cada simulação, foi computado o tempo consumido por cada veículo para completar seu percurso e, em seguida, calculada a média destes tempos, que estão distribuídas nas seguintes tabelas.

Na Tabela 6.1 estão contidas as médias dos tempos de percurso apenas das ambulâncias nas simulações geradas. Nota-se que o algoritmo KNN obteve médias menores de tempo em todas as simulações, comprovando sua eficiência na priorização dos veículos de emergência ao determinar as fases dos semáforos.

Tabela 6.1 – Médias dos tempos de percurso das ambulâncias (em segundos).

		Nº de ambulâncias			
Simulação	**Controle das fases**	**1**	**2**	**3**	**4**
1	**Tempos Fixos**	73,45	63,97	62,00	52,33
	KNN	25,10	43,50	36,90	36,25
2	**Tempos Fixos**	40,40	76,15	66,00	49,90
	KNN	25,10	44,60	37,13	38,85
3	**Tempos Fixos**	41,20	74,80	53,23	58,93
	KNN	25,20	47,90	37,13	35,65
4	**Tempos Fixos**	38,30	58,15	47,67	54,13
	KNN	25,00	47,85	41,87	40,63
5	**Tempos Fixos**	37,00	74,95	67,60	59,68
	KNN	32,30	53,70	41,37	41,48
6	**Tempos Fixos**	43,20	94,45	67,50	67,83
	KNN	25,10	46,90	46,47	37,78
7	**Tempos Fixos**	64,60	95,60	63,47	58,78
	KNN	25,00	48,75	46,03	38,90
8	**Tempos Fixos**	43,20	74,75	64,33	46,55
	KNN	25,00	39,95	45,17	43,18
9	**Tempos Fixos**	40,20	74,80	58,10	69,00
	KNN	24,80	42,25	37,53	38,20
10	**Tempos Fixos**	56,60	76,20	63,43	73,03
	KNN	42,20	48,30	36,47	41,05

Para comprovar que essa priorização não afetou o controle dos demais veículos, a tabela Tabela 6.2 apresenta as médias dos tempos de percurso de todos os veículos presentes em cada simulação, incluindo as ambulâncias.

Todas as rotas dos veículos foram geradas aleatoriamente pelo algoritmo *RandomTrips*, apresentado no subseção 2.4.2.

Observando estes dados, conclui-se que o algoritmo desenvolvido é eficaz, também, para gerenciar veículos que não sejam de emergência. O tempo de percurso destes diminuiu inclusive nas simulações com veículos que possuam essa prioridade.

Tabela 6.2 – Médias dos tempos de percurso de todos os veículos (em segundos).

		Nº de ambulâncias				
Simulação	**Controle das fases**	**0**	**1**	**2**	**3**	**4**
1	**Tempos Fixos**	36,35	32,31	33,07	34,09	31,46
	KNN	24,12	24,90	24,76	24,82	24,68
2	**Tempos Fixos**	36,81	36,73	41,24	36,80	42,36
	KNN	20,71	20,59	25,41	20,81	35,28
3	**Tempos Fixos**	34,74	31,65	36,08	41,19	30,27
	KNN	23,05	21,95	26,67	25,96	24,50
4	**Tempos Fixos**	25,05	34,63	42,20	28,26	32,99
	KNN	19,49	28,89	25,39	26,22	27,91
5	**Tempos Fixos**	32,07	33,50	35,75	36,03	33,87
	KNN	25,05	24,16	27,13	28,82	28,66
6	**Tempos Fixos**	29,49	27,25	31,22	38,31	37,83
	KNN	25,06	21,22	24,38	27,33	28,93
7	**Tempos Fixos**	33,63	44,21	36,27	36,74	35,97
	KNN	24,60	25,44	25,50	28,26	29,08
8	**Tempos Fixos**	23,29	30,63	37,08	45,71	29,81
	KNN	17,81	21,40	22,70	30,16	26,94
9	**Tempos Fixos**	28,73	26,97	27,63	38,55	42,83
	KNN	23,80	17,67	20,66	27,37	29,89
10	**Tempos Fixos**	32,64	37,03	32,95	32,56	39,28
	KNN	25,31	28,40	25,32	21,51	29,91

6.2 Análise dos resultados

Para fundamentar a analise dos gráficos é preciso levar em conta que a aleatoriedade das rotas dos veículos pode desequilibrar a comparação dos resultados. Assim, a diferença entre as médias obtidas nos diferentes controles varia de acordo com a simulação, mas é notável que o algoritmo desenvolvido obteve melhora em todas as simulações executadas.

A Tabela 6.3 mostra, a partir do desvio padrão referente a cada média, que nas simulações que utilizam o algoritmo KNN, os tempos de percurso se mantiveram mais constantes, pois não obtiveram grande influência dos demais veículos presentes na simulação.

As simulações com apenas uma ambulância não possuem desvio padrão, pois para calcular este valor é necessário obter pelo menos dois valores de média.

Tabela 6.3 – Desvio padrão dos tempos de percurso das ambulâncias.

		Nº de ambulâncias		
Simulação	**Controle das fases**	**2**	**3**	**4**
1	**Tempos Fixos**	49,43	42,99	15,93
	KNN	7,07	10,22	14,85
2	**Tempos Fixos**	53,10	43,62	18,76
	KNN	8,49	10,91	14,36
3	**Tempos Fixos**	51,19	21,82	36,56
	KNN	9,33	10,85	13,70
4	**Tempos Fixos**	27,93	24,55	39,93
	KNN	13,36	11,50	19,04
5	**Tempos Fixos**	51,27	36,66	13,35
	KNN	21,35	17,37	13,98
6	**Tempos Fixos**	77,71	34,49	29,13
	KNN	11,17	19,86	17,75
7	**Tempos Fixos**	80,19	39,16	36,68
	KNN	14,35	19,26	14,18
8	**Tempos Fixos**	51,12	40,47	20,40
	KNN	2,05	23,92	11,18
9	**Tempos Fixos**	51,48	18,22	30,01
	KNN	5,73	11,85	14,22
10	**Tempos Fixos**	53,17	39,17	51,53
	KNN	13,86	10,76	16,81

A Tabela 6.4 comprova que a priorização das ambulância não interferiu, também, na diminuição do desvio padrão dos tempos de percurso de todos os veículos. Esta análise valida a possibilidade de predições dos tempos de percurso das ambulâncias, pois seus valores não possuem grande desvio padrão.

Tabela 6.4 – Desvio padrão dos tempos de percurso de todos os veículos.

		Nº de ambulâncias				
Simulação	**Controle das fases**	**0**	**1**	**2**	**3**	**4**
1	**Tempos Fixos**	21,22	20,10	22,07	22,01	20,46
	KNN	10,82	11,99	10,92	12,52	13,72
2	**Tempos Fixos**	22,04	18,83	26,28	29,61	18,70
	KNN	10,11	7,52	10,62	11,43	13,55
3	**Tempos Fixos**	21,97	20,06	26,36	22,64	22,22
	KNN	13,44	11,99	12,75	13,21	12,98
4	**Tempos Fixos**	15,69	15,51	21,01	19,12	23,42
	KNN	8,65	14,44	11,82	13,94	15,84
5	**Tempos Fixos**	21,32	21,80	23,31	24,14	19,37
	KNN	12,50	13,14	14,63	13,32	13,93
6	**Tempos Fixos**	17,41	18,28	27,28	22,57	24,04
	KNN	14,78	9,36	12,27	14,52	12,49
7	**Tempos Fixos**	20,68	20,27	28,67	21,47	23,73
	KNN	13,86	10,28	15,16	12,76	13,60
8	**Tempos Fixos**	16,78	17,81	26,54	24,82	22,55
	KNN	7,35	10,01	11,80	16,01	13,43
9	**Tempos Fixos**	16,23	16,71	21,51	24,45	27,91
	KNN	12,88	6,99	10,42	13,97	14,23
10	**Tempos Fixos**	18,32	17,82	23,63	24,58	25,11
	KNN	11,69	16,39	13,27	11,15	15,83

Para melhor visualização dos resultados obtidos e da eficiência do algoritmo desenvolvido, foram gerados gráficos com as médias de tempo de percurso apresentadas. Esses gráficos estão divididos por quantidade de ambulâncias presentes nas simulações, para auxiliar a análise da influência desses veículos no controle dos semáforos.

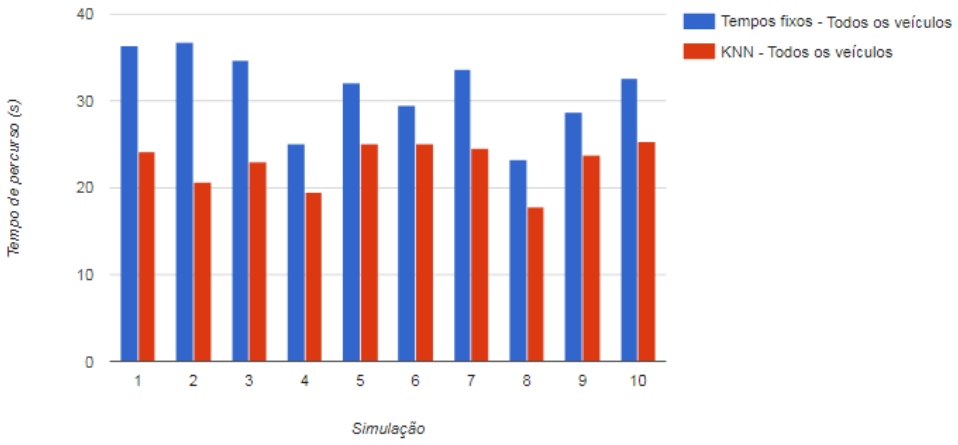

Figura 6.1 – Média dos tempos de percurso com 0 ambulâncias.

Na Figura 6.1 nota-se que o algoritmo gerenciou o trânsito sem ambulâncias de forma mais eficiente, controlando o congestionamento na região.

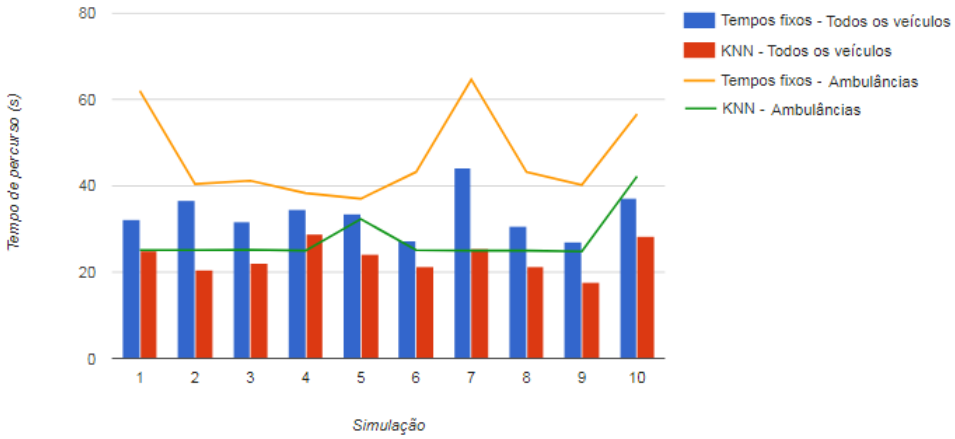

Figura 6.2 – Média dos tempos de percurso com 1 ambulância.

A Figura 6.2 mostra pontos muito elevados nas médias com tempos fixos que foram minimizados com a utilização do algoritmo KNN.

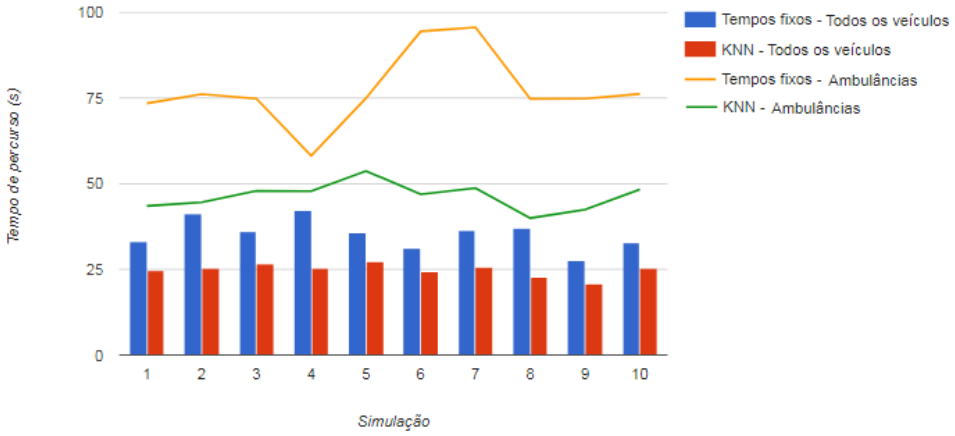

Figura 6.3 – Média dos tempos de percurso com 2 ambulâncias.

Na Figura 6.3 é fortemente notável a eficiência do algoritmo KNN, onde a presença de duas ambulâncias priorizou ainda mais sua passagem pelo cruzamento. Os tempos fixos geraram aumento nos tempos de percurso de ambas as ambulâncias, aumentando ainda mais a diferença entre as médias.

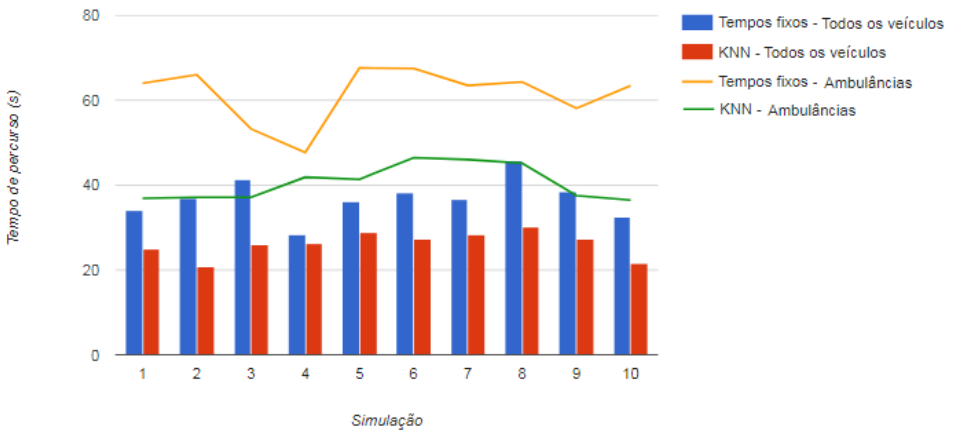

Figura 6.4 – Média dos tempos de percurso com 3 ambulâncias.

Na Figura 6.4 a grande diferença entre as médias se manteve, já que a presença das três ambulâncias obteve total prioridade nos cruzamentos apenas na utilização do algoritmo KNN.

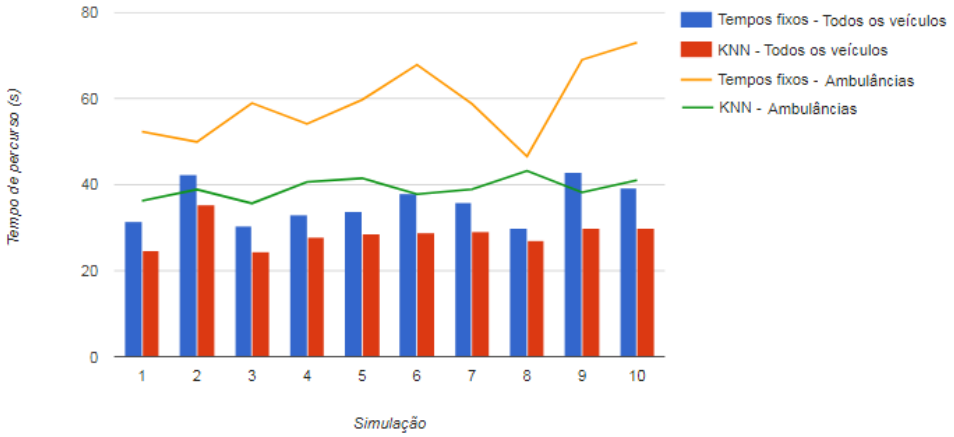

Figura 6.5 – Média dos tempos de percurso com 4 ambulâncias.

A Figura 6.5 ilustra a influência dos demais veículos nos tempos de percurso das ambulâncias com a utilização dos tempos fixos. Isso, pois os pontos de menor diferença entre as médias são referentes às simulações com menos veículos presentes, enquanto que as maiores diferenças estão nas simulações com mais veículos e portanto, as ambulâncias possuíram maior dificuldade em atravessar os cruzamentos.

Capítulo 7
CONCLUSÃO

Neste capítulo será apresentada uma análise geral sobre este projeto e seus resultados. Destacando as impressões obtidas ao longo de seu planejamento, desenvolvimento e sua validação.

7.1 Considerações sobre o projeto

A partir da observação dos resultados das simulações e de suas análises, conclui-se que a implantação do algoritmo KNN obteve grande eficiência na economia de tempo de percurso dos veículos. Essa melhoria foi atingida em todas as simulações efetuadas, com diferentes rotas, quantidades de veículos e ambulâncias.

O sucesso na melhoria dos resultados nas diferentes simulações comprova a dinamicidade do algoritmo desenvolvido, capaz de se ajustar ao cenário implementado e obter sempre o melhor controle das fases dos semáforos.

Para preparação do cenário real definido foi preciso realizar a captação manual dos tempos fixos atuais definidos nos semáforos desta área, pois o Departamento de Trânsito da cidade não obtinha a relação dos semáforos existentes ou os tempos de suas fases. Com o avanço da criação de tecnologias para o cenário urbano, o processo de administração desse ambiente torna-se muito mais simples e prático.

Além disso, a utilização de algoritmos de aprendizado de máquina pode ser muito complexo, abordando dados, definições e métodos, muitas vezes não compreendidos por quem não é especializado no assunto. O algoritmo KNN foi escolhido por seu funcionamento simples, rápido e eficaz. O fato dos testes de treinamento serem criados manualmente por observação de um cenário real gera confiança nos dados gerados, facilitando a utilização deste em diversas aplicações.

7.2 Trabalhos Futuros

Futuramente, espera-se a implementação de outros modelos de aprendizado de máquina neste cenário para tentar obter resultados ainda melhores no controle dos semáforos. Assim, é possível encontrar modelos que lidam melhor com o peso das variáveis e sua ordenação, por exemplo.

A validação dessa nova implementação poderá seguir o mesmo modelo deste projeto, reproduzindo os passos apresentados no Capítulo 4. Assim, será muito mais justa a comparação dos novos resultados com os apresentados no Capítulo 6.

Qualquer novo projeto proposto para integrar, analisar ou complementar o projeto desenvolvido é totalmente incentivado por este. Assim, diversas soluções serão produzidas de forma muito mais rápida e eliminarão os problemas urbanos que são o foco da produção deste projeto.

REFERÊNCIAS

AHMAD, F. et al. Shortest Processing Time Scheduling to Reduce Traffic Congestion in Dense Urban Areas. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, PP, n. 99, p. 1–16, 2016. ISSN 2168-2216.

BAJPAI, A.; MATHEW, T. V. Development of an Interface between Signal Controller and Traffic Simulator. *1st Conference of Transportation Research Group of India (CTRG)*, p. 1–13, 2011.

BOUKERCHE, A. *Algorithms and Protocols for Wireless, Mobile Ad Hoc Networks*. [S.l.: s.n.], 2008. 500 p.

CHOWDHURY, A. Priority based and secured traffic management system for emergency vehicle using IoT. In: *2016 International Conference on Engineering & MIS (ICEMIS)*. IEEE, 2016. p. 1–6. ISBN 978-1-5090-5579-1. Disponível em: <http://ieeexplore.ieee.org/document/7745309/>.

DETRAN, D. N. D. T. *Frota de Veículos*. 2016. Disponível em: <http://www.denatran.gov.br/index.php/estatistica/237-frota-veiculos>.

ERIC, I. P.; SHARI, L. P. BRAESS' PARADOX: SOME NEW INSIGHTS. *-B*, Elsevier Ltd, v. 31, n. 3, p. 265–276, 1997.

JAIN, P.; SETHI, M. Fuzzy Based Real Time Traffic Signal Controller to Optimize Congestion Delays. In: *2012 Second International Conference on Advanced Computing & Communication Technologies*. IEEE, 2012. p. 204–207. ISBN 978-1-4673-0471-9. Disponível em: <http://ieeexplore.ieee.org/document/6168361/>.

LAROSE, D. T. *Discovering Knowledge in Data: An Introduction to Data Mining*. [S.l.: s.n.], 2005. ISBN 0471666572.

MIYADA, A. Proposta de solução para rotas de veículos de carga. 2016.

QI, L.; ZHOU, M.; LUAN, W. An emergency traffic light strategy to prevent traffic congestion. In: *ICNSC 2016 - 13th IEEE International Conference on Networking, Sensing and Control*. [s.n.], 2016. ISBN 9781467399753. Disponível em: <http://ieeexplore.ieee.org/document/7479013/>.

SHARMA, S. Comparative Analysis between Different Clustering Techniques. v. 141401, p. 1–5, 2017.

SOMMER, C. *Veins Documentation*. 2016. Disponível em: <http://veins.car2x.org/documentation/>.

THAPA, R. B.; MURAYAMA, Y. Urban mapping, accuracy, & image classification: A comparison of multiple approaches in Tsukuba City, Japan. *Applied Geography*, Elsevier Ltd, v. 29, n. 1, p. 135–144, 2009. ISSN 01436228. Disponível em: <http://dx.doi.org/10.1016/j.apgeog.2008.08.001>.

YANG, Y.; HAO, S.; CAI, H. Comparison and Evaluation of Routing Protocols Based On A Collaborative Simulation Using SUMO and NS3 with TraCI. 2016.

ZHOU, B. A Real-Time Traffic-Responsive Strategy for Road Congestion Problem. In: *2013 International Conference on Computational and Information Sciences*. IEEE, 2013. p. 1146–1149. ISBN 978-0-7695-5004-6. Disponível em: <http://ieeexplore.ieee.org/document/6643221/>.

Apêndice A

IMPLEMENTAÇÃO DO KNN

```
#!usr/bin/env python
# -*- coding: utf-8 -*-
import math
import sys, os, json

class TLControl:
  def __init__(self, file, qtdV, qtdR, qtdRT, pTeste):
    if(os.path.exists(file)==False or os.path.isfile(file)==False):
      print "ERRO: Arquivo de teste nao existe!"
      sys.exit()

    self.allData = []
    self.dataFile = file
    self.qtdVariaveis = qtdV
    self.qtdResultado = qtdR
    self.qtdResTipos = qtdRT
    self.pctTreinamento = (float)(100-pTeste)/100
    self.qtdTreinamentos = 100

  def read_data(self):
    amostras=[]
    with open(self.dataFile,'r') as f:
      for linha in f.readlines():
        atribs = linha.replace('\n','').split(',');
        if(len(atribs)!=(self.qtdResultado+self.qtdVariaveis)):
          print 'Tamanho incorreto de teste!'
          sys.exit()
        amostra = []
        for atrib in atribs:
          try:
            amostra.append(float(atrib));
          except ValueError, e:
            amostra.append(-99999);
```

```
34          amostras.append(amostra)
35      self.qtdTreinamentos = int(len(amostras)*self.pctTreinamento)
36      return amostras
37
38    def divide_data(self, amostras):
39      train,test = [],[]
40      for i in range(int(self.pctTreinamento*len(amostras))):
41        train.append(amostras[i])
42        self.allData.append(amostras[i])
43
44      for i in range(int(self.pctTreinamento*len(amostras)),len(amostras)):
45        test.append(amostras[i])
46        self.allData.append(amostras[i])
47
48      return train,test
49
50    def dist_euclidiana(self, p, q):
51      soma=0
52      for i in range(len(p)-self.qtdResultado):
53        soma += math.pow((p[i]-q[i]),2)
54      return math.sqrt(soma)
55
56    def knn(self, train, test, k):
57      correct=0
58      for te in test:
59        resps = self.getResps(train, te, k)
60        qtd = 0
61        for res in resps:
62          if te[(qtd-self.qtdResultado)] == res:
63            correct += 1
64          qtd += 1
65
66      return correct, resps
67
68    def getResps(self, train, te, k):
69      dists = {}
70      for i in range(len(train)):
71        dists[i] = self.dist_euclidiana(train[i],te)
72      nn = sorted(dists, key=dists.get)[:k]
73      qtds = {}
74      for i in xrange(self.qtdResultado):
75        qtds[i] = {}
76        for j in xrange(self.qtdResTipos):
77          qtds[i][j] = 0
78      for i in nn:
79        qtd = 0
80        for q in qtds:
```

```
81          for j in xrange(self.qtdResTipos):
82            if train[i][(qtd-self.qtdResultado)] == j:
83              qtds[q][j] += 1
84          qtd += 1
85
86      resps = []
87      for q in qtds:
88        rr = sorted(qtds[q], key=qtds[q].get, reverse=True)
89        resps.append(rr[0])
90      return resps
91
92    def treino(self):
93      print 'Treinando'
94      amostras = self.read_data()
95      train,test = self.divide_data(amostras)
96
97      for i in range(self.qtdTreinamentos):
98        crct, res = self.knn(train,test,i)
99
100   def aplicacao(self, dados, k):
101     if(isinstance(dados, list)==False):
102       print 'Dados em formato invalido!'
103       sys.exit()
104     if(len(dados)!=self.qtdVariaveis):
105       print 'Quantidade invalida de variaveis!'
106       sys.exit()
107     treino = []
108     for dado in dados:
109       try:
110         treino.append(int(dado))
111       except ValueError, e:
112         print '******'
113         print dados
114         sys.exit()
115         treino.append(-99999)
116     resps = self.getResps(self.allData, dados, k)
117     return resps
118
119   def exportar(self, fileName=''):
120     print 'Exportando'
121     if(fileName==''):
122       return self.allData
123
124     file = open(fileName, "w")
125         file.write(json.dumps(self.allData))
126     file.close()
127
```

```
128   def importar(self, fileName, data=''):
129     print 'Importando'
130     if(bool(fileName)==False):
131       self.allData = data
132       return
133 
134     if(os.path.exists(fileName)==False or os.path.isfile(fileName)==False):
135       print "ERRO: Arquivo de importacao nao existe!"
136       sys.exit()
137 
138     file = open(fileName, "r")
139     self.allData = file.readlines()[0]
140     self.allData = json.loads(self.allData)
141     file.close()
```

Apêndice B

SIMULAÇÃO UTILIZANDO KNN

```
#!usr/bin/env python
# -*- coding: utf-8 -*-
import os, subprocess, sys

tools = os.path.join(os.environ['SUMO_HOME'], 'tools')
sys.path.append(tools)
sys.path.append(os.path.join(os.path.dirname(__file__), '..', '..', '..',
    '..', 'tools'))
sys.path.append(os.path.join(os.environ.get("SUMO_HOME", os.path.join(os.
    path.dirname(__file__), '..', '..', '..')), 'tools'))

import traci
from TLControl import TLControl
tlp_ns_v = "Gr"  # fase verde para sentido Norte-Sul
tlp_ns_y = "Yr"  # fase amarela para sentido Norte-Sul
tlp_ol_v = "rG"  # fase verde para sentido Oeste-Leste
tlp_ol_y = "rY"  # fase amarela para sentido Oeste-Leste
tempos = open("tempos.txt", "a")  # arquivo para armazenar os tempos de
    percurso


##### FASE DE TREINAMENTO ######

# Parametros para semaforo de cima
file1 = 'testes1.data'
qtdVariaveis1 = 7
qtdRes1 = 1
qtdResTipos1 = 2
porcentagemTeste1 = 30
k1 = 3
# Parametros para semaforo da esquerda
file2 = 'testes2.data'
qtdVariaveis2 = 6
```

```
31 qtdRes2 = 1
32 qtdResTipos2 = 2
33 porcentagemTeste2 = 30
34 k2 = 3
35 # Parametros para semaforo do centro
36 file3 = 'testes3.data'
37 qtdVariaveis3 = 8
38 qtdRes3 = 1
39 qtdResTipos3 = 2
40 porcentagemTeste3 = 30
41 k3 = 3
42 # Parametros para semaforo da direita
43 file4 = 'testes4.data'
44 qtdVariaveis4 = 7
45 qtdRes4 = 1
46 qtdResTipos4 = 2
47 porcentagemTeste4 = 30
48 k4 = 3
49 # Parametros para semaforo de baixo
50 file5 = 'testes5.data'
51 qtdVariaveis5 = 8
52 qtdRes5 = 1
53 qtdResTipos5 = 2
54 porcentagemTeste5 = 30
55 k5 = 3
56
57 # Treinamento do semaforo de cima
58 tlc1 = TLControl(file1, qtdVariaveis1, qtdRes1, qtdResTipos1,
       porcentagemTeste1)
59 tlc2.treino()
60 tlc2.exportar('treinamento1.txt')
61 # Treinamento do semaforo da esquerda
62 tlc2 = TLControl(file2, qtdVariaveis2, qtdRes2, qtdResTipos2,
       porcentagemTeste2)
63 tlc2.treino()
64 tlc2.exportar('treinamento2.txt')
65 # Treinamento do semaforo do centro
66 tlc3 = TLControl(file3, qtdVariaveis3, qtdRes3, qtdResTipos3,
       porcentagemTeste3)
67 tlc3.treino()
68 tlc3.exportar('treinamento3.txt')
69 # Treinamento do semaforo da direita
70 tlc4 = TLControl(file4, qtdVariaveis4, qtdRes4, qtdResTipos4,
       porcentagemTeste4)
71 tlc4.treino()
72 tlc4.exportar('treinamento4.txt')
73 # Treinamento do semaforo de baixo
```

```
74 tlc5 = TLControl(file5, qtdVariaveis5, qtdRes5, qtdResTipos5,
       porcentagemTeste5)
75 tlc5.treino()
76 tlc5.exportar('treinamento5.txt')
77
78 if not traci.isEmbedded():
79   # Inicia o TraCI
80   sumoBinary = 'sumo-gui'
81   sumoConfig = "tcc/map.sumo.cfg"
82   traci.start([sumoBinary, "-c", sumoConfig, "-S", "-Q"])
83
84 carros = {}
85 opened = {
86   '0': {'open': 0, 'tempo': 0},
87   '1': {'open': 0, 'tempo': 0},
88   '2': {'open': 0, 'tempo': 0},
89   '3': {'open': 0, 'tempo': 0},
90   '4': {'open': 0, 'tempo': 0},
91 }
92 step = 0
93 simutime = 0
94 while step==0 or traci.simulation.getMinExpectedNumber() > 0:
95   traci.simulationStep()
96
97   # Calcula tempo de percurso dos veiculos
98   time = traci.simulation.getCurrentTime()/1000.0
99   departeds = traci.simulation.getDepartedIDList()
100  for departed in departeds:
101    route = traci.vehicle.getRoute(departed)
102    carros[departed] = {'tempo': time, 'quadras':len(route)}
103  arriveds = traci.simulation.getArrivedIDList()
104  for arrived in arriveds:
105    carros[arrived]['tempo'] = round(time-carros[arrived]['tempo'], 2)
106  if(time-simutime != 1):
107    continue
108  simutime = time
109
110  ruas = {
111    '1': {'id': '128391419#3', 'amb': 0, 'carros': 0, 'espera': 0},
112    '2': {'id': '128391419#4', 'carros': 0},
113    '3': {'id': '128498008#1', 'amb': 0, 'carros': 0, 'espera': 0},
114    '4': {'id': '126576860#3', 'amb': 0, 'carros': 0, 'espera': 0},
115    '5': {'id': '119219397#2', 'amb': 0, 'carros': 0, 'espera': 0},
116    '6': {'id': '370197088#2', 'amb': 0, 'carros': 0, 'espera': 0},
117    '7': {'id': '370197088#1', 'amb': 0, 'carros': 0, 'espera': 0},
118    '8': {'id': '370197088#0', 'amb': 0, 'carros': 0, 'espera': 0},
119    '9': {'id': '126576860#2', 'amb': 0, 'carros': 0, 'espera': 0},
```

```
120     '10': {'id': '119219394#1', 'amb': 0, 'carros': 0, 'espera': 0},
121     '11': {'id': '119219394#2', 'carros': 0},
122     '12': {'id': '126576860#1', 'amb': 0, 'carros': 0, 'espera': 0},
123   }
124
125   # Captacao do numero de carros, prioridade e tempo de espera deles
126   for idx in ruas:
127     ruas[idx]['carros'] = traci.edge.getLastStepVehicleNumber(ruas[idx]['id
        ']) 
128   tls = traci.trafficlights.getIDList()
129   for tl in tls:
130     lanes = traci.trafficlights.getControlledLanes(tl)
131     for lane in lanes:
132       lane_id = lane.split('_')[0]
133       for rua in ruas:
134         if(ruas[rua]['id']!=lane_id):
135           continue
136         break
137       vehicles = traci.lane.getLastStepVehicleIDs(lane)
138       for vehicle in vehicles:
139         classe = traci.vehicle.getVehicleClass(vehicle)
140         if(classe=='emergency'):
141           ruas[rua]['amb'] += 1
142         time = traci.vehicle.getWaitingTime(vehicle)
143         if(ruas[rua]['espera']<time):
144           ruas[rua]['espera'] = int(time)
145
146
147   ##### FASE DE APLICACAO ######
148
149   # Definicao da fase do semaforo de cima
150   tls = tlc1.aplicacao([
151     ruas['1']['amb'], ruas['4']['amb'],
152     ruas['1']['carros'], ruas['2']['carros'], ruas['4']['carros'],
153     ruas['1']['espera'], ruas['4']['espera']
154     ], k1)
155   if(tls[0]==opened['0']['open']):
156     opened['0']['tempo'] += 1
157   else:
158     opened['0']['tempo'] = 1
159   opened['0']['open'] = tls[0]
160   if(tls[0]==0):
161     if(opened['0']['tempo']<3):
162       traci.trafficlights.setRedYellowGreenState("1419087922", tlp_ol_y)
163     else:
164       traci.trafficlights.setRedYellowGreenState("1419087922", tlp_ns_v)
165   else:
```

```
166     if(opened['0']['tempo']<3):
167       traci.trafficlights.setRedYellowGreenState("1419087922", tlp_ns_y)
168     else:
169       traci.trafficlights.setRedYellowGreenState("1419087922", tlp_ol_v)
170
171   # Definicao da fase do semaforo da esquerda
172   tls = tlc2.aplicacao([
173     ruas['3']['amb'], ruas['6']['amb'],
174     ruas['3']['carros'], ruas['6']['carros'],
175     ruas['3']['espera'], ruas['6']['espera']
176     ], k2)
177   if(tls[0]==opened['1']['open']):
178     opened['1']['tempo'] += 1
179   else:
180     opened['1']['tempo'] = 1
181   opened['1']['open'] = tls[0]
182   if(tls[0]==1):
183     if(opened['1']['tempo']<3):
184       traci.trafficlights.setRedYellowGreenState("1512465305", tlp_ol_y)
185     else:
186       traci.trafficlights.setRedYellowGreenState("1512465305", tlp_ns_v)
187   else:
188     if(opened['1']['tempo']<3):
189       traci.trafficlights.setRedYellowGreenState("1512465305", tlp_ns_y)
190     else:
191       traci.trafficlights.setRedYellowGreenState("1512465305", tlp_ol_v)
192
193
194   # Definicao da fase do semaforo do centro
195   tls = tlc3.aplicacao([
196     ruas['7']['amb'], ruas['9']['amb'],
197     ruas['4']['carros'], ruas['6']['carros'], ruas['7']['carros'], ruas
      ['9']['carros'],
198     ruas['7']['espera'], ruas['9']['espera']
199     ], k3)
200   if(tls[0]==opened['2']['open']):
201     opened['2']['tempo'] += 1
202   else:
203     opened['2']['tempo'] = 1
204   opened['2']['open'] = tls[0]
205   if(tls[0]==1):
206     if(opened['2']['tempo']<3):
207       traci.trafficlights.setRedYellowGreenState("1420551146", tlp_ol_y)
208     else:
209       traci.trafficlights.setRedYellowGreenState("1420551146", tlp_ns_v)
210   else:
211     if(opened['2']['tempo']<3):
```

```
212       traci.trafficlights.setRedYellowGreenState("1420551146", tlp_ns_y)
213     else:
214       traci.trafficlights.setRedYellowGreenState("1420551146", tlp_ol_v)
215
216
217   # Definicao da fase do semaforo da direita
218   tls = tlc4.aplicacao([
219     ruas['5']['amb'], ruas['8']['amb'],
220     ruas['5']['carros'], ruas['7']['carros'], ruas['8']['carros'],
221     ruas['5']['espera'], ruas['8']['espera']
222     ], k4)
223   if(tls[0]==opened['3']['open']):
224     opened['3']['tempo'] += 1
225   else:
226     opened['3']['tempo'] = 1
227   opened['3']['open'] = tls[0]
228   if(tls[0]==1):
229     if(opened['3']['tempo']<3):
230       traci.trafficlights.setRedYellowGreenState("1340185323", tlp_ol_y)
231     else:
232       traci.trafficlights.setRedYellowGreenState("1340185323", tlp_ns_v)
233   else:
234     if(opened['3']['tempo']<3):
235       traci.trafficlights.setRedYellowGreenState("1340185323", tlp_ns_y)
236     else:
237       traci.trafficlights.setRedYellowGreenState("1340185323", tlp_ol_v)
238
239
240   # Definicao da fase do semaforo de baixo
241   tls = tlc5.aplicacao([
242     ruas['10']['amb'], ruas['12']['amb'],
243     ruas['9']['carros'], ruas['10']['carros'], ruas['11']['carros'], ruas
      ['12']['carros'],
244     ruas['10']['espera'], ruas['12']['espera']
245     ], k5)
246   if(tls[0]==opened['4']['open']):
247     opened['4']['tempo'] += 1
248   else:
249     opened['4']['tempo'] = 1
250   opened['4']['open'] = tls[0]
251   if(tls[0]==0):
252     if(opened['4']['tempo']<3):
253       traci.trafficlights.setRedYellowGreenState("1422994624", tlp_ol_y)
254     else:
255       traci.trafficlights.setRedYellowGreenState("1422994624", tlp_ns_v)
256   else:
257     if(opened['4']['tempo']<3):
```

```
258         traci.trafficlights.setRedYellowGreenState("1422994624", tlp_ns_y)
259       else:
260         traci.trafficlights.setRedYellowGreenState("1422994624", tlp_ol_v)
261
262
263     step += 1
264
265 ordem = sorted(carros, reverse=True)
266 for x in ordem:
267     tempos.write(str(x)+' -> '+str(carros[x])+'\n')
268 tempos.write('\n')
269
270 tempos.close()
271 traci.close()
272 sys.stdout.flush()
```