UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

CONTROLE ADAPTATIVO DE INTERSEÇÕES: UMA IMPLEMENTAÇÃO APOIADA POR UMA FERRAMENTA DE SIMULAÇÃO PARA O CENÁRIO DE TRANSPORTES

TRABALHO DE GRADUAÇÃO

Emmanuel Katende Dinanga

Santa Maria, RS, Brasil

2014

CONTROLE ADAPTATIVO DE INTERSEÇÕES: UMA IMPLEMENTAÇÃO APOIADA POR UMA FERRAMENTA DE SIMULAÇÃO PARA O CENÁRIO DE TRANSPORTES

Emmanuel Katende Dinanga

Trabalho de Graduação apresentado ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Bacharel em Ciência da Computação

Orientadora: Profª. Dr. Marcia Pasin

Trabalho de Graduação N.368
Santa Maria, RS, Brasil

2014

Katende Dinanga, Emmanuel

Controle adaptativo de Interseções: uma Implementação apoiada por uma Ferramenta de Simulação para o Cenário de Transportes / por Emmanuel Katende Dinanga. – 2014.
71 f.: il.; 30 cm.

Orientadora: Marcia Pasin
Monografia (Graduação) - Universidade Federal de Santa Maria, Centro de Tecnologia, Curso de Bacharelado em Ciência da Computação, RS, 2014.

1. Controle Adaptativo. 2. Interseção. 3. Trânsito. 4. SUMO. I. Pasin, Marcia. II. Título.

E-mail: emmanuel@inf.ufsm.br

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Bacharelado em Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

CONTROLE ADAPTATIVO DE INTERSEÇÕES: UMA IMPLEMENTAÇÃO APOIADA POR UMA FERRAMENTA DE SIMULAÇÃO PARA O CENÁRIO DE TRANSPORTES

elaborado por
Emmanuel Katende Dinanga

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Marcia Pasin, Dr.
(Presidente/Orientador)

Iara Agustin, Dr. (UFSM)

Luis Alvaro de Lima Silva, Dr. (UFSM)

Santa Maria, 21 de Janeiro de 2014.

AGRADECIMENTOS

Agradeço a minha mãe Marie Madelaine tudo o que ela realizou para eu chegar até aqui. Além de uma guerreira, ela sempre foi visionária, enxergando longe e focada no objetivo final. Agradeço todas as bençõs proclamadas na minha vida através das declarações positivas e motivadoras todas as manhãs ao meu sair para o colégio.

Agradeço a minha família pelo grande apoio expressado não somente por palavras, como também por gestos. Vocês sempre acreditaram em mim e sempre me exergaram vitorioso nos meus desafios. Isso faz toda diferença para mim, me servindo de motivação e fôlego para enfrentar as situações da vida.

Agradeço a minha amiga, daqui a pouco noiva, e futura esposa Gabriela V. H. pelo amor, pela força e auxílio em todos os sentidos. Tu sempre foste e continuas presente, me auxiliando no que eu preciso, e me lembrando como eu posso conseguir, acreditando em mim, e fazendo todo o possível para que eu me sinta feliz. Te amo!

Agradeço a minha grande orientadora, Professora Dr^{a}. Márcia Pasin, por ser minha orientadora. Como sempre te digo, tu és mais do que uma orientadora, pois tu abraçaste minha causa e fez disso a tua também, me ensinaste o que eu precisava aprender, e sempre foste paciente comigo nas minhas longas discussões até me convencer.

Agradeço à Professora Dr^{a}. Iara Augustin e ao Professor Dr. Luis Álvaro Silva por terem aceitado o convite para participarem da banca examinadora, e por terem sido grandes e bons professores para mim.

Agradeço a meus amigos e irmãos na fé por terem me abraçado quando cheguei em Santa Maria, quando precisei de tudo para sobreviver. Em particular ao meu companheiro e irmão Guilherme R. Pavanello, por tudo quanto fez.

Por fim, agradeço à UFSM, ao Brasil, e a todos que de perto como de longe, participaram, oraram e torceram para eu conquistar este título cujo presente trabalho faz parte da avaliação parcial. Sempre serei grato a todos vocês. Muito obrigado!

“Porque estou certo de que, nem a morte, nem a vida, nem os anjos, nem os principados, nem as potestades, nem o presente, nem o porvir, nem a altura, nem a profundidade, nem alguma outra criatura nos poderá separar do amor de Deus, que está em Cristo Jesus nosso Senhor. ”

— ROMANOS 8:38-39

RESUMO

Trabalho de Graduação
Curso de Bacharelado em Ciência da Computação
Universidade Federal de Santa Maria

CONTROLE ADAPTATIVO DE INTERSEÇÕES: UMA IMPLEMENTAÇÃO APOIADA POR UMA FERRAMENTA DE SIMULAÇÃO PARA O CENÁRIO DE TRANSPORTES

AUTOR: EMMANUEL KATENDE DINANGA
ORIENTADORA: MARCIA PASIN
Local da Defesa e Data: Santa Maria, 21 de Janeiro de 2014.

Este trabalho de graduação propõe a implementação e avaliação de diferentes políticas para controlar a passagem de veículos por interseções entre vias em redes de transporte. A avaliação foi realizada no sentido de comparar e investigar qual política é mais adequada no ponto de vista de equitabilidade, dados diferentes estados de trânsito. Primeiro, é construída uma rede viária com duas vias e uma interseção, e são definidos diferentes estados do trânsito baseados na preferência entre vias na passagem pela interseção, na frequência do fluxo de cada via, por fim, na velocidade máxima que veículos que transitam em cada via pode alcançar. Em seguida, foram implementadas diferentes políticas para controlar a passagem dos veículos pela interseção, com suporte de simulação computacional. No final, as políticas são aplicadas, a cada um dos estados de trânsito definidos na simulação. Pode ser concluído, através da execução de experimentos, que diferentes políticas são mais adequadas, do ponto de vista de equitabilidade, para cada estado do trânsito. Também, foi observado que uma mesma política pode se mostrar mais adequada na distribuição do fluxo de um estado, e menos adequada na distribuição de outro fluxo, indicando, desta forma, que políticas híbridas ainda precisam ser propostas e analisadas. A ferramenta usada para simulação foi o SUMO, simulador específico para redes de transporte.

Palavras-chave: Controle Adaptativo. Interseção. Trânsito. SUMO.

ABSTRACT

Undergraduate Final Work
Post-Graduate Program in Informatics
Federal University of Santa Maria

INTERSECTION ADAPTIVE CONTROL: AN IMPLEMENTATION SUPPORTED BY A SIMULATION TOOL FOR THE TRANSPORTATION SCENARIO

AUTHOR: EMMANUEL KATENDE DINANGA
ADVISOR: MARCIA PASIN
Defense Place and Date: Santa Maria, March 21st, 2014.

This work proposes the implementation and evaluation of different policies to intersection control concerning vehicles in transportation networks. The evaluation was conducted in order to compare and investigate which policy is more appropriate regarding equitability, for each defined traffic state. First of all, a road network was constructed with two routes and an intersection, and different traffic states were defined based on the priority of a given route from another, the flow rate of each route, finally, the maximum speed that vehicles traveling on each route may achieve. Then different policies to intersection control were implemented with support of computational simulation. Lastly, policies were applied to each state of traffic flow defined in the simulation. According to the experiments, it can be concluded that different policies are more appropriate for each defined state of traffic flow, regarding equitability. Likewise, it was observed that a given policy can be more appropriate in distributing a state of traffic flow and less appropriate with regard to another state, indicating that hybrid policies still need to be proposed and analyzed. SUMO is a specific simulator for transportation networks used as the simulation tool.

Keywords: Adaptive Control, Intersection, Traffic, SUMO.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE APÊNDICES

LISTA DE ABREVIATURAS E SIGLAS

CTB	*Código do Trânsito Brasileiro*
SUMO	*Simulation of Urban MObility*
VANETs	*Vehicular Ad Hoc NETworks*
TraCI	*Traffic Control Interface*
TraCI4J	*Traffic Control Interface for Java*
V2I	*Vehicle to Infrastructure*
V2V	*Vehicle to Vehicle*

LISTA DE SÍMBOLOS

A	Algoritmo
E	Estado do Fluxo do Trânsito
C	Cenário
G	Grupo de Cenários
P	Interseção
T	Taxa de Variação
X	Taxa Total de Variação
dX	Diferença das Taxas Totais de Variação

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

Soluções para provimento de trânsito eficiente e mobilidade urbana são assuntos que vem ganhando destaque em nosso cotidiano. Pesquisas no setor do transporte têm se focado com mais ênfase em otimizar fluxo de pessoas e produtos transportados (KHORANI V.; RAZAVI, 2011). Vários fatores podem ser destacados para explicar os diferentes cenários caóticos do trânsito, entre outros: o crescimento significativo do número de veículos em circulação devido ao aumento da demanda de mobilidade urbana, a falta de uma infraestrutura satisfatória para suprir essa demanda crescente, carrência no uso de infraestruturas e políticas adequadas para o controle da circulação dos veículos nas vias, e outros problemas de detecção da variação do fluxo do trânsito no âmbito de aplicar políticas adequadas para gerenciar os diferentes estados, etc.

Com base nos fatos observados e vivenciados no dia-a-dia, a aplicação de mecanismos para melhorar o gerenciamento do trânsito tornam-se imprescindíveis. Opções, neste cenário quase caótico incluem: (i) aplicação de novas políticas, como limitar a quantidade de veículos em circulação, alocação de faixas exclusivas para ônibus, (ii) tornar o transporte público mais atrativo, e (iii) investir em infraestrutura, como construção de novas vias e o emprego de tecnologias adequadas para análise e gerenciamento do trânsito.

Pesquisas sobre Sistemas Inteligentes de Transporte (ITS) propõem dois tipos principais de arquitetura de comunicação para gerenciamento dos elementos que fazem parte do trânsito: a comunicação veículo-a-veículo $(V2V)$, onde veículos são parte de uma rede e comunicam-se com outros veículos mais próximos, e a veículo-a-infraestrutura $(V2I)$ que permite que veículos se comuniquem com infraestruturas das estradas no âmbito de fornecer dados tais como: a velocidade do veículo, sua locação, etc (DEZANI H.; GOMES, 2012).

De um modo inicial, investigar e simular diferentes estados do trânsito de um lado, e implementar políticas para controlar fluxos do trânsitos utilizando arquiteturas $V2I$ e $V2V$ do outro, no âmbito de avaliar as mesmas aplicando-as aos estados do trânsito definidos e observando os resultados gerados ajudaria a descobrir, para cada estado do trânsito, quais as políticas adequadas para o seu gerenciamento.

1.2 Objetivos

Objetivo geral

Este projeto de pesquisa propõe a definição de diferentes estados do trânsito e a implementação de diferentes políticas para controlar, de um modo inicial, a passagem dos veículos por interseções entre duas vias. O que se busca é a aplicação dessas políticas para descobrir quais delas são mais adequadas para distribuir a passagem dos veículos pelas interseções - isto é - a investigação é realizada em termos de equitabilidade, dado cada estado de trânsito.

Para este fim, é usada o SUMO *Simulation of Urban Mobility*, uma ferramenta de simulação de redes viárias para o cenário de transporte, na qual são definidos os diferentes estados de trânsito e aplicadas as políticas implementadas para avaliar o comportamento de cada estado definido.

Para atingir o objetivo do trabalho, precisa-se realizar os seguintes passos:

- Estudo e escolha de algoritmos para controle adaptivo com suporte de comunicação $V2V$ e $V2I$;
- Estudo do simulador SUMO;
- Implementação de algoritmos para controle adaptivo com suporte de comunicação $V2V$ e $V2I$;
- Validação experimental dos algoritmos implementados em diferentes cenários;
- Avaliação e comparação dos algoritmos considerando equitabilidade;
- Escrita de relatório final em formato de Trabalho de Graduação;
- Geração de material para publicação de resultados em eventos científicos e/ou revistas.

1.3 Organização do texto

O texto está organizado como segue. O capítulo 2 apresenta uma revisão da literatura. O objetivo no capítulo 2 é realizar uma análise das publicações correntes na área de controle e otimização do trânsito e áreas afins, de modo a entender como o problema de controle e otimização do trânsito é abordado, e destacar a contribuição deste trabalho no assunto.

O capítulo 3 descreve resumidamente o SUMO, uma ferramenta de simulação para o cenário de transporte. A ideia no capítulo 3 é introduzir o SUMO, citando os autores e as razões da sua criação. Em seguida, explicar suscintamente sua instalação e uso. Por fim, descrever como se contrói uma simulação usando o SUMO, explicando com ilutração como cada elemento da simulação é utilizado, baseado no contexto deste trabalho.

O capítulo 4 descreve os algoritmos escolhidos para controle de interseções. No capítulo 4, inicia-se com a contextualização de certos termos usados neste trabalho. Em seguida, descreve-se cada algoritmo de modo genérico.

O capítulo 5 aborda a implementação dos algoritmos descritos no capítulo 4. O objetivo é detalhar como foi implementado cada algoritmo através de pseudo-códigos. Descreve-se primeiro as bibliotecas e classes adicionais usadas para a implementação dos algoritmos, seguidos dos pseudo-códigos. Todo o código fonte deste trabalho costa nos anexos.

O capítulo 6 explica a avaliação experimental realizada neste trabalho. O objetivo é criar inicialmente diferentes estados do trânsito a partir da modificação do fluxo nas vias, para aplicar, em seguida, cada algoritmo descrito nos capítulos 4 e 5 a estes estados. Por fim, comparar os resultados usando testes estatísticos adequados a este contexto.

A aplicação dos algoritmos a estados do trânsito gera *cenários* representados em forma de gráficos bidimencionais (*tempo / número de veículos nos segmentos das vias*). Os resultados dos testes estatísticos são elencados em uma tabela comparativa de forma a investigar, de modo inicial, quais os algoritmos mais adequados para cada estado do trânsito, de ponto de vista *equitabilidade do fluxo nas vias*. Em seguida, investiga-se levando-se em os esses algoritmos mais equitáveis, qual é o melhor algoritmo para cada estado do trânsito.

O capítulo 7 resume as conclusões de acordo com os resultados da avaliação experimental realizada. Além do mais, elenca-se os possíveis trabalhos futuros a serem realizados no âmbito de trazer mais contribuição no controle e otimização do trânsito.

2 REVISÃO DA LITERATURA

Agentes autônomos tem sido frequentemente usados como base para modelar e simular os efeitos da aplicação de VANETs em redes de transporte, por exemplo, para modelar o comportamento dos motoristas, e benefícios de uso de informação para reduzir tempos de percurso e para tomada de decisão em interseções. Em (DRESNER; STONE., 2004), um veículo que deseja atravessar uma interseção reserva um *slot* de tempo e espaço em uma central. Se o veículo ganha o *slot*, pode passar a interseção. Em comparação com o semáforo convencional, esta política de alocação possui melhores resultados para o escoamento de tráfego. Entretanto, se o veículo não conseguir reservar um *slot*, pode sofrer espera indefinida. Além disso, com o uso de uma central para controlar a política de passagem em todos os semáforos, há ponto único de falha.

(VASIRANI; OSSOWSKI., 2009) aplica a solução de Dresner & Stone para tratar uma rede de interseções. A ideia é oferecer um serviço adequado ao coletivo, com uma solução inspirada no mercado de ações. De fato, esta ideia é similar ao algoritmo do banqueiro para alocação de recursos de Dijkstra (1965). Veículos negociam com a infraestrutura para alocar múltiplos *slots*, mas ainda sem comunicação V2V. Se a ação de alocar um *slot* não pode ser satisfeita, quando o veículo atinge a interseção, ele precisa aguardar e tentar uma nova reserva. Comunicação entre os agentes que controlam as interseções é permitida.

(KRAJZEWICZ et al., 2005) compara tamanhos de filas de veículos para passagem de interseções em uma simulação. A fila maior tem maior prioridade sobre uma fila menor. De fato, a política que prioriza a fila maior tem maior *throughput* em relação à política que prioriza a fila menor. Na política que prioriza a fila maior, um número menor de carros percebe a penalidade de chaveamento de passagem de veículos de uma fila para outra. (MUGNELA; NETTO, 2012) apresenta um enfoque diferente, onde o fluxo do trânsito foi medido manualmente em uma via real, avaliado e, então, melhorias para a calibração semafórica no que diz respeito a valores de janelas de tempo foram propostas pelo uso de algoritmos genéticos. Também nesses trabalhos, comunicação veicular não é levada em conta.

Em (GRADINESCU et al., 2007) semáforos adaptativos são propostos com o apoio de comunicação V2V. Veículos periodicamente transmitem para o semáforo informações sobre eles próprios e sobre os demais veículos. O tempo de apresentação da cor verde é calculada com base no volume de veículos de determinada fila e na estimativa de atraso. A avaliação

experimental demonstrou que a solução adaptativa apresenta melhora de quase 30% nos atrasos de viagens.

Finalmente, em (FERREIRA et al., 2010), através do suporte de comunicação V2V e de AVL, o veículo mais próximo de uma interseção é eleito para coordenar a passagem de veículos em uma determinada interseção. Quando o coordenador finalmente passa a interseção, um novo veículo é escolhido para gerenciar a interseção. Entretanto, dado que dois veículos v_i e v_j podem estar em vias distintas q_m e q_n, mas possuir mesma distância d em relação à interseção, uma garantia de eleição de coordenador único precisa ser imposta. Além disso, o artigo não descreve a política adotada para controlar o tempo de passagem de veículos na interseção.

3 SUMO: SIMULATION OF URBAN MOBILITY

3.1 Introdução ao SUMO

SUMO (KRAJZEWICZ et al., 2012) é um simulador de trânsito urbano, desenvolvido pelo Instituto de Pesquisa do Transporte do Centro Aeroespacial da Alemanha, disponibilizado em código aberto, que permite simular desde um pequeno número de veículos se movendo em uma rede de transporte, até grandes circulações de trânsitos suportadas por controles semafóricos.

O desenvolvimento do SUMO iniciou em 2000, com o objetivo de apoiar a comunidade de pesquisa do transporte através de uma ferramenta capaz de implementar e avaliar algoritmos de redes de transporte (KRAJZEWICZ et al., 2012). Hoje o SUMO está na sua versão 0.18.0 lançada em 28 de Agosto de 2013.

3.2 Instalação e Uso

O SUMO possui dois formatos: o código fonte e o compilado.

Para executar e usar a aplicação apenas, existe a versão compilada contendo os pacotes necessários, disponível em `http://sumo-sim.org`. Esta versão é adequada para usuários em geral dos sistemas operacionais *Windows* e *Linux*. É preciso instalar o *Python (2.5 ou 2.6)* para executar $scripts$ adicionais dentro do pacote *tool*, (KRAJZEWICZ et al., 2012).

Desenvolvedores podem baixar os pacotes fontes SUMO para o sistema operacional *Windows*, para *Linux*, e para *MacOS*. Salientando que ainda não há versão compilada do SUMO para o *MacOS*.

3.3 Construção de uma simulação

SUMO possui vários elementos, funcionalidades, e maneiras para construção e configuração de redes e simulações de trânsito. Aqui serão definidos e explicados apenas os elementos e as funcionalidades que fazem parte do escopo deste trabalho. Mais informações a respeito das funcionalidades que SUMO oferece podem ser encontradas em `http://sumo-sim.org`.

Uma simulação no SUMO ocorre dentro de um intervalo de tempo cuja unidade é o $Step$, que significa *Passo* em português, e que por default no SUMO GUI, é equivalente a 1

segundo, mas pode ser alterado baseado na necessidade do desenvolvedor.

Uma rede de vias em uma simulação no SUMO é basicamente composta dos seguintes objetos: ***nodes*** (nodos ou junções) e ***edges*** (setas ou $links$, ou também segmentos de vias, conectam as junções), ***routes*** (rotas ou vias), ***vehicles*** (veículos), além de outros objetos que fazem parte do trânsito. A Figura 3.1 apresenta a ilustração de uma rede de vias destacando cada objeto a ser definido.

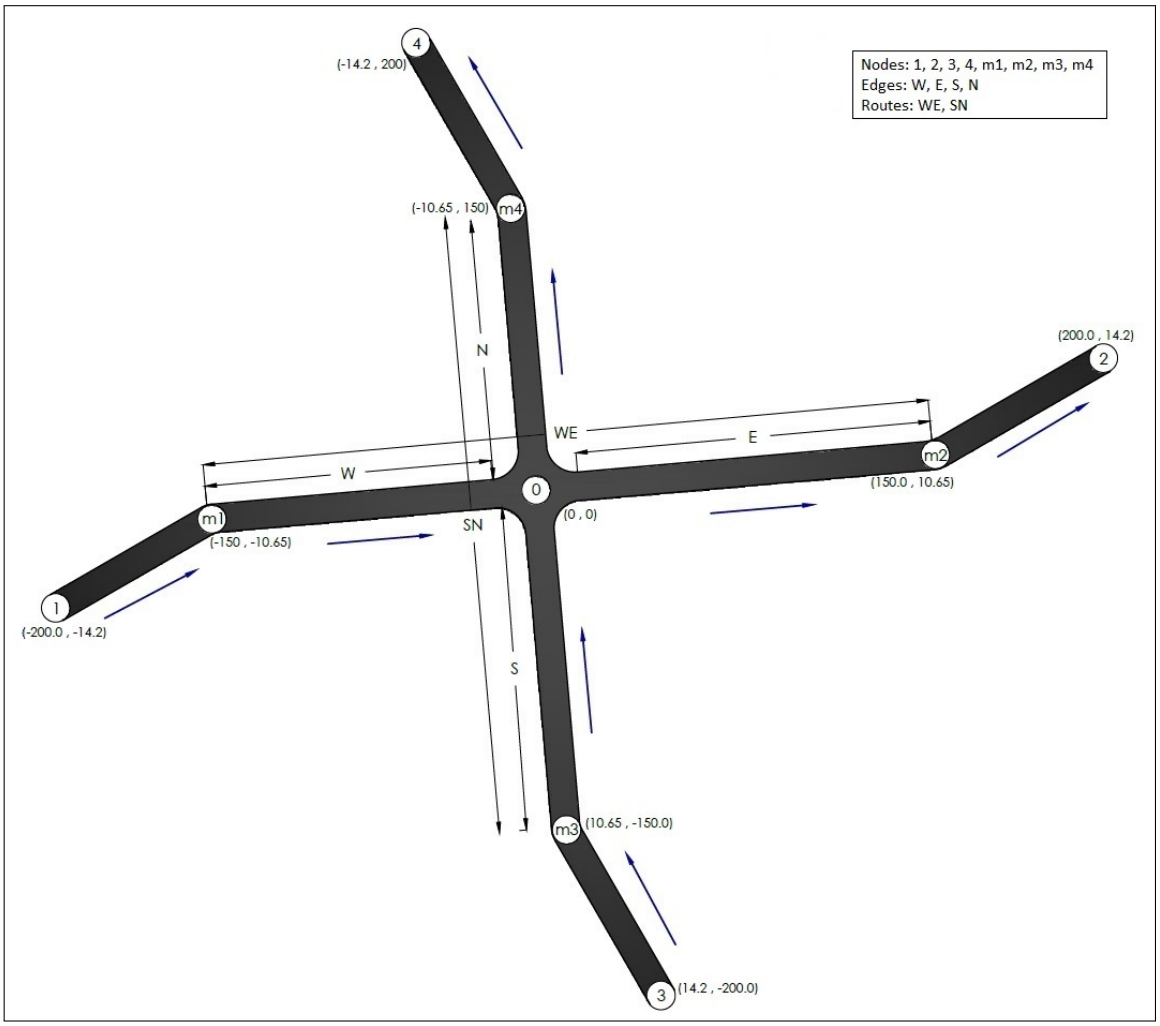

Figura 3.1 – Exemplo de uma rede de vias construída no SUMO

Para construir uma rede de vias tal como mostrado na Figura 3.1, e desenvolver uma simulação usando o SUMO, é preciso gerar e configurar arquivos em *eXtensible Markup Language (XML)*, representando os diferentes objetos. Cada elemento da rede de vias na Figura 3.1 possui atributos e será descrito a seguir. Serão enfocados apenas os atributos que fazem parte do escopo deste trabalho.

Nodos

Os pontos de origens, interseções, e destinos de vias são vistos como localizações, e respectivamente representados como nodos em uma rede viária. Os nodos são criados e salvos em um arquivo (*[. . .].nod.xml*) com vários atributos entre os quais:

- id: (*String*), representa o identificador do nodo na rede.
- x: (*float*), representa a coordenada x de localização do nodo na rede, definida em metros.
- y: (*float*), representa a coordenada y de localização do nodo na rede, definida em metros.
- $type$: (*enum "priority", "traffic_light"*), representa um tipo opcional do nodo. *priority* expressa que veículos que estão em segmentos de baixas prioridades devem esperar até que veículos em segmentos de altas prioridades passem primeiro. *traffic_light* significa que a interseção das duas vias que passam por esse nodo é controlada por semáforos.

A Figura 3.2 apresenta o exemplo de vários nodos definidos em um arquivo *[. . .].nod.xml*.

```
<nodes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/nodes_file.xsd»

l1. <node id="0"x="0.0"y="0.0"type="priority"/>

l2. <node id="1"x=-200.0"y=-14.2"type="priority"/>
l3. <node id="2"x="+200.0"y="14.2"type="priority"/>
l4. <node id="3"x="14.2"y=-200.0"type="priority"/>
l5. <node id="4"x=-14.2"y="+200.0"type="priority"/>

l6. <node id="m1"x=-150.0"y=-10.65"type="priority"/>
l7. <node id="m2"x="+150.0"y="10.65"type="priority"/>
l8. <node id="m3"x="10.65"y=-150.0"type="priority"/>
l9. <node id="m4"x=-10.65"y="+150.0"type="priority"/>

</nodes>
```

Figura 3.2 – Exemplo da configuração de um arquivo *.nod.xm*

Na Figura 3.2, o nodo 0 encontra-se na posição central, em coordenadas (0, 0) (ver $l1$), com relação aos demais nodos. O grupo dos nodos (1, 2, 3, 4) localizam-se respectivamente na posição sul-oeste, norte-este, sul-este, norte-oeste (ver $l2 - l5$). E o grupo dos nodos (m1, m2,

m3, m4) localizam-se respectivamente na posição oeste, este, sul, norte (ver $l6 - l9$). Todos os nodos são do tipo *priority*.

Segmentos de vias

Os segmentos de vias, ou setas, são representadas como $links$ e construídas conectando os nodos e especificando o número de faixas (*lanes*) dentro delas. Segmentos de vias são criados e configurados em um arquivo *[...].edge.xml* contendo os seguintes atributos:

- id (*String*): representa o identificador do segmento na rede;
- $from$ (*String*): representa o *id* do nodo origem do segmento;
- to (*String*): representa o *id* do nodo destino do segmento;
- $numlanes$ (*int*): representa o número de faixas que o segmento possui;
- $speed$ (*float*): representa a velocidade máxima que um dado veículo pode alcançar no segmento;
- $priority$ (*int*): representa a prioridade da seta do segmento na rede.

A Figura 3.3 apresenta um exemplo de rede de transporte com vários segmentos de vias definidos em um arquivo *[...].edg.xml*.

Na configuração definida na Figura 3.3 cada segmento possui somente uma faixa, ou seja $numLanes = "1"$. Os segmentos *w0, e0, s0, n0* possuem velocidades máxima e prioridades iguais (ver $l2$, $l5$, $l8$ e $l11$) sendo estas inferiores as dos segmentos *w, e, s, n*, que são as mesmas velocidades (ver $l3$, $l6$, $l9$ e $l12$). Cada segmento possui um nodo de origem e de destino, que correspondem aos nodos definidos na Figura 3.2.

Geração de rede viária

Em SUMO, a conexão de nodos por segmentos de vias mapeia uma rede de trânsito urbano. Após a criação e configuração dos arquivos XML ilustrados nas Figuras 3.2- 3.3, estes são compilados para gerar um arquivo *[...].net.xml*, (*net* de *network*), que descreve a rede.

A compilação dos arquivos pode ser realizada de duas maneiras utilizando o *NET_CONVERT*, que é uma ferramenta do SUMO para gerar redes viárias. Para compilar, precisa primeiramente setar o caminho do *NET_CONVERT* na *classpath*. Como também, precisa

```
<edges xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/edges_file.xsd»

l1. <!– WEST EDGES–>
l2. <edge id="w0"from="1"to="m1"priority="1"numLanes="1"speed="11.1"/>
l3. <edge id="w"from="m1"to="0"priority="2"numLanes="1"speed="16.7"/>

l4. <!– EST EDGES–>
l5. <edge id="e0"from="m2"to="2"priority="1"numLanes="1"speed="11.1"/>
l6. <edge id="e"from="0"to="m2"priority="2"numLanes="1"speed="16.7"/>

l7. <!– SOUTH EDGES–>
l8. <edge id="s0"from="3"to="m3"priority="1"numLanes="1"speed="11.1"/>
l9. <edge id="s"from="m3"to="0"priority="3"numLanes="1"speed="16.7"/>

l10. <!– NORTH EDGES–>
l11. <edge id="n0"from="m4"to="4"priority="1"numLanes="1"speed="11.1"/>
l12. <edge id="n"from="0"to="m4"priority="3"numLanes="1"speed="16.7"/>

</edges>
```

Figura 3.3 – Exemplo da configuração de um arquivo *.edg.xm*

setar o caminho . . . /sumo[. . .]/bin no **PATH** da viariável do ambiente do sistema para compilar e executar tudo que estiver ligado ao SUMO.

A primeira maneira consiste na digitação direta da linha do comando: *netconvert - -node-files=[. . .].nod.xml - -edge-files=[. . .].edg.xml –output-file=[. . .].net.xml*. Esta linha de comando fornece para o *NET CONVERT* como entrada os arquivos XML dos nodos e setas, e pede para este gerar como saida o arquivo XML da rede viária correspondente.

A segunda maneira de compilação consiste em criar outro arquivo *[. . .].net.cfg* onde informa-se os arquivos XML de nodos e segmentos de vias como entradas, e especifica-se o arquivo XML da rede viária correspondente como saída.

A Figura 3.4 apresenta o exemplo da configuração de um arquivo *[. . .].net.cfg*.

```
l1. <configuration>
l2. <node-files value="cross.nod.xml"/>
l3. <edge-files value="cross.edg.xml"/>
l4. <output value="cross.net.xml"/>
l5. <no-turnarounds value="true"/>
l6. </configuration>
```

Figura 3.4 – Exemplo da configuração de um arquivo *.net.cfg*

Observa-se na Figura 3.4 que as $tags$ <*node-files*> (ver $l2$) e <*edges-files*> (ver $l3$) especificam respectivamente os arquivos dos nodos e segmentos de vias correspondentes. Enquanto a tag *output* (ver $l4$) indica o arquivo da rede viária em si como saída.

Vias

As vias no SUMO são construidas unificando vários segmentos de vias vistos acima. A sequência dos segmentos de vias especificada na construção de uma via define o tragetório do percurso dos veículos nessa via. As vias são criadas e configuradas em um arquivo *[. . .].rou.xml* com $tags$ especificando atributos tais como: <*vTypeDistribution*>, <*route*>, e <*flow*>.

<*vTypeDistribution*> diz respeito aos tipos de veículos que vão transitar nessas vias, e possui por sua vez a tag *vType* que será abordada na subsseção do veículo.

<*route*> especifica a sequência dos segmentos de vias que definem o percurso que os veículos irão seguir. Ele possui os seguintes atributos:

- id: (*String*), representa o identificador da via.
- $edges$: (*String*), representam os *ids* dos respectivos segmentos que compõem essa via.

Por fim, <*flow*> especifica o fluxo em uma dada via, e é composto pelos seguintes atributos:

- id: (*String*), representa o identificador do fluxo.
- $route$: (*String*), representa o identificador da via.
- $begin$: (*int*), representa o número inicial de carros que irão transitar na via.
- end: (*int*), representa o número final dos carros que irão transitar na via.
- $period$: (*int*), representa o intervalo de tempo entre o início do percurso de um veículo e do outro na via.

A Figura 3.5 apresenta o exemplo da configuração de um arquivo *[. . .].rou.xml*.

Observa-se na Figura 3.5 que as $tags$ <*vTypeDistribution*>, <*route*>, e <*flow*> especificam respectivamente os tipos de veículos, via, e fluxo das vias definidas (ver $l3$, $l12$, e $l13$). Há duas vias: *WE* sentido do oeste para o leste, e *SN*, sentido sul para o norte (ver $l12$ e $l15$). Os fluxos das duas vias possuem o mesmo período, e o mesmo número inicial e final dos veículos

```
l1. <routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
l2. xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/routes_file.xsd»

l3. <vTypeDistribution id="DEFAULT_VEHTYPE»

l4. <vType length="5.0"accel="3.0"decel="6.0"maxSpeed="22.2"sigma="0.5"id="passenger"
l5. minGap="2.5"guiShape="passenger"osgFile="vw_golf.3ds"probability="13"/>

l6. <vType length="12.5"accel="2.0"decel="6.0"maxSpeed="16.7"sigma="0.5"id="bus"
l7. minGap="2.5"guiShape="bus"osgFile="tour_bus.3ds"width="2.4"probability="3"/>

l8. <vType length="16.2"accel="1.0"decel="5.0"maxSpeed="11.1"sigma="0.5"
l9. id="transport/trailer"minGap="2.5"guiShape="transport/trailer"osgFile="lorry.3ds"
l10. width="2.6"probability="1"/>

l11. </vTypeDistribution>

l12. <!– WEST TO EAST ROUTE–>
l13. <route id="WE"edges="w0 w e e0"/>
l14. <flow id="WE"route="WE"begin="0"end="9000"period="1"/>

l15. <!– SOUTH TO NORTH ROUTE–>
l16. <route id="SN"edges="s0 s n n0"/>
l17. <flow id="SN"route="SN"begin="0"end="9000"period="1"/>

l18. </routes>
```

Figura 3.5 – Exemplo da configuração de um arquivo *.rou.xml*

que transitam nelas (ver $l13$ e $l16$).

Veículos

Os veículos em SUMO são definidos de várias formas. No caso deste trabalho, há a especificação dos tipos de veículos e o número inicial e final dos veículos que transitam nas vias. Esses dados são definidos no arquivo das vias. A distribuição dos tipos de veículos é especificada na tag <*vTypeDistribution*> como explicado acima, e possui o seguinte atributo:

- id: (*enum*), representa o tipo de veículo.

E esta é composta por tipos de veículos específicos, definidos pelas $tags$ <*vType*> com os seguintes atributos:

- id: (*enum*), representa o identificador do tipo de veículo.

- $length$: (*double*), representa o comprimento do veículo em metros (m).
- $accel$: (*double*), representa a aceleração do veículo em metro quadrado por segundo (m^2/s).
- $decel$: (*double*), representa a deceleração do veículo em metro quadrado por segundos (m^2/s).
- $maxSpeed$: (*double*), representa a velocidade máxima do veículo em metro por segundo (m/s).
- $sigma$: (*double*), representa erros do condutor na direção do veículo, entre 0 e 1.
- $probability$: (*int*), representa a probabilidade da ocorrência do veículo no trânsito.

A Figura 3.6 apresenta o exemplo da configuração da tag $< vTypeDistribuition >$ no arquivo *[. . .].rou.xml*.

```
l1. <vTypeDistribution id="DEFAULT_VEHTYPE»

l2. <vType length="5.0"accel="3.0"decel="6.0"maxSpeed="22.2"sigma="0.5"id="passenger"
l3. minGap="2.5"guiShape="passenger"osgFile="vw_golf.3ds"probability="13"/>

l4. <vType length="12.5"accel="2.0"decel="6.0"maxSpeed="16.7"sigma="0.5"id="bus"
l5. minGap="2.5"guiShape="bus"osgFile="tour_bus.3ds"width="2.4"probability="3"/>

l6. <vType length="16.2"accel="1.0"decel="5.0"maxSpeed="11.1"sigma="0.5"
l7. id="transport/trailer"minGap="2.5"guiShape="transport/trailer"osgFile="lorry.3ds"
l8. width="2.6"probability="1"/>

l9. </vTypeDistribution>
```

Figura 3.6 – Exemplo da configuração dos veículos em um arquivo *.rou.xml*

Observa-se na Figura 3.6 que há três tipos diferentes de veículos. O primeiro é do tipo $passenger$, que corresponde a carros comuns, possui o menor comprimento de todos, mas com mais velocidade, aceleração, e probabilidade de ocorrência (ver $l3$). O segundo tipo é bus, correspondendo a ónibus, é mais comprido que o carro comum, possui mais probabilidade que o terceiro tipo (ver $l4$). O terceiro tipo é $transport/trailer$, correspondea caminhões, é o mais comprido de todos e o menos veloz também. Além de ter bem menos probabilidade de ocorrência do que os outros (ver $l7$).

4 ALGORITMOS

4.1 Conceituação

Antes de descrever os algoritmos, precisa-se definir, de um modo inicial, certos conceitos de acordo com o contexto deste trabalho.

Definição 1 O Controle Semafórico diz respeito ao controle da circulação dos veículos no trânsito, realizado por semáforos.

Definição 2 O controle veicular consiste no controle da circulação dos veículos realizado por dispositivos inteligentes embutidos nos próprios veículos, que se comunicam entre-si baseado em uma política do trânsito definida. Controle veicular pode ser usado para reduzir os tempos de viagens, para reduzir congestionamentos, e para evitar acidentes.

Definição 3 Uma interseção é definida como o ponto de encontro entre duas vias, onde precisa ser estabelecida uma política de controle, seja semafórico ou veicular, para a passagem dos veículos.

Definição 4 Uma região de conflito diz respeito à região da via antes da interseção, onde somente os veículos que estão na via com maior prioridade podem se deslocar, independente da política estabelecida, quando o controle da interseção é veicular.

Definição 5 Uma linha de retenção l em um segmento de via s diz respeito à linha que separa a região de conflito do resto de s. Essa linha é invisível, mas sabe-se que ela foi cruzada por um veículo v em s quando a distância entre v e o ponto de interseção das vias se torna superior ou igual ao comprimento da região de conflito.

Definição 6 Uma fila de veículos, ou simplesmente *fila*, constitue-se da sequência de um a muitos veículos em um segmento de via.

4.2 Descrição dos Algoritmos

No escopo deste trabalho, foram implementados cinco algoritmos para o controle das interseções, entre os quais: o algoritmo *Default*, que reproduz a política de passagem pelas interseções das vias gerada pela configuração padrão do SUMO, por isso o nome *default*, o

algoritmo do *Semáforo* cujo controle é baseado em cores apresentadas de acordo com ciclo e fatia, o algoritmo de *Maior fila sempre*, de *Maior fila primeiro*, e o de *No mínimo k veículos por vez*, que implementam o controle veicular.

A descrição de cada algoritmo é seguida da explicação da razão da sua escolha, baseado na relação da política implementada pelo algoritmo com o trânsito (fluxo do trânsito), como também nos resultados esperados. Estes algoritmos serão descritos a seguir.

4.2.1 Algoritmo Default

O *algoritmo default* resulta da configuração do fluxo do trânsito nas vias feita no SUMO para geração da simulação. Este algoritmo é baseado na atribuição da maior prioridade para passagem dos veículos pelas interseções é via à direita.

Considerando dois segmentos de vias s e w que se encontram em um ponto de interseção P, e suponhamos que a maior prioridade da passagem dos veículos por P seja atribuída a s. O *algoritmo default* faz com que todas as vezes que veículos em s cruzarem a linha de retenção, isso leve os veículos em w a pararem até esses veículos em s passarem por P. Os veículos em w só passam por P quando não há veículos dentro da região de conflito em s.

A política da *prioridade à direita* implementada pelo algoritmo *Default* Sfoi escolhida pelo fato de que ela é usada convencionalmente para controlar a passagem dos veículos nas interseções não sinalizadas, de acordo com o Código do Trânsito Brasileiro (CTB) (REF). A partir da avaliação desta política, poder-se-á descobrir se a mesma é adequada para todos os tipos de estados de trânsito, ou, se existem estados de trânsitos para os quais ela se torna prejudicial.

4.2.2 Semáforo

O algoritmo do *semáforo* reproduz a mesma política que semáforos tradicionais utilizam para controlar a passagem dos veículos nas interseções das vias.

Basea-se no escalonamento de fases semafóricas corelacionadas a intervalos de tempo (fatia ou fases). Entre as fases tem-se: a fase $verde$ ou sinal verde, na qual os veículos que se encontram na respectiva via podem seguir em frente, a $amarela$, levando os veículos da respectiva via a diminuir a velocidade para uma futura parada, por fim, a $vermelha$, onde há parada total dos veículos na respectiva via. Veículos dispostos em cada via recebem a exposição das cores em um determinado ciclo que se repete.

Considerando dois segmentos de vias s e w que se encontram em um ponto de interseção

P, o algoritmo do semáforo inicia carregando na memória uma sequência de fases semafóricas para um dos dois segmentos de vias. Suponhamos que seja para s, se a fase carregada no topo da sequência para s é $verde$ ou $amarela$, w adquiri automaticamente a fase $vermelha$. Caso contrário, w adquiri a fase $verde$, e assim por diante, até terminar o tempo da simulação ou chegar ao fim das sequências semafóricas.

Se w, por exemplo, está na fase $verde$, passa primeiro para a $amarela$, e depois, para a $vermelha$. Enquanto isso, s permanece na $vermelha$ e passa para a $verde$ somente depois de w estar na $vermelha$. Desta forma, as mudanças de fases formam um cíclo definido, de acordo com uma temporização, como: de $verde$ para $amarela$, de $amarela$ para $vermelha$, por fim, de $vermelha$ para $verde$.

A política do *semáforo* foi escolhida pelo fato de ela ser usada por semáforos tradicionais para controlar a passagem dos veículos nas interseções das vias. A avaliação da política do *semáforo*, poderá ajudar a descobrir se a mesma é adequada para todos os tipos de estados do trânsito, ou, se existem estados do trânsito para os quais ela se torna prejudicial.

4.2.3 Maior Fila Sempre

A política definida no algoritmo da *maior fila sempre* consiste em atribuir maior prioridade da passagem pela interseção ao segmento de via com maior fila de veículos fora da região de conflito.

Considerando dois segmentos de vias s e w que se encontram em um ponto de interseção P, o algoritmo da *maior fila sempre* inicia capturando todos os veículos fora da região de conflito em s e w, calculando o número dos veículos em cada segmento de via, e comparando os dois valores. Se s tiver mais veículos, o veículo no topo da fila capturada em w para na linha de retenção, levando os demais veículos da fila em w a parar também, até o último veículo da fila capturada em s passar pela interseção. Em seguida, volta a compara as próximas filas nos dois segmentos. O mesmo processo é repetido até o fim da simulação.

A política da *Maior fila sempre* foi escolhida com a espectativa de observar como ela se comportará na distribuição da passagem dos veículos pela interseção de vias com fluxos de trânsito de frequências parecidas. Pela sua descrição, pode-se entender que quando uma via possui a maior fila de veículos e tem a permissão de passar, automaticamente a fila da outra via possuirá a maior fila na próxima comparação, pois as duas vias possuem fluxos de frequências parecidas.

Porém, para casos extremos, como duas vias onde uma possui um fluxo com frequência extremamente maior do que a outra, a política da *Maior fila sempre* pode se tornar prejudicial para a via com menor frequência.

4.2.4 Maior Fila Primeiro

A política definida no algoritmo da *maior fila primeiro* é semelhante ao da *maior fila primeiro*, exceto que este não concede apenas a prioridade da passagem pela interseção ao segmento de via com maior fila de veículos fora da região de conflito. Mas depois deixa passar a menor fila do outro segmento de via também, antes de voltar a comparar novamente as duas filas.

Considerando dois segmentos de vias s e w que se encontram em um ponto de interseção P, o algoritmo da *maior fila primeiro* inicia capturando todos os veículos fora da região de conflito em s e w, calculando o número dos veículos em cada segmento de via, e comparando os dois valores. Se s tiver mais veículos, o veículo no topo da fila capturada em w para na linha de retenção, levando os demais veículos da fila em w a parar também, até o último veículo da fila capturada em s passar pela interseção. Em seguida, deixa passar toda fila em w antes de comparar novamente as próximas filas nos dois segmentos. O mesmo processo é repetido até o fim da simulação.

Assim como a política da *Maior fila sempre*, a política da *Maior fila primeiro* foi escolhida também com a espectativa de observar como ela se comportará na distribuição da passagem dos veículos pela interseção de vias com fluxos de trânsito de frequências parecidas. Pois sua descrição é quase parecida que a da *Maior fila sempre*.

Porém, a diferença é que, para casos extremos, como duas vias onde uma possui um fluxo com frequência extremamente maior do que a outra, a política da *Maior fila primeiro* não se revela prejudicial para a via com menor frequência, pois, embora dê preferência à via com maior fila, ela deixa a via com menor fila passar logo em seguida.

4.2.5 No mínimo k por vez

O algoritmo de *No mínimo k por vez* consiste na passagem consecutiva dos veículos de cada via, desde que o número de veículos na via que possui a vez de passar seja superior ou igual a k, com $k < N$ sendo k um número inteiro informado e N o total de veículos.

Considerando dois segmentos de vias s e w que se encontram em um ponto de interseção

P, sendo s o segmento de via escolhido para iniciar a vez. O algoritmo de *No mínimo k por vez* inicia capturando todos os veículos fora da região de conflito em s, e calculando o número de veículos nessa fila. Se este for superior ou igual a k, captura-se a fila fora da região de conflito em w. O veículo no topo da fila capturada em w para na linha de retenção, levando os demais veículos da fila em w a parar também, até o último veículo da fila capturada em s passar pela interseção. Caso contrário, a vez passa para w. O mesmo processo é repetido até o fim da simulação.

A escolha da política de *No mínimo k por vez* foi com a espectativa de descobrir os resultados que uma política com filosofia semelhante que a do *Semáforo* geraria. Filosofia semelhante porque, enquanto a política do *Semáforo* efetua a passagem dos veículos pela interseção por intervalo de tempo, a política de *No mínimo k por vez* a efetua por intervalo de número de veículos.

5 IMPLEMENTAÇÃO

5.1 Ferramentas utilizadas

5.1.1 TraCI4J

TraCI4J (Traffic Control Interface for Java) é uma biblioteca implementada em *Java* para interagir e/ou assistir uma simulação do trânsito no SUMO através da interface TraCI. Maiores informações a respeito de *TraCI4J* se encontram em: https://github.com/egueli/TraCI4J

TraCI (Traffic Control Interface) é uma interface implementada em *Python*, que controla uma simulação do trânsito em execução no SUMO. *TraCI* permite a capturar os valores dos objetos da simulação, como também a manipular seus comportamentos em tempo real. Maiores informações a respeito de *TraCI* se encontram em: http://sumo-sim.org/userdoc/TraCI.html

TraCI usa uma arquitetura *TCP* do tipo *cliente - servidor* para proporcionar o acesso ao SUMO, considerando o SUMO como servidor para o qual realiza requisições a respeito dos objetos da simulação em execução, e recebe respostas do SUMO para manipulação desses objetos. Desta forma, *TraCI* pode controlar simulações executando em máquinas diferentes de onde está instalada.

TraCI4J pode atuar como *front-end* para uma instância do SUMO, podendo iniciar, parar, ou fazer a simulação andar $step$ por $step$. *TraCI4J* pode também extrair e alterar, de uma simulação em execução, informações tais como: a topologia de uma rede de vias, o número total de veículos em um segmento de via, a posição, velocidade, e tipos de veículo, o período do fluxo do trânsito em uma via, o tempo inicial e final da simulação, etc. Das classes de *TraCI4J* utilizadas para implementação dos algoritmos deste trabalho, algumas são:

- $SumoTraciConnection$: pode ser visto como a classe principal, pois é ela que permite a estabelecer a conexão com o SUMO, extrair, e manipular todos os objetos da simulação em execução. Nesta classe, alguns dos métodos que foram mais usados são: $runServer()$, que executa o SUMO depois de configurar as variáveis para a execução, $nextSimStep()$, que faz a simulação avançar um $step$, $getCurrentSimStep()$, que retorna o $step$ corrente, $getVehicleRepository()$, que retorna o repositório dos veículos

da simulação, $getEdgeRepository()$, que retorna o repositório dos segmentos de vias da simulação, $getRouteRepository()$, que retorna o repositório das vias da simulação, e $close()$, que fecha a instância do SUMO e encerra a conexão.

- $Vehicle$: classe que representa um veículo no SUMO, através da qual são extradas e alteradas as informações dos veículos da simulação. Dos métodos desta classe, os mais usados foram: $queryReadCurrentEdge()$, que retorna o segmento de via corrente, $queryReadPosition()$, que retorna a posição do veículo, e $queryChangeMaxSpeed()$ que alterar a velocidade máxima do veículo.

- $Edge$: Classe que representa um segmento de via em SUMO, através da qual se extrai e altera todas as informações dos segmentos de vias da simulação.

- $Route$:Classe que representa uma via em SUMO, através da qual se extrai e altera todas as informações das vias da simulação.

5.1.2 Classes e Métodos Implementados

Esta sessão explica de modo geral as classes e métodos implementados neste trabalho. Os detalhes da implementação dos mesmos, assim como todo o código fonte deste trabalho encontram-se anexados no apéndice.

TrafficSimulation

$TrafficSimulation$ é uma classe que extende a classe $SumoTraciConnection$ abordada na sessão 5.1.1. Ela foi implementada no âmbito de juntar métodos da super classe para criar novos métodos que forneçam resultados com relação à demanda deste trabalho, tendo um código refatorado e claro. Dos métodos criados na classe $TrafficSimulation$, temos:

```
1 public TrafficSimulation(String sumoCfgFile, String sumoGUIPath,
2 int randomSeed, boolean quitOnFinish)
```

Construtor da classe, estabelece a conexão com o SUMO utilizando o método construtor da superclasse, e executa o SUMO GUI. Ele recebe como parámetros: $sumoCfgFile$, o nome do arquivo *.net.cfg*, de configuração da simulação, visto no capítulo 3, $sumoGUIPath$, o caminho do arquivo executável do SUMO GUI, $randomSeed$, um número aleatório que, se diferente de -1, ele sobreescreve o atributo $value$ no arquivo $sumoCfgFile$, $quitOnFinish$, um booleano

que, se verdadeiro, fecha o SUMO GUI no fim da simulação.

```
1 public List<Vehicle> getAllVehicules()
```

Este método retorna todos os veículos correntes na simulação no $step$ no qual foi chamado. Ele utiliza o método $getVehicleRepository()$ da super classe para coletar o repositório dos veículos. Em seguida extrai desse repositório, a coleção dos veículos, e a transforma na lista de veículos a ser retornada.

```
1 public List<Edge> getAllEdges()
```

Este método retorna todos os segmentos de vias da simulação. Ele utiliza o método $getEdge\ Repository()$ da super classe para coletar o repositório dos segmentos de vias. Em seguida, extrai desse repositório, a coleção dos segmentos de vias, e a transforma na lista de segmentos de vias a ser retornada.

```
1 public List<Route> getAllRoutes()
```

Este método retorna todas as vias da simulação. Ele utiliza o método $getRouteRepository()$ da super classe para coletar o repositório das vias. Em seguida, extrai desse repositório, a coleção das vias, e a transforma na lista de vias a ser retornada.

```
1 public double distance(Vehicle v, Point2D point)
```

Dado um veículo v e um ponto P na via, este método retorna a distância entre v e P.

```
1 public List<Vehicle> getVsOutOfZone(Edge e)
```

Dada um segmento de via e, este método retorna todos os veículos em e fora da região de conflito.

```
1 public Vehicle getFirstVOutOfZone(List<Vehicle> vehicles)
```

Dada uma lista de veículos, este método retorna o primeiro veículo fora da região de conflito.

```
1 public boolean isVehicleAtEdge(Vehicle v, Edge e)
```

Dado um veículo v e um segmento de via e, este método verifica se v ainda está em e.

TrafficParser

$TrafficParser$ é a classe principal do trabalho. Nela estão implementadas os algoritmos descritos no capítulo 4 em forma de métodos, além de demais métodos auxiliares e constantes declaradas. $TrafficParser$ implementa também a $main$ que dispara a execução da aplicação. Maiores detalhes a respeito da implementação da classe encontram-se no Apêndice.

5.1.3 Gnuplot

Gnuplot é uma ferramenta que gera gráficos em 2D e 3D, disponível tanto como um *pompt* quanto em uma interface gráfica. *Gnuplot* funciona no *Windows*, *Linux*, *Mac OS*, assim como em outras plataformas, e seu código fonte é distribuído, porém, de direitos autorais reservados.

Gnuplot foi originalmente desenvolvido para permitir que cientístas visualizassem funções matemáticas e dados de forma interativa. Mas hoje, dá também suporte a outros usos.

Neste trabalho, o *Gnuplot* foi usado para visualizar os cenários gerados. Com a linha de comando $plot\ [0\ :\ 400]\ [-0.1\ :\ 1]\ "west.txt"\ with\ lines\ lc\ 1$, se visualiza os dados do arquivo *west.txt* em forma de linhas vermelhas onde as abcissas estão em um intervalo de 0 a 400, e as ordenadas, em um intervalo de -0,1 a 1, e com a linha de comando $replot\ "south.txt"\ with\ lines\ lc\ 3$, se visualiza os dados do arquivo *south.txt* em forma de linhas azuis com os mesmos intervalos que o primeiro comando, respectivamente.

6 AVALIAÇÃO EXPERIMENTAL

Este capítulo inicia descrevendo conceitos necessários para o entendimento da avaliação experimental. A seguir, são descritos detalhes da simulação, bem como os cenários usados na simulação. Finalmente, resultados obtidos com a aplicação de diferentes algoritmos em diferentes cenãrios são comentados.

6.1 Conceituação

Definição 7 Define-se um **estado do trânsito ou estado do fluxo do trânsito** E como o comportamento do fluxo do trânsito, descrito a partir do seu período (*period*), da preferência de uma via com relação a outra (*priority*), e da velocidade máxima que possui cada via (*maxSpeed*).

Definição 8 Um **cenário** C é o resultado da aplicação de um dos algoritmos descritos nos capítulos 4 e 5, a um dos estados (fluxo) do trânsito da tabela 6.1. Um cenário é representado em forma de gráfico bidimencional cujo eixo das abscissas representa os *steps*, e o das ordenadas representa a *taxa de variação* dos veículos nos segmentos das vias, definida em seguida. No que diz respeito a cenários as seguintes premissas devem ser obedecidas:

- Dois ou mais cenários pertencem a um mesmo *Grupo* G de cenários se e somente se estes derivam de um mesmo estado de fluxo do trânsito E.
- Dois cenários C_1 e C_2 são iguais se e somente se estes derivam de um mesmo algoritmo A, aplicado a um mesmo estado de fluxo do trânsito E.

Definição 9 Considerando um cenário C, com um ponto de interseção P, de dois segmentos de via s e w. A taxa de variação dos veículos de s no intervalo de $k\ step$, representada por T_s, é a proporção da passagem dos veículos de s por P no intervalo de $k\ step$. Ela é definida pela seguinte fórmula:

$$T_s = \frac{\sum v_p}{\sum v}$$

onde:

- $\sum v_p \in [0, N]$ e $\sum v \in (0, N]$, onde N_v é o número total de veículos que podem transitar pelo segmento de vias v durante a simulação;

- $k \in (0, N)$, onde N é o núumero total de $steps$ da simulação;
- v_p: representa o veículo de s que passou pelo ponto de interseção P.
- v: representa o veículo em s (ainda não passou por P).

Se T_s > T_w no $step\ i$, então houve mais veículos de s que passaram por P no $step\ i$, do que os veículos de w.

Definição 10 Considerando um cenário C, com um ponto de interseção P, de dois segmentos de via s e w. A taxa total de variação dos veículos de s durante toda a simulação, representada por $\overline{X}_s$, é o somatório das taxas de variações em s. A taxa total de variação é definida pela seguinte fórmula:

$$\overline{X}_s = \sum T_s$$

- Se $\overline{X}_s$ > $\overline{X}_w$, então houve mais veículos de s que passaram por P do que veículos de w.

Definição 11 Considerando um cenário C com dois segmentos de vias s e w que se encontram em um ponto de interseção P. A diferença das taxas totais de variação de s e w em C, representada por $d\overline{X}$, é o módulo da diferença entre $\overline{X}_s$ e $\overline{X}_w$. Ela é expressa pela seguinte fórmula:

$$d\overline{X} = |\overline{X}_s - \overline{X}_w|$$

Definição 12 A distribuição do fluxo do trânsito entre dois segmentos de vias s e w que se cruzam em um ponto de interseção P, em um cenário C, diz respeito à comparação das proporções de passagens de veículos de s e w por P. Considerando dois cenários C_i e C_j de um grupo G, e sejam $d\overline{X}_i$ e $d\overline{X}_j$, as respectivas diferenças de taxas totais em C_i e C_j:

- Se $d\overline{X}_i$ > $d\overline{X}_j$, diz-se que a distribuição do fluxo em C_i é melhor do que em C_j. Em outras palavras, C_ié mais distribuído do que C_j.
- Se $d\overline{X}_i$ tender 0, diz-se que a distribuição do fluxo em C_i tende a ser equitável, ou seja, C_i tende a ser equitavelmente distribuído com relação ao fluxo.
- Um algoritmo A_i é considerado mais eficaz do que um outro A_j com base em um estado de fluxo do trânsito E_k, se e somente se o cenário C_i gerado por A_i aplicado a E_k for mais distribuído do que C_j gerado por A_j aplicado a E_k.

6.2 Simulação

Após finalizar a contextualizado dos conceitos-base que definem a metodologia de comparação a ser utilizada nesta avaliação experimental, é descrita a simulação construída para geração dos estados de fluxo do trânsito aos quais são aplicados os algoritmos definidos nos capítulos 4 e 5.

Foi contruída uma rede viária composta por nove nodos, oitos segmentos de vias, sendo que quatro segmentos são iniciais e quatros são principais às duas vias, e duas vias, como mostra a figura 3.1. A simulação usa, no total, nove mil veículos transitando em cada via.

Nodos

Conforme ilustrado na Figura 3.2, o nodo 0 encontra-se na posição central, em coordenadas (0, 0) (ver $l1$), com relação aos demais nodos. O grupo dos nodos (1, 2, 3, 4) localizam-se respectivamente na posição sul-oeste, norte-este, sul-este, norte-oeste (ver $l2 - l5$). E o grupo dos nodos (m1, m2, m3, m4) localizam-se respectivamente nas posições oeste, este, sul, norte (ver $l6 - l9$).Todos os nodos são do tipo *priority*.

Segmentos de Vias

Como mostrado na Figura 3.3, dos segmentos de vias criados, os segmentos $w0$, $e0$, $s0$, $n0$ representam respectivamente os segmentos iniciais oeste, leste, sul, e norte (ver $l2$, $l5$, $l8$ e $l11$). E os segmentos w, e, s, n representam respectivamente os segmentos principais oeste, leste, sul, e norte (ver $l3$, $l6$, $l9$ e $l12$).

Vias

A Figura 3.5, entre as linhas $l12$ e 17, mostra a configuração das vias utilizadas na simulação. observa-se que a via WE é a que vai do oeste para o leste, construida pelos segmentos de vias $w0$, w, e, $e0$ (ver $l2$). A via SN é aquela que vai do sul para o norte, contruída pelos segmentos $s0$, s, n, $n0$ (ver $l5$).

Veículos

A Figura 3.5, entre a linha $l4$ e a linha 10, mostra a configuração dos veículos utilizados na simulação. Pode-se observar que os veículos são classificados por tipos, sendo estes: $passenger$ (ver $l2$), bus (ver $l4$), e $transport/trailer$ (ver $l6$).

6.3 Geração dos cenários

Configuração dos estados de fluxos do trânsito

Foram configurados três estados de fluxos do trânsito sumarizados na Tabela 6.1. Cada estado de fluxo do trânsito ocorre em um intervalo de 14400 $steps$. Esse intervalo se revelou satisfatório para coletar os dados expressando suficientemente a influência dos algoritmos no fluxo do trânsito das vias.

Estados	Priority	Period	MaxSpeed
E_1	SN > WE (3, 2)	SN = WE (1 step, 1 step)	SN = WE (16.7 m/s, 16.7 m/s)
E_2	SN > WE (3, 2)	SN < WE (1 step, 10 step)	SN = WE (16.7 m/s, 16.7 m/s)
E_3	SN > WE (3, 2)	SN > WE (10 step, 1 step)	SN = WE (16.7 m/s, 16.7 m/s)

Tabela 6.1 – Estados do trânsito simulados

A coluna *Estados* na Tabela 6.1 descreve os nomes de cada estado do trânsito. Enquanto constam, na coluna *Priority*, a comparação das prioridades ($priority$) das vias SN e WE, e na coluna *MaxSpeed*, constam a comparação das velocidades máximas ($maxSpeed$) das duas vias.

Como visualizado na Tabela 6.1, o estado de fluxo do trânsito E_1 é definido como o estado onde a via *Sul-Norte* (SN) tem prioridade na passagem dos veículos pela interseção com relação à via *Oest-Leste* (WE). O fluxo do trânsito é denso nas duas vias, ou seja, os veículos acessam as duas vias com a mesma frequência, e são submetidos às mesmas velocidades máximas a não ultrapassar.

O estado de fluxo do trânsito E_2 é definido como o estado onde a via *Sul-Norte* (SN) tem também prioridade na passagem dos veículos pela interseção com relação à via *Oest-Leste* (WE). Porém, o fluxo do trânsito é denso somente na via *Sul-Norte*, ou seja, os veículos acessam a via *Sul-Norte* com uma frequência dez vezes superior à via *Oeste-Leste*. Mas os veículos são submetidos às mesmas velocidades máximas a não ultrapassar nas duas vias.

O estado de fluxo do trânsito E_3 possui um fluxo contrário ao E_2. A via *Sul-Norte* (SN) continua tendo prioridade na passagem dos veículos pela interseção com relação à via *Oest-*

Leste (WE), e a velocidade máxima a alcançar continua sendo a mesma nas duas vias. NO entanto, o fluxo do trânsito é denso somente na via *Oeste-Leste*, ou seja, os veículos acessam a via *Oeste-Leste* com uma frequência dez vezes superior à via *Sul-Norte*.

Algoritmos

Os algoritmos descritos nos capítulos 4 e 5 são sumarizados na Tabela 6.2, renomeados por letras de $A0$ a $A5$.

Algoritmos	Nomes
A_0	Algoritmo default
A_1	Semáforo
A_2	Maior fila sempre
A_3	Maior fila primeiro
A_4	No mínimo k por vez

Tabela 6.2 – Algoritmos a serem comparados

Na Tabela 6.2, a coluna *Algoritmos* possui as siglas dos algoritmos descritos nos capítulos 4 e 5. Enquanto a coluna *Nomes* possui os nomes de cada algoritmos.

Grupos de Cenários

Os cenários gerados são classificados em três grupos de cenários, beseados nos estados de fluxos do trânsito nos quais foram aplicados os algoritmos. Todos eles ocorrem em um intervalo de 14400 $steps$, que é aproximadamente equivalente a 4 horas.

O primeiro grupo de cenários G_1 é composto dos cenários gerados pela aplicação dos algoritmos A_1 a A_4 ao estado do fluxo de trânsito E_1, e está sumarizado na Tabela 6.3.

Cenários	Estados	Algoritmos
$C_{1.0}$	E_1	A_0
$C_{1.1}$	E_1	A_1
$C_{1.2}$	E_1	A_2
$C_{1.3}$	E_1	A_3
$C_{1.4}$	E_1	A_4

Tabela 6.3 – Cenários gerados pela aplicação dos algoritmos A_1 a A_5 ao estado E_1

O segundo grupo de cenários G_2 é composto dos cenários gerados pela aplicação dos algoritmos A_1 a A_4 ao estado do fluxo de trânsito E_2, e está sumarizado na Tabela 6.4.

O terceiro grupo de cenários G_3 é composto dos cenários gerados pela aplicação dos algoritmos A_1 a A_4 ao estado do fluxo de trânsito E_3, e está sumarizado na Tabela 6.5.

Cenários	Estados	Algoritmos
$C_{2.0}$	E_2	A_0
$C_{2.1}$	E_2	A_1
$C_{2.2}$	E_2	A_2
$C_{2.3}$	E_2	A_3
$C_{2.4}$	E_2	A_4

Tabela 6.4 – Cenários gerados pela aplicação dos algoritmos A_1 a A_5 ao estado E_2

Cenários	Estados	Algoritmos
$C_{3.0}$	E_3	A_0
$C_{3.1}$	E_3	A_1
$C_{3.2}$	E_3	A_2
$C_{3.3}$	E_3	A_3
$C_{3.4}$	E_3	A_4

Tabela 6.5 – Cenários gerados pela aplicação dos algoritmos A_1 a A_5 ao estado E_3

Nas Tabelas 6.3, 6.4, e 6.5, a coluna *Cenários* possui as siglas dos Cenários gerados. Enquanto a coluna *Estados* possui as siglas dos estados dos fluxos do trânsito configurados, e a coluna *Algoritmos* possui as siglas dos algoritmos aplicados aos estados dos fluxos do trânsito.

Gráficos obtidos através dos experimentos

Considerando um cenário C com dois segmentos de vias s e w que se encontram em um ponto de interseção P. E seja T_s, a taxa de variação dos veículos em s no $step\ i$ e T_w, a taxa de variação dos veículos em w no $step\ j$, tem-se que:

- C é visualizado através de um gráfico que possui dois tipos de linhas: as linhas vermelhas que representam as taxas de variações T_w, por unidade de $step$, e as linhas azuis que representam as taxas de variações T_s por unidade de $step$.
- T_s indica a porcentagem dos veículos de s que passaram por P no $step\ i$ e T_w indica a porcentagem dos veículos de w que passaram por P no $step\ i$.
- Quando T_s é 0, isso indica que não há nenhum veículo de s que passou por P no $step\ i$ e quando T_w é 0, isso indica que não há nenhum veículo de w que passou por P no $step\ j$.
- Quando T_s é máxima, isso indica que todos os veículos de s passaram por P no $step\ i$ e quando T_w é máxima, isso indica que todos os veículos de w passaram por P no $step\ j$.

Cenário $C_{1.0}$

O Cenário $C_{1.0}$ resulta da aplicação do algoritmo A_0 ao estado do fluxo do trânsito E_1, e é visualizado na Figura 6.1

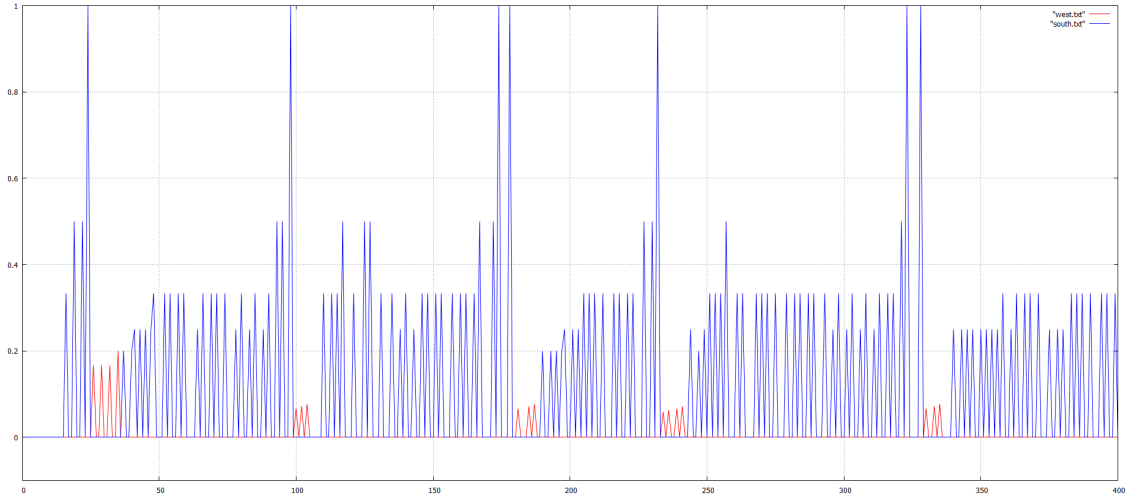

Figura 6.1 – Cenário $C_{1.0}$

Na Figura 6.1, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais a 0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em alguns $steps$ específicos observa-se que T_s é máxima. Portanto, todos os carros de s passaram por P naquele intervalo de $steps$.

Cenário $C_{1.1}$

O Cenário $C_{1.1}$ resulta da aplicação do algoritmo A_1 ao estado do fluxo do trânsito E_1, e é visualizado na Figura 6.2

Na Figura 6.2, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais a 0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em nenhum $step$ T_s ou T_w são máximas. Portanto, em nenhum momento houve todos os carros de s ou w passando por P.

Cenário $C_{1.2}$

O Cenário $C_{1.2}$ resulta da aplicação do algoritmo A_2 ao estado do fluxo do trânsito E_1, e é visualizado na Figura 6.3

Na Figura 6.3, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais a 0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em nenhum $step$ T_s ou T_w

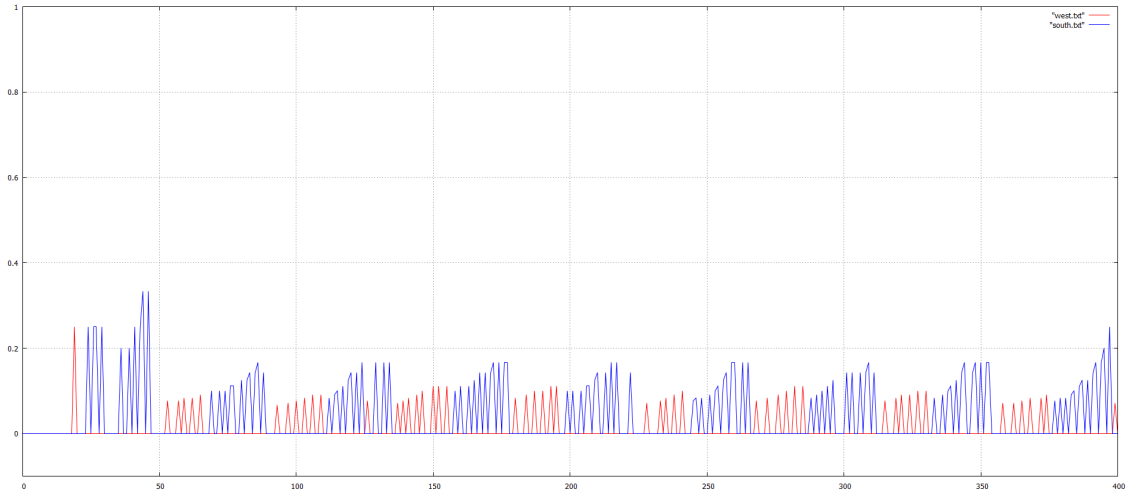

Figura 6.2 – Cenário $C_{1.1}$

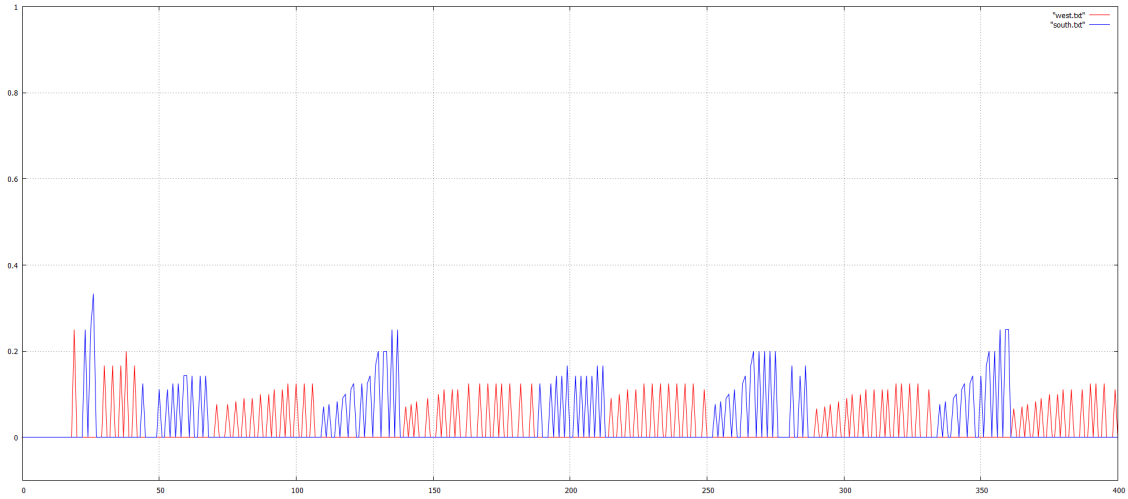

Figura 6.3 – Cenário $C_{1.2}$

são máximas. Portanto, em nenhum momento houve todos os carros de s ou w passando por P.

Cenário $C_{1.3}$

O Cenário $C_{1.3}$ resulta da aplicação do algoritmo A_3 ao estado do fluxo do trânsito E_1, e é visualizado na Figura 6.4

Na Figura 6.4, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais a 0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em nenhum $step$ T_s ou T_w são máximas. Portanto, em nenhum momento houve todos os carros de s ou w passando por P.

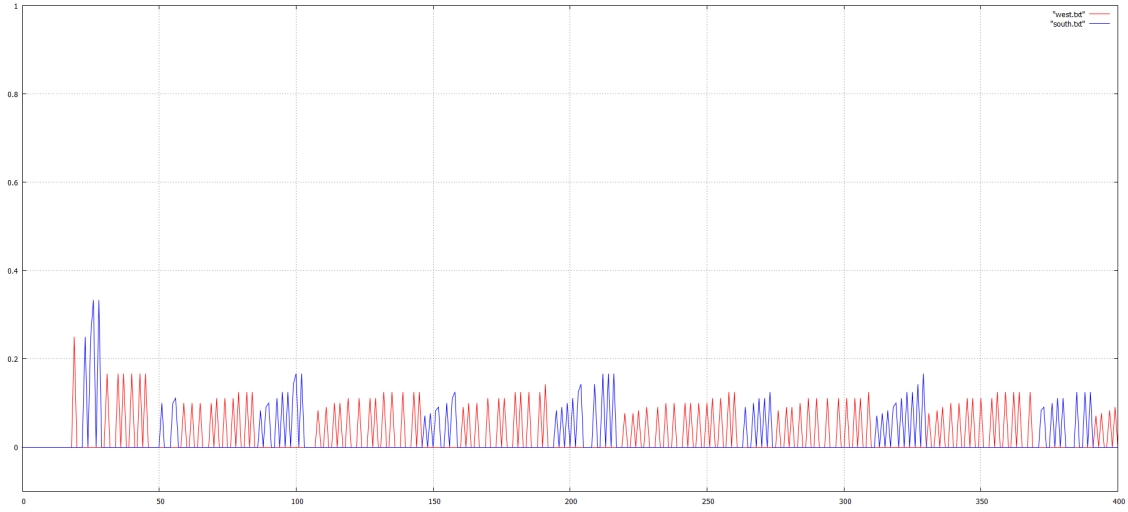

Figura 6.4 – Cenário $C_{1.3}$

Cenário $C_{1.4}$

O Cenário $C_{1.4}$ resulta da aplicação do algoritmo A_4 ao estado do fluxo do trânsito E_1, e é visualizado na Figura 6.5

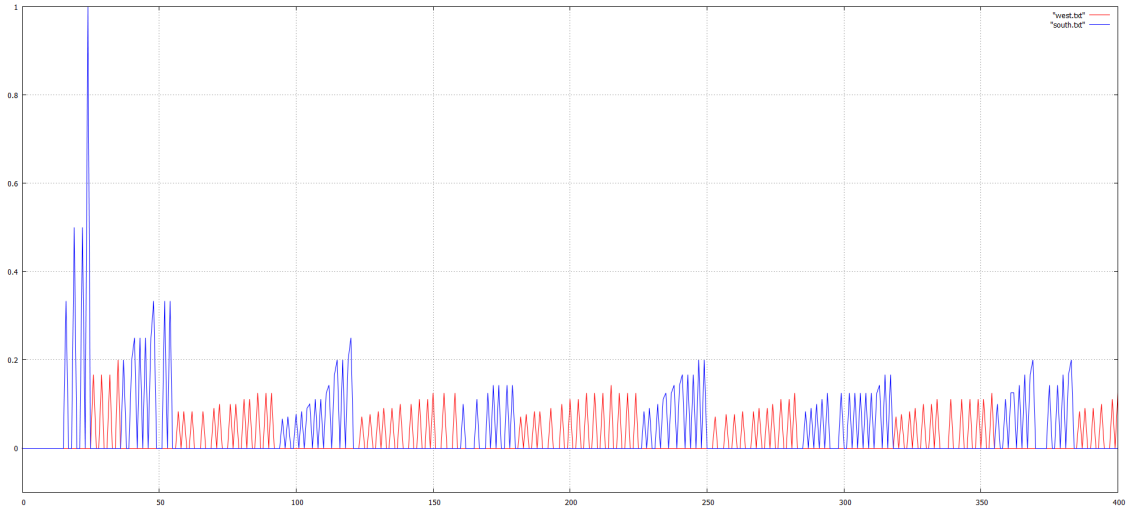

Figura 6.5 – Cenário $C_{1.4}$

Na Figura 6.5, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais a 0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em nenhum $step$ T_s ou T_w são máximas. Portanto, em nenhum momento houve todos os carros de s ou w passando por P.

Cenário $C_{2.0}$

O Cenário $C_{2.0}$ resulta da aplicação do algoritmo A_0 ao estado do fluxo do trânsito E_2,

e é visualizado na Figura 6.6

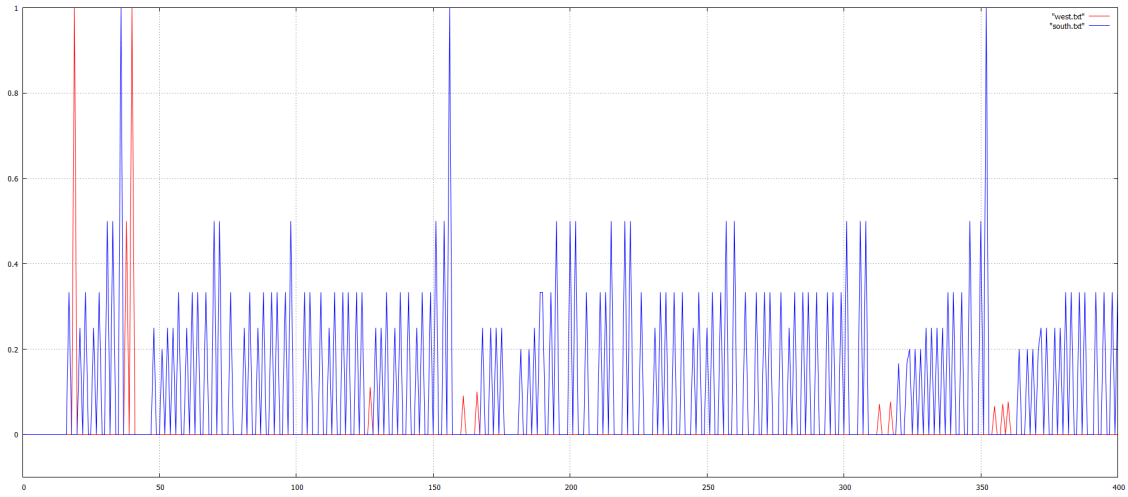

Figura 6.6 – Cenário $C_{2.0}$

Na Figura 6.6, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais a 0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em alguns $step$ específicos, T_s e T_w são máximas. Portanto, todos os carros em s e w passaram por P naqueles $steps$.

Cenário $C_{2.1}$

O Cenário $C_{2.1}$ resulta da aplicação do algoritmo A_1 ao estado do fluxo do trânsito E_2, e é visualizado na Figura 6.7

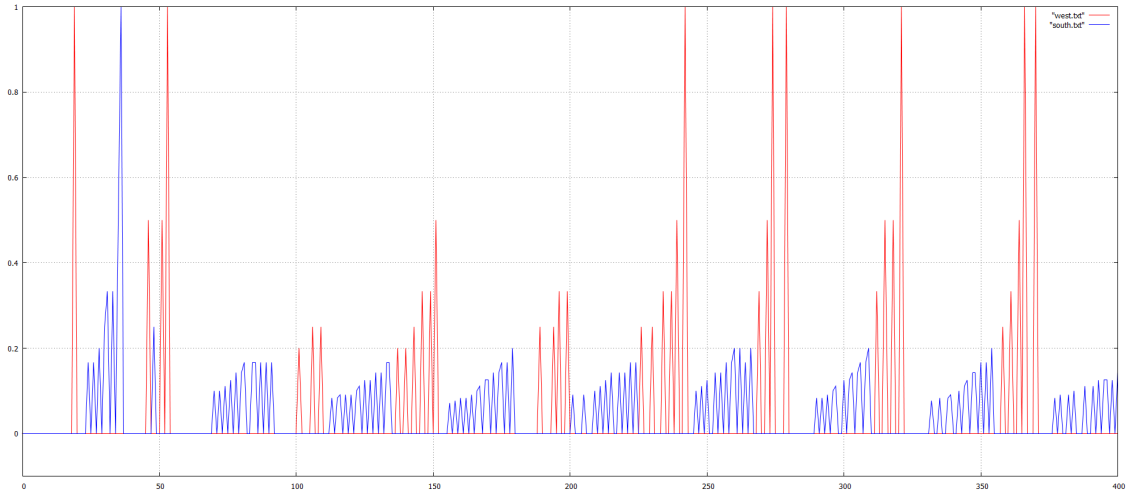

Figura 6.7 – Cenário $C_{2.1}$

Na Figura 6.7, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais a 0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em alguns $step$ específicos,

T_s e T_w são máximas. Portanto, todos os carros em s e w passaram por P naqueles $steps$.

Cenário $C_{2.2}$

O Cenário $C_{2.2}$ resulta da aplicação do algoritmo A_2 ao estado do fluxo do trânsito E_2, e é visualizado na Figura 6.8

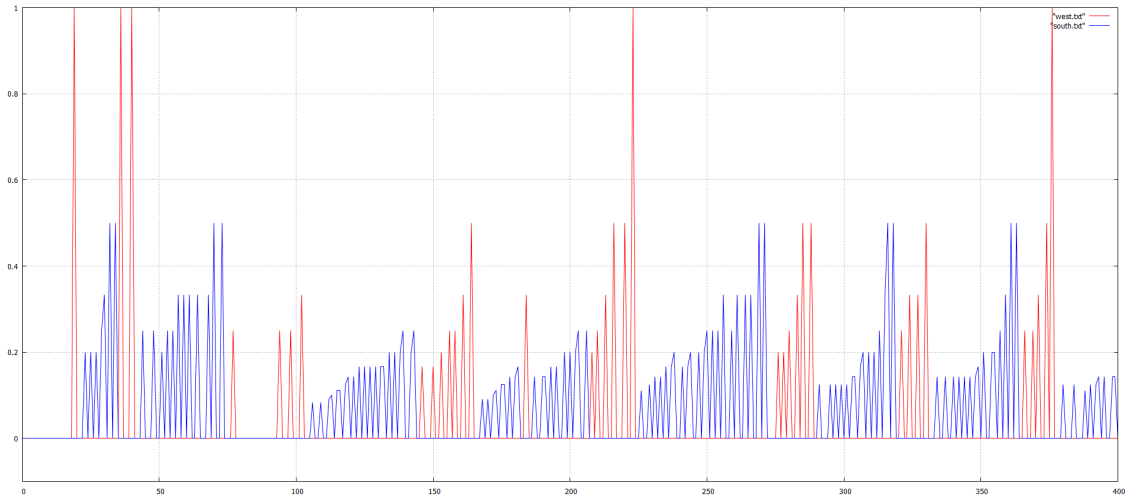

Figura 6.8 – Cenário $C_{2.2}$

Na Figura 6.8, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais a 0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em alguns $step$ específicos, T_w é máxima. Portanto, todos os carros em w passaram por P naqueles $steps$.

Cenário $C_{2.3}$

O Cenário $C_{2.3}$ resulta da aplicação do algoritmo A_3 ao estado do fluxo do trânsito E_2, e é visualizado na Figura 6.9

Na Figura 6.9, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais a 0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em alguns $step$ específicos, T_w é máxima. Portanto, todos os carros em w passaram por P naqueles $steps$.

Cenário $C_{2.4}$

O Cenário $C_{2.4}$ resulta da aplicação do algoritmo A_4 ao estado do fluxo do trânsito E_2, e é visualizado na Figura 6.10

Na Figura 6.10, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais a

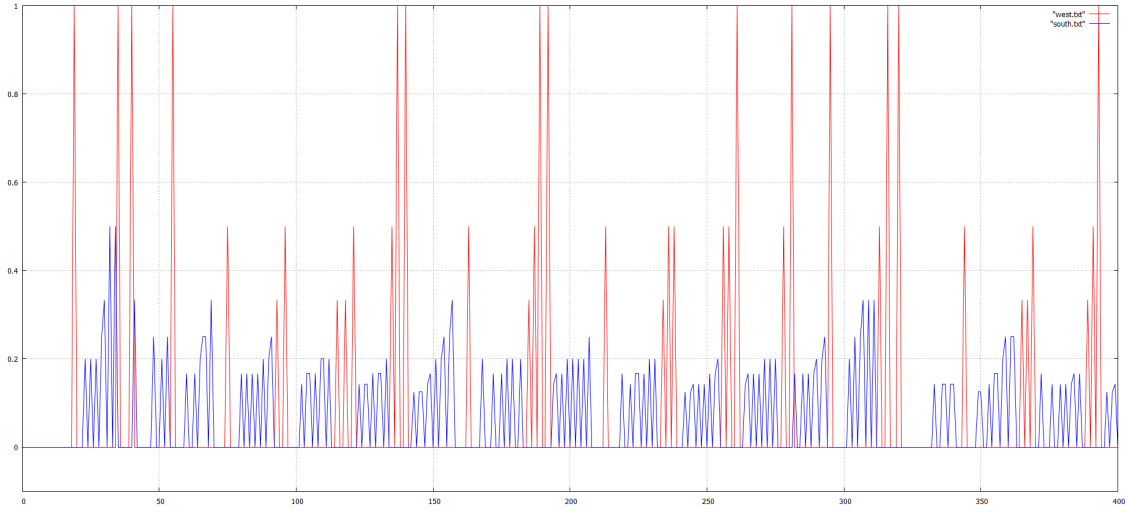

Figura 6.9 – Cenário $C_{2.3}$

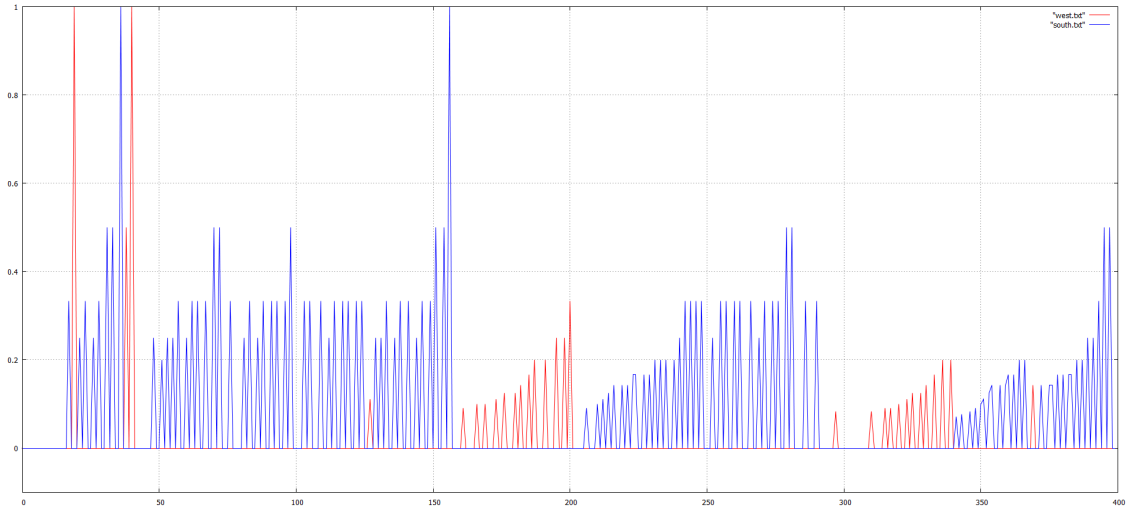

Figura 6.10 – Cenário $C_{2.4}$

0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em alguns $step$ específicos, T_w é máxima. Portanto, todos os carros em w passaram por P naqueles $steps$.

Cenário $C_{3.0}$

O Cenário $C_{3.0}$ resulta da aplicação do algoritmo A_0 ao estado do fluxo do trânsito E_3, e é visualizado na Figura 6.11

Na Figura 6.11, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais a 0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em alguns $steps$ específicos observa-se que T_s é máxima. Portanto, todos os carros de s passaram por P naquele intervalo.

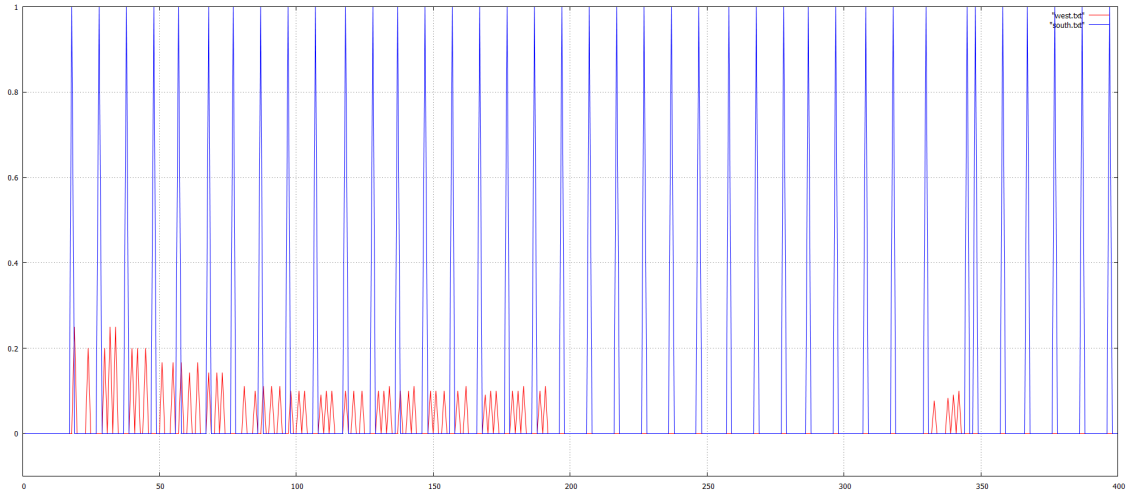

Figura 6.11 – Cenário $C_{3.0}$

Cenário $C_{3.1}$

O Cenário $C_{3.1}$ resulta da aplicação do algoritmo A_1 ao estado do fluxo do trânsito E_3, e é visualizado na Figura 6.12

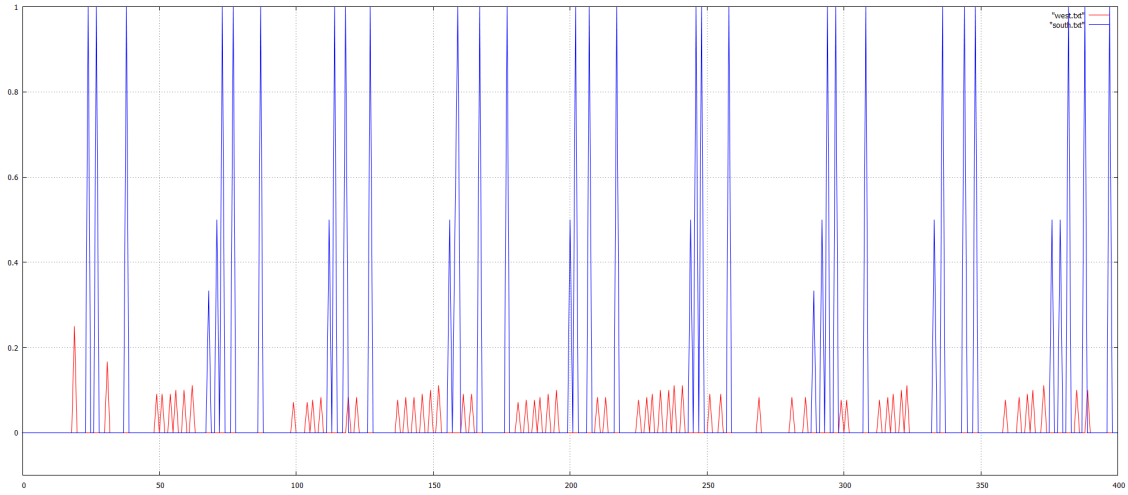

Figura 6.12 – Cenário $C_{3.1}$

Na Figura 6.12, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais a 0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em alguns $steps$ específicos observa-se que T_s é máxima. Portanto, todos os carros de s passaram por P naquele intervalo.

Cenário $C_{3.2}$

O Cenário $C_{3.2}$ resulta da aplicação do algoritmo A_2 ao estado do fluxo do trânsito E_3, e é visualizado na Figura 6.13

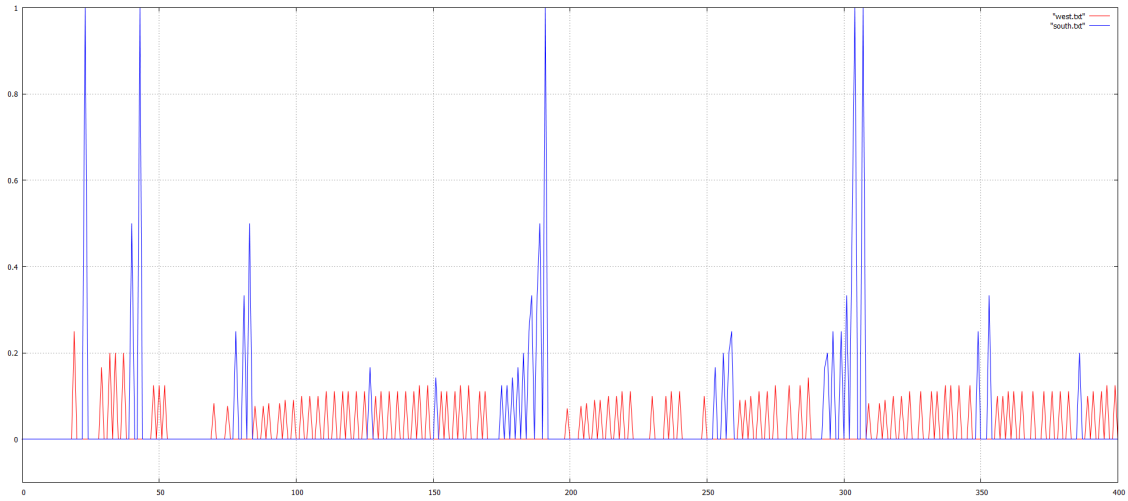

Figura 6.13 – Cenário $C_{3.2}$

Na Figura 6.13, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais a 0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em alguns $steps$ específicos observa-se que T_s é máxima. Portanto, todos os carros de s passaram por P naquele intervalo.

Cenário $C_{3.3}$

O Cenário $C_{3.3}$ resulta da aplicação do algoritmo A_3 ao estado do fluxo do trânsito E_3, e é visualizado na Figura 6.14

Na Figura 6.14, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais a 0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em alguns $steps$ específicos observa-se que T_s é máxima. Portanto, todos os carros de s passaram por P naquele intervalo.

Cenário $C_{3.4}$

O Cenário $C_{3.4}$ resulta da aplicação do algoritmo A_4 ao estado do fluxo do trânsito E_3, e é visualizado na Figura 6.15

Na Figura 6.15, observa-se entre os $steps$ 0 e 20, por exemplo, que T_s e T_w são iguais

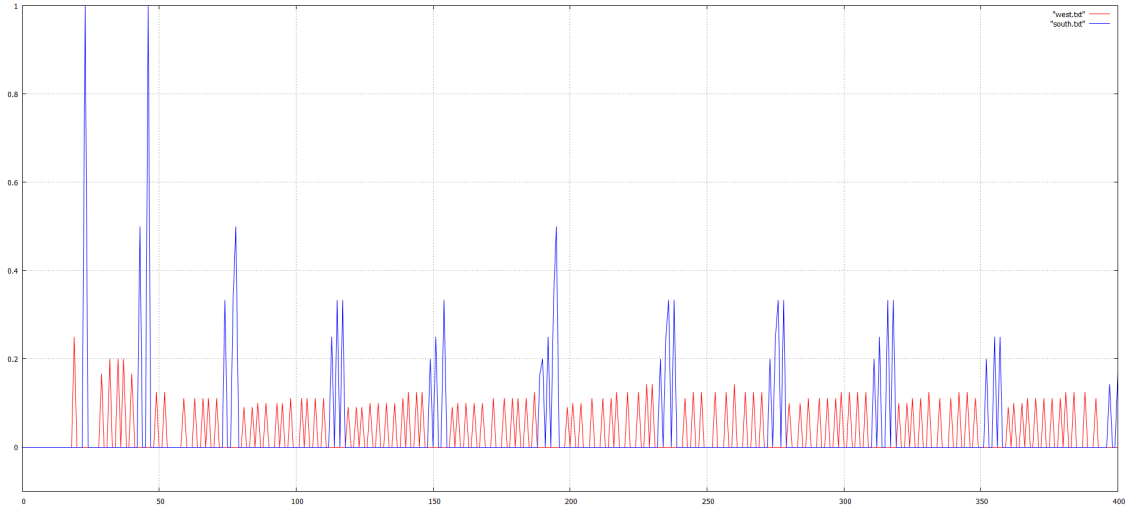

Figura 6.14 – Cenário $C_{3.3}$

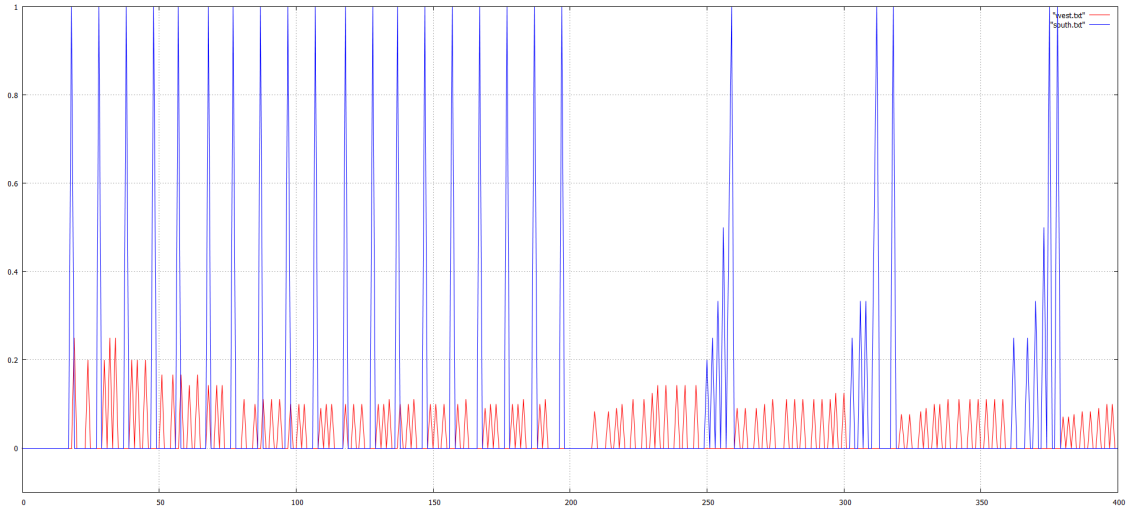

Figura 6.15 – Cenário $C_{3.4}$

a 0. Logo, nenhum veículo de s ou w passou por P naquele intervalo. Em alguns $steps$ específicos observa-se que T_s é máxima. Portanto, todos os carros de s passaram por P naquele intervalo.

6.4 Comparação dos cenários

A comparação é realizada entre os cenários que fazem parte de um mesmo grupo. O objetivo da comparação dos cenários é descobrir qual é o cenário mais distribuído de cada grupo, de modo a descobrir qual o algoritmo que distribui melhor a passagem dos veículos pela interseção para cada grupo de cenários.

A varáavel alvo de comparação é a *diferença das taxas totais de variações* $d\overline{X}$ de cada cenário. Todavia foram colocadas também nas Tabelas de cada grupo, as taxas totais, X_s e X_w, por raízes ilustrativas apenas, ou seja, para mostrar como a distribuição das passagens de veículos foi realizada em cada cenário.

Grupo de cenários G_1

A Tabela 6.6 sumariza os resultados obtidos com experimentos usando os cenários no G_1 dentro do intervalo de 14400 $steps$.

Cenários	X_s	X_w	$d\overline{X}$
$C_{1.0}$	1626,13	41,62	1584,51
$C_{1.1}$	463,71	196,59	267,12
$C_{1.2}$	400,18	301,56	98,62
$C_{1.3}$	277,21	334,54	57,32
$C_{1.4}$	395,33	282,58	112,75

Tabela 6.6 – Resultados dos cenários do primeiro grupo

Pelos resultados obtidos e visualizados na Tabela 6.6, pode-se observar que o cenário $C_{1.3}$ foi o mais distribuído de todos com uma diferença de taxas totais $d\overline{X}$ de 57,32, lembrando que quanto menor $d\overline{X}$, mais distribuído é o cenário. Em segundo lugar, temos $C_{1.2}$ com $d\overline{X}$ = 98,62, depois $C_{1.4}$ com $d\overline{X}$ = 112,75, depois $C_{1.2}$ com $d\overline{X}$ = 267,12, e por último, temos $C_{1.0}$ com 1584,51.

Deste modo, podemos concluir que, das políticas implementadas, a da *Maior Fila Primeiro* se revelou a mais adequada para controlar a passagem dos veículos pela interseção entre duas vias com fluxo de mesma frequência e velocidade máxima, cuja prioridade de uma é superior à da outra, assim como definido pelo estado do fluxo do trânsito E_1. Depois temos a política da *Maior Fila Sempre*, semelhante a da *Maior Fila Primeira*, que se mostrou a segunda mais adequada para o controle desse tipo de trânsito, e a de *No Mínimo* k *Veículos por Vez* vem em terceiro lugar, a do *Semáforo* em quarto lugar, por fim, a da *Prioridade à direita* que se mostrou a menos adequada para esse tipo de trânsito.

Grupo de cenários G_2

A Tabela 6.7 sumariza os resultados obtidos com experimentos usando os cenários no G_2 dentro do intervalo de 14400 $steps$.

Pelos resultados obtidos e visualizados na Tabela 6.7, pode-se observar que o cenário

Cenários	X_s	X_w	$d\overline{X}$
$C_{2.0}$	1619,45	43,75	1575,70
$C_{2.1}$	456,85	589,28	132,43
$C_{2.2}$	1156,05	300,90	855,15
$C_{2.3}$	1057,24	428,57	628,67
$C_{2.4}$	1294,14	127,17	1166,97

Tabela 6.7 – Resultados dos cenários do segundo grupo

$C_{2.1}$ foi o mais distribuído de todos com uma diferença de taxas totais $d\overline{X}$ de 132,43, Em segundo lugar, temos $C_{2.3}$ com $d\overline{X}$ = 628,67, depois $C_{2.2}$ com $d\overline{X}$ = 855,15, depois $C_{2.4}$ com $d\overline{X}$ = 1166,97, e por último, temos $C_{2.0}$ com 1575,70.

Deste modo, podemos concluir que, das políticas implementadas, a do *Semáforo* se revelou a mais adequada para controlar a passagem dos veículos pela interseção entre duas vias com o fluxo de trânsito tal como definido pelo estado do fluxo do trânsito E_2. Depois temos a política da *Maior Fila Primeiro* que, desta vez, se mostrou a segunda mais adequada para o controle desse tipo de trânsito, e a de *Maior Fila Sempre* está na terceira posição, a de *No Mínimo k Veículos por Vez* em quarta posição, por fim, a da *Prioridade à direita* que, mais uma vez, se mostrou a menos adequada para esse tipo de trânsito.

Grupo de cenários G_3

A Tabela 6.8 sumariza os resultados obtidos com experimentos usando os cenários no G_3 dentro do intervalo de 14400 $steps$.

Cenários	X_s	X_w	$d\overline{X}$
$C_{3.0}$	899,00	385,64	513,36
$C_{3.1}$	751,75	233,57	518,18
$C_{3.2}$	282,58	547,90	265,32
$C_{3.3}$	227,80	545,51	317,71
$C_{3.4}$	505,64	515,40	9,76

Tabela 6.8 – Resultados dos cenários do terceiro grupo

Pelos resultados obtidos e visualizados na Tabela 6.8, pode-se observar que o cenário $C_{3.4}$ foi o mais distribuído de todos com uma diferença de taxas totais $d\overline{X}$ de 9,76, alcançando quase a distribuição equitável entre as duas vias, lembrando que a distribuição da passagem dos veículos pela interseção entre duas vias em um dado cenário tende à equitabilidade quanto mais a diferença das taxas totais $d\overline{X}$ tende a Zero. Em segunda colocação, temos $C_{3.2}$ com $d\overline{X}$ = 265,32, depois $C_{3.3}$ com $d\overline{X}$ = 317,71, depois $C_{2.0}$ com $d\overline{X}$ = 513,36, e por último, temos $C_{2.1}$ com 518,18.

Deste modo, podemos concluir que, das políticas implementadas, a de *No Mínimo k Veículos por Vez* se revelou claramente a mais adequada para controlar a passagem dos veículos pela interseção entre duas vias com o fluxo de trânsito tal como definido pelo estado do fluxo do trânsito E_3. Depois temos a política da *Maior Fila Sempre* que se mostrou a segunda mais adequada para o controle desse tipo de trânsito, e a de *Maior Fila Primeiro* está na terceira posição, a da *Prioridade à direita* em quarta posição, por fim, a do *Semáfor* que, desta vez, se mostrou a menos adequada para esse tipo de trânsito.

7 CONCLUSÕES

Este trabalho apresentou uma simulação no contexto de redes de transporte. Foram avaliadas diferentes políticas para o gerenciamento de interseções. Foi configurado uma simulação com duas vias: uma do sul ao norte (SN), e outra do oeste ao leste (WE), as duas se encontrando em um ponto de interseção P. Em seguida foram criados três estados de fluxos do trânsitos: E_1, E_2, e E_3, baseados na preferência da SN sobre a WE na passagem dos veículos por P, no período do acesso dos veículos nas duas vias, e na velocidade máxima das duas vias. Depois foram implementadas cinco políticas para controlar a passagem dos veículos de SN e WE por P. Por fim, foram gerados 15 cenários decorrentes da aplicação das políticas implementadas aos estados de fluxos do trânsito definidos, classificados em três grupos de cenários de acordo com cada estado do trânsito.

Experimentos com estes três cenários foram executados. Após avaliação dos cenários, os resultados mostraram que quando o trânsito é do tipo definido pelo estado E_1 (a frequência do fluxo da via preferencial SN é igual à da WE), a política implementada pelo algoritmo da *Maior Fila Primeiro* é a mais adequada para controlar a passagem dos veículos por P. Enquanto, quando o trânsito é do tipo definido pelo estado E_2 (a frequência do fluxo da via preferencial SN é superior à da WE), a política implementada pelo algoritmo do *Semáforo* é a mais adequada para o controle das passagens por P. Por fim, quando o trânsito do tipo definido pelo estado E_3 (a frequência do fluxo da via preferencial SN é inferior à da WE), a política implementada pelo algoritmo *No Mínimo k Veículos por Vez* é a mais adequada para efetuar o controle das passagens por P.

Com esses resultados, pode-se afirmar que aplicar apenas uma política para controlar o trânsito urbano não é a solução mais adequada, pois o trânsito sendo variado, como foi ilustrado pelos estados definidos no capítulo 6, além de outras realidades que não foram abordadas neste trabalho, cada estado gerado pelo trânsito precisa de uma política adaptável ao seu contexto. Um caso a destacar é o da política do semáforo tradicional que hoje ainda permanece a mais usada no trânsito brasileiro nas interseções sinalizadas. Ela foi a mais adequada apenas no estado E_2, no qual a frequência do fluxo da via preferencial é 10 vezes superior à da outra, realidade que é menos frequente no dia-a-dia, comparado ao estado E_1 que é a mais frequente dos três estados, e para o qual a política do *semáforo* foi a penúltima mais adequada para controlar a passagem dos veículos por P.

Como trabalhos futuros, poderá ser proposta uma política dinâmica adaptável ao estado corrente do trânsito. Em outras palavras, poderá ser desenvolvida uma política híbrida, que é a junção de várias políticas e que analisa o trânsito no âmbito de investigar os diferentes tipos de estados gerados, de modo, a se adaptar a cada estado. Além do mais, pode-se definir outros estados do trânsito que descrevem com mais ênfase o realismo na simulação, levando em conta outras variáveis que influenciam o fluxo do trânsito, entre outras: a variação da velocidade máxima das vias, a prioridade dos tipos de veículos, ambulâncias,. . . , nas passagens pelas interseções, pedestres, acidentes e outros incidentes que obstruem e mudam o fluxo do trânsito, etc. Por fim, pode-se ampliar a rede viária do trânsito acrescentando faixas nas vias e adicionando várias interseções.

REFERÊNCIAS

DEZANI H.; GOMES, L. D. F. M. N. Controlling traffic jams on urban roads modeled in Coloured Petri net using Genetic Algorithm. In: IECON 2012 - 38TH ANNUAL CONFERENCE ON IEEE INDUSTRIAL ELECTRONICS SOCIETY, Montreal, QC. **Anais...** [S.l.: s.n.], 2012. v.12, p.3043 – 3048.

DRESNER, K.; STONE., P. Multiagent traffic management: a reservation-based intersection control mechanism. In: AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, 2004. AAMAS 2004. PROCEEDINGS OF THE THIRD INTERNATIONAL JOINT CONFERENCE ON, VOL., NO., PP. 530-537, JULY 2004. **Anais...** [S.l.: s.n.], 2004.

FERREIRA, M. et al. Self-organized traffic control. In: ACM INTERNATIONAL WORKSHOP ON VEHICULAR INTERNETWORKING (VANET '10), PAGES 85–90., 7. **Proceedings...** [S.l.: s.n.], 2010.

GRADINESCU, V. et al. Adaptive Traffic Lights Using Car-to-Car Communication. In: VEHICULAR TECHNOLOGY CONFERENCE, 2007. VTC2007-SPRING. IEEE 65TH , VOL., NO., PP. 21-25. **Anais...** [S.l.: s.n.], 2007.

KHORANI V.; RAZAVI, F. D. V. A Mathematical Model for Urban Traffic and Traffic Optimization Using a Developed ICA Technique. In: INTELLIGENT TRANSPORTATION SYSTEMS, IEEE TRANSACTIONS ON. **Anais...** [S.l.: s.n.], 2011. p.1024 – 1036.

KRAJZEWICZ, D. et al. Simulation of modern traffic lights control systems using the open source traffic simulation SUMO. In: INDUSTRIAL SIMULATION CONFERENCE 2005, EUROSIS-ETI. 3RD INDUSTRIAL SIMULATION CONFERENCE, PP. 299-302., 3. **Proceedings...** [S.l.: s.n.], 2005.

KRAJZEWICZ, D. et al. Recent Development and Applications of SUMO - Simulation of Urban MObility. In: INTERNATIONAL JOURNAL ON ADVANCES IN SYSTEMS AND MEASUREMENTS, 5 (3E4):128-138, DECEMBER 2012. **Anais...** [S.l.: s.n.], 2012.

MUGNELA, B.; NETTO, M. GenPolis: prototipagem e aplicação de uma ferramenta especializada para otimização via algoritmos genéticos de planos fixos de sinalização semafórica em sub-redes urbanas. In: X SIMPÓSIO BRASILEIRO DE SISTEMAS DE INFORMAçãO (SBSI 2012). **Anais...** [S.l.: s.n.], 2012.

VASIRANI, M.; OSSOWSKI., S. A market-inspired approach to reservation-based urban road traffic management. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS - VOLUME 1 (AAMAS '09), VOL. 1. INTERNATIONAL FOUNDATION FOR AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, RICHLAND, SC, PP. 617-624, 8. **Proceedings...** [S.l.: s.n.], 2009.

APÊNDICES

APÊNDICE A – Detalhes da implementação

A.1 TrafficParser

```
package tgApp;

import java.awt.Point;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import it.polito.appeal.traci.Edge;
import it.polito.appeal.traci.Vehicle;
/**
 * @author Emmanuel Katende Dinanga
 * emmanuelkat@hotmail.com;
 * emmanuel@inf.ufsm.br
 */
public class TrafficParser {

  private static final String CROSS_TEST_FILE = "C:/TCC/Cross-Test" +
      "/cross.sumocfg";
  private static final String SUMO_GUI_PATH =
      "C:/sumo-0.18.0/bin/sumo-gui.exe";
  private static final int END_STEP = 400;              //4 hours
  private static final int SEMAFORIC_INTERAL = 20;
  private static final double LINHA_RETENCAO = 257.0;
  private static final String WEST_EDGE_ID = "w";
  private static final String SOUTH_EDGE_ID = "s";
  private static final String NORTH_EDGE_ID = "n";
  private static final String EAST_EDGE_ID = "e";

  private TrafficSimulation sim;
  private Map<Integer, Double> westMap;
  private Map<Integer, Double> southMap;
  private Edge w, s, e, n;
  private List<Vehicle> nVList, eVList;

  public TrafficParser() {
    sim = new TrafficSimulation(CROSS_TEST_FILE,
        SUMO_GUI_PATH, 5100, true);
    westMap = new HashMap<>();
    southMap = new HashMap<>();
    try {
      sim.runServer();
      w = sim.getEdge(WEST_EDGE_ID);
      s = sim.getEdge(SOUTH_EDGE_ID);
      n = sim.getEdge(NORTH_EDGE_ID);
      e = sim.getEdge(EAST_EDGE_ID);

//      defaultAlgoritm();          // Algoritmo Default
```

```
52 //        trafficLight();             // Semaforo
53 //        longerQueueEver();            // Maior fila sempre
54          longerQueueFirst();           // Maior fila primeiro
55 //        nVehiclesATime(10);           // No minimo N veiculos
56
57          generateGraphic();
58          double xs = getTotalVariationTaxe(s);
59          double xw = getTotalVariationTaxe(w);
60          double dx = getTotalTaxeDifference(xs, xw);
61
62          List<String> content = new ArrayList<String>();
63          content.add("xs: "+xs);
64          content.add("xw: "+xw);
65          content.add("dx: "+dx);
66          FileManager fm = new FileManager();
67          fm.writeFile("result3-4.txt", content);
68          sim.close();
69        } catch (IOException | InterruptedException e) {
70          e.printStackTrace();
71        }
72      }
73
74      public void defaultAlgoritm() throws IllegalStateException, IOException {
75        int time = 0;
76        nVList = sim.getVehicleAtEdge(n);
77        eVList = sim.getVehicleAtEdge(e);
78        while(time < END_STEP) {
79          time = sim.getCurrentSimStep();
80          sim.nextSimStep();
81          getSimDatas();
82        }
83      }
84
85      public void trafficLight() throws IllegalStateException, IOException {
86        int time = sim.getCurrentSimStep();
87        int flag = 0, init = 0;
88        while(time < END_STEP) {
89          init = sim.getCurrentSimStep();
90          if(flag \% 2 == 0) setSouthTrafficLightRed(init,
91              init + SEMAFORIC_INTERAL);
92          else setWestTrafficLightRed(init,
93              init + SEMAFORIC_INTERAL);
94          flag++;
95          getSimDatas();
96          sim.nextSimStep();
97          time = sim.getCurrentSimStep();
98        }
99      }
100
101     public void longerQueueEver() throws IllegalStateException, IOException {
102       int time = sim.getCurrentSimStep();
103       while(time < END_STEP){
104         List<Vehicle> sVList = sim.getVsOutOfZone(s);
105         List<Vehicle> wVList = sim.getVsOutOfZone(w);
106         if(!(sVList == null || wVList == null ||
107             sVList.isEmpty() || wVList.isEmpty())) {
108           if(sVList.size() > wVList.size())
109             passSouthVehicles(sVList, wVList, -1);
```

```
110           else passWestVehicles(sVList, wVList, -1);
111         }
112         getSimDatas();
113         sim.nextSimStep();
114         time = sim.getCurrentSimStep();
115       }
116     }
117
118     public void longerQueueFirst() throws IllegalStateException, IOException
        {
119       int time = sim.getCurrentSimStep();
120       while(time < END_STEP){
121         List<Vehicle> sVList = sim.getVsOutOfZone(s);
122         List<Vehicle> wVList = sim.getVsOutOfZone(w);
123         if(!(sVList == null || wVList == null ||
124             sVList.isEmpty() || wVList.isEmpty())) {
125           if(sVList.size() > wVList.size())
126             passSouthVehiclesFirst(sVList, wVList);
127           else passWestVehiclesFirst(sVList, wVList);
128         }
129         getSimDatas();
130         sim.nextSimStep();
131         time = sim.getCurrentSimStep();
132       }
133     }
134
135     private void nVehiclesATime(int n) throws IllegalStateException,
136     IOException {
137       int time = sim.getCurrentSimStep();
138       while(time < END_STEP){
139         List<Vehicle> sVList = sim.getVsOutOfZone(s);
140         List<Vehicle> wVList = sim.getVsOutOfZone(w);
141         if(!(sVList == null || wVList == null ||
142             sVList.isEmpty() || wVList.isEmpty())) {
143           if(wVList.size() >= n) {
144             passWestVehicles(sVList, wVList, n);
145             sVList = sim.getVsOutOfZone(s);
146             wVList = sim.getVsOutOfZone(w);
147             sVList = sim.getVsOutOfZone(s);
148             if(sVList.size() >= n) {
149               passSouthVehicles(sVList, wVList, n);
150               sVList = sim.getVsOutOfZone(s);
151               wVList = sim.getVsOutOfZone(w);
152             }
153           }
154           else if(sVList.size() >= n) {
155             passSouthVehicles(sVList, wVList, n);
156             sVList = sim.getVsOutOfZone(s);
157             wVList = sim.getVsOutOfZone(w);
158             wVList = sim.getVsOutOfZone(w);
159             if(wVList.size() >= n) {
160               passWestVehicles(sVList, wVList, n);
161               sVList = sim.getVsOutOfZone(s);
162               wVList = sim.getVsOutOfZone(w);
163             }
164           }
165         }
166         getSimDatas();
```

```
167         sim.nextSimStep();
168         time = sim.getCurrentSimStep();
169       }
170     }
171
172     private void setSouthTrafficLightRed(int init, int end)
173         throws IllegalStateException, IOException {
174       while(init < end) {
175         List<Vehicle> sVList = sim.getVsOutOfZone(s);
176         if(!(sVList == null || sVList.isEmpty())) {
177           Vehicle sFirstV = sim.getFirstVOutOfZone(sVList);
178           double d = sim.distance(sFirstV, new Point(0, 0));
179           while(d < LINHA_RETENCAO) {
180             getSimDatas();
181             sim.nextSimStep();
182             init = sim.getCurrentSimStep();
183             if(init >= end) return;
184             d = sim.distance(sFirstV, new Point(0, 0));
185           }
186           sFirstV.stop();
187           while(init < end) {
188             getSimDatas();
189             sim.nextSimStep();
190             init = sim.getCurrentSimStep();
191           }
192           sFirstV.start();
193         }
194         getSimDatas();
195         sim.nextSimStep();
196         init = sim.getCurrentSimStep();
197       }
198     }
199
200     private void setWestTrafficLightRed(int init, int end)
201         throws IllegalStateException, IOException {
202       while(init < end) {
203         List<Vehicle> wVList = sim.getVsOutOfZone(w);
204         if(!(wVList == null || wVList.isEmpty())) {
205           Vehicle wFirstV = sim.getFirstVOutOfZone(wVList);
206           double d = sim.distance(wFirstV, new Point(0, 0));
207           while(d < LINHA_RETENCAO) {
208             getSimDatas();
209             sim.nextSimStep();
210             init = sim.getCurrentSimStep();
211             if(init >= end) return;
212             d = sim.distance(wFirstV, new Point(0, 0));
213           }
214           wFirstV.stop();
215           while(init < end) {
216             getSimDatas();
217             sim.nextSimStep();
218             init = sim.getCurrentSimStep();
219           }
220           wFirstV.start();
221         }
222         getSimDatas();
223         sim.nextSimStep();
224         init = sim.getCurrentSimStep();
```

```
225       }
226     }
227
228     private void passSouthVehicles(List<Vehicle> sVList, List<Vehicle>
229     wVList, int n) throws IllegalStateException, IOException {
230       Vehicle sLastV = sim.getLastVehicle(sVList);
231       if(n != -1) sLastV = sim.getVehicleAtIndex(sVList, n);
232       Vehicle wFirstV = sim.getFirstVehicle(wVList);
233       double d = sim.distance(wFirstV, new Point(0, 0));
234       while(d < LINHA_RETENCAO) {
235         getSimDatas();
236         sim.nextSimStep();
237         d = sim.distance(wFirstV, new Point(0, 0));
238       }
239       wFirstV.stop();
240       while(sim.isVehicleAtEdge(sLastV, s)) {
241         getSimDatas();
242         sim.nextSimStep();
243       }
244       wFirstV.start();
245     }
246
247     private void passWestVehicles(List<Vehicle> sVList, List<Vehicle>
248     wVList, int n) throws IllegalStateException, IOException {
249       Vehicle wLastV = sim.getLastVehicle(wVList);
250       if(n != -1) wLastV = sim.getVehicleAtIndex(wVList, n);
251       Vehicle sFirstV = sim.getFirstVehicle(sVList);
252       double d = sim.distance(sFirstV, new Point(0, 0));
253       while(d < LINHA_RETENCAO) {
254         getSimDatas();
255         sim.nextSimStep();
256         d = sim.distance(sFirstV, new Point(0, 0));
257       }
258       sFirstV.stop();
259       while(sim.isVehicleAtEdge(wLastV, w)) {
260         getSimDatas();
261         sim.nextSimStep();
262       }
263       sFirstV.start();
264     }
265
266     private void passSouthVehiclesFirst(List<Vehicle> sVList, List<Vehicle>
267     wVList) throws IllegalStateException, IOException {
268       Vehicle sLastV = sim.getLastVehicle(sVList);
269       Vehicle wFirstV = sim.getFirstVehicle(wVList);
270       Vehicle wLastV = sim.getLastVehicle(wVList);
271       double d = sim.distance(wFirstV, new Point(0, 0));
272       while(d < LINHA_RETENCAO) {
273         getSimDatas();
274         sim.nextSimStep();
275         d = sim.distance(wFirstV, new Point(0, 0));
276       }
277       wFirstV.stop();
278       while(sim.isVehicleAtEdge(sLastV, s)) {
279         getSimDatas();
280         sim.nextSimStep();
281       }
282       sVList = sim.getVsOutOfZone(s);
```

```
283       Vehicle sFirstV = sim.getFirstVehicle(sVList);
284       sFirstV.stop();
285       wFirstV.start();
286       while(sim.isVehicleAtEdge(wLastV, w)) {
287         getSimDatas();
288         sim.nextSimStep();
289       }
290       sFirstV.start();
291     }
292
293     private void passWestVehiclesFirst(List<Vehicle> sVList, List<Vehicle>
294     wVList) throws IllegalStateException, IOException {
295       Vehicle wLastV = sim.getLastVehicle(wVList);
296       Vehicle sFirstV = sim.getFirstVehicle(sVList);
297       Vehicle sLastV = sim.getLastVehicle(sVList);
298       double d = sim.distance(sFirstV, new Point(0, 0));
299       while(d < LINHA_RETENCAO) {
300         getSimDatas();
301         sim.nextSimStep();
302         d = sim.distance(sFirstV, new Point(0, 0));
303       }
304       sFirstV.stop();
305       while(sim.isVehicleAtEdge(wLastV, w)) {
306         getSimDatas();
307         sim.nextSimStep();
308       }
309       wVList = sim.getVsOutOfZone(w);
310       Vehicle wFirstV = sim.getFirstVehicle(wVList);
311       wFirstV.stop();
312       sFirstV.start();
313       while(sim.isVehicleAtEdge(sLastV, s)) {
314         getSimDatas();
315         sim.nextSimStep();
316       }
317       wFirstV.start();
318     }
319
320     private double getSPassedVehicles() {
321       List<Vehicle> nVAnt = nVList;
322       int t = 0;
323       nVList = sim.getVehicleAtEdge(n);
324       for(Vehicle v : nVList) if(!nVAnt.contains(v)) t++;
325       return t;
326     }
327
328     private double getWPassedVehicles() {
329       List<Vehicle> eVAnt = eVList;
330       int t = 0;
331       eVList = sim.getVehicleAtEdge(e);
332       for(Vehicle v : eVList) if(!eVAnt.contains(v)) t++;
333       return t;
334     }
335
336     private void getSimDatas() {
337       double ts = 0, tw = 0;
338       if(sim.getVehicleAtEdge(s).isEmpty()) ts = 0;
339       else ts = getSPassedVehicles()/sim.getVehicleAtEdge(s).size();
340       if(sim.getVehicleAtEdge(w).isEmpty()) tw = 0;
```

```
341       else tw = getWPassedVehicles()/sim.getVehicleAtEdge(w).size();
342       southMap.put(sim.getCurrentSimStep(), ts);
343       westMap.put(sim.getCurrentSimStep(), tw);
344     }
345
346     private double getTotalTaxeDifference(double xs, double xw) {
347       return Math.abs((xs - xw));
348     }
349
350     private double getTotalVariationTaxe(Edge e) {
351       double sum = 0;
352       if(e == s) {
353         List<Double> tsList = new ArrayList<>(southMap.values());
354         for(Double ts : tsList) sum += ts;
355       }
356       else if(e == w) {
357         List<Double> twList = new ArrayList<>(westMap.values());
358         for(Double tw : twList) sum += tw;
359       }
360       return sum;
361     }
362
363     public void generateGraphic() {
364       FileManager fm = new FileManager();
365       List<Integer> westSteps = new ArrayList<>(westMap.keySet());
366       List<Integer> southSteps = new ArrayList<>(southMap.keySet());
367       Collections.sort(westSteps, new Comparator<Integer>() {
368
369         @Override
370         public int compare(Integer o1, Integer o2) {
371           return o1 - o2;
372         }
373       });
374       Collections.sort(southSteps, new Comparator<Integer>() {
375
376         @Override
377         public int compare(Integer o1, Integer o2) {
378           return o1 - o2;
379         }
380       });
381       List<String> content = new ArrayList<String>();
382       List<String> content2 = new ArrayList<String>();
383       for(int i = 0; i < southSteps.size(); i++)
384         content.add(""+southSteps.get(i)+" "+
385             southMap.get(southSteps.get(i)));
386       fm.writeFile("south.txt", content);
387       for(int i = 0; i < westSteps.size(); i++)
388         content2.add(""+westSteps.get(i)+" "+
389             westMap.get(westSteps.get(i)));
390       fm.writeFile("west.txt", content2);
391     }
392
393     public static void main(String[] args) {
394       TrafficParser parser = new TrafficParser();
395     }
396
397   }
```

A.2 TrafficSimulation

```
1  package tgApp;
2
3  import java.awt.Point;
4  import java.awt.geom.Point2D;
5  import java.io.IOException;
6  import java.util.ArrayList;
7  import java.util.Collection;
8  import java.util.List;
9  import java.util.Map;
10
11 import it.polito.appeal.traci.Edge;
12 import it.polito.appeal.traci.Route;
13 import it.polito.appeal.traci.SumoTraciConnection;
14 import it.polito.appeal.traci.Vehicle;
15 /**
16  * @author Emmanuel Katende Dinanga
17  * emmanuelkat@hotmail.com;
18  * emmanuel@inf.ufsm.br
19  */
20 public class TrafficSimulation extends SumoTraciConnection {
21
22   /**Emmanuel K.D - init the Traffic simulation
23    * Receive the SUMO config file (.sumocfg)
24    * Receive the SUMO GUI Path
25    * and the random seed number and create a connection*/
26   public TrafficSimulation(String sumoCfgFile, String sumoGUIPath,
27       int randomSeed, boolean quitOnFinish) {
28     super(sumoCfgFile, randomSeed);
29     System.setProperty(SUMO_EXE_PROPERTY, sumoGUIPath);
30     if(quitOnFinish) addOption("quit-on-end", "1");
31   }
32
33   public List<Vehicle> getAllVehicules() {
34     try {
35       Collection<Vehicle> vehicles = getVehicleRepository().
36           getAll().values();
37       List<Vehicle> v = new ArrayList<>(vehicles);
38       return v;
39     } catch (IOException e) {
40       e.printStackTrace();
41       return null;
42     }
43   }
44
45   public List<Edge> getAllEdges() {
46     try {
47       Collection<Edge> edges = getEdgeRepository().
48           getAll().values();
49       List<Edge> e = new ArrayList<>(edges);
50       return e;
51     } catch (IOException e) {
52       e.printStackTrace();
53       return null;
54     }
55   }
```

```
56
57    public List<Route> getAllRoutes() {
58      try {
59        Map<String, Route> routeMap = getRouteRepository().getAll();
60        List<Route> routes = new ArrayList<>(routeMap.values());
61        return routes;
62      } catch (IOException e) {
63        e.printStackTrace();
64        return null;
65      }
66    }
67
68    public List<Vehicle> getVehicleAtEdge(Edge edge) {
69      List<Vehicle> res = new ArrayList<Vehicle>();
70      for(Vehicle v : getAllVehicules()) {
71        try {
72          if(v.queryReadCurrentEdge().get().equals(edge)) res.add(v);
73        } catch (IOException e1) {
74          return null;
75        }
76      }
77      return res;
78    }
79
80    public Edge getEdge(String edgeId) {
81      for(Edge e : getAllEdges())
82        if(e.getID().equals(edgeId)) return e;
83      return null;
84    }
85
86    public Vehicle getFirstVehicle(List<Vehicle> vehicles) {
87      if(vehicles.isEmpty()) return null;
88      List<String> aux = Tools.mySplit(vehicles.get(0).
89          getID().trim(), '.');
90      int minor = Integer.parseInt(aux.get(aux.size()-1));
91      Vehicle res = vehicles.get(0);
92      for(Vehicle v : vehicles) {
93        aux = Tools.mySplit(v.getID().trim(), '.');
94        int n = Integer.parseInt(aux.get(aux.size()-1));
95        if(n < minor) {
96          minor = n;
97          res = v;
98        }
99      }
100     return res;
101   }
102
103   public Vehicle getLastVehicle(List<Vehicle> vehicles) {
104     if(vehicles.isEmpty()) return null;
105     Vehicle res = vehicles.get(0);
106     int major = 0;
107     for(Vehicle v : vehicles) {
108       List<String> aux = Tools.mySplit(v.getID().trim(), '.');
109       int n = Integer.parseInt(aux.get(aux.size()-1));
110       if(n > major) {
111         major = n;
112         res = v;
113       }
```

```
114       }
115       return res;
116     }
117
118     public boolean isVehicleAtEdge(Vehicle v, Edge e) {
119       if(v == null) return false;
120       try {
121         if(v.queryReadCurrentEdge().get().equals(e))
122           return true;
123         else return false;
124       } catch (IOException e1) {
125         return false;
126       }
127     }
128
129     public double distance(Vehicle v, Point2D point) {
130       try {
131         Point2D p = v.queryReadPosition().get();
132         return p.distance(point);
133       } catch (IOException e) {
134         return -1;
135       }
136     }
137
138     public Vehicle getFirstVOutOfZone(List<Vehicle> vehicles) {
139       Vehicle firstV = getLastVehicle(vehicles);
140       if(firstV == null) return null;
141       double minor = distance(firstV, new Point(0, 0));
142       if(minor > 260.0) return null;
143       for(Vehicle v : vehicles) {
144         double d = distance(v, new Point(0, 0));
145         if(d > minor && d < 260.0) {
146           minor = d;
147           firstV = v;
148         }
149       }
150       return firstV;
151     }
152
153     public List<Vehicle> getVsOutOfZone(Edge e) {
154       List<Vehicle> vehicles = getVehicleAtEdge(e);
155       if(vehicles == null) return null;
156       List<Vehicle> res = new ArrayList<Vehicle>();
157       for(Vehicle v : vehicles) {
158         double d = distance(v, new Point(0, 0));
159         if(d <= 260.0) res.add(v);
160       }
161       return res;
162     }
163
164     public Vehicle getVehicleAtIndex(List<Vehicle> vehicles, int numVeh) {
165       if(vehicles.size() < numVeh) return null;
166       do {
167         Vehicle firstV = getFirstVehicle(vehicles);
168         vehicles.remove(firstV);
169         numVeh--;
170       }while(numVeh > 1);
171       return getFirstVehicle(vehicles);
```

```
172     }
173
174 }
```

A.3 Tools

```
1  package tgApp;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  /**
7   * @author Emmanuel Katende Dinanga
8   * emmanuelkat@hotmail.com;
9   * emmanuel@inf.ufsm.br
10  */
11 public class Tools {
12
13   public static List<String> mySplit(String s, char c) {
14     if(s.isEmpty() || s == null) return null;
15     List<String> list = new ArrayList<String>();
16     int i = 0;
17     do {
18       char d = s.charAt(i);
19       String aux = "";
20       while(d != c && i < s.length()) {
21         aux += d;
22         i++;
23         if(i < s.length()) d = s.charAt(i);
24       }
25       list.add(aux); i++;
26     }while(i < s.length());
27     return list;
28   }
29
30 }
```

A.4 FileManager

```
1  package tgApp;
2  /**
3   * @author Emmanuel Katende Dinanga
4   * emmanuelkat@hotmail.com;
5   * emmanuel@inf.ufsm.br
6   */
7  import java.io.*;
8  import java.util.ArrayList;
9  import java.util.List;
10
11 public class FileManager {
12
13   private BufferedReader reader;
14   private BufferedWriter writer;
15   private BufferedInputStream wrReader;
```

```
16
17    public FileManager() {}
18
19    protected List<String> readFile(String file) {
20      try {
21        List<String> lines = new ArrayList<String>();
22        reader = new BufferedReader(new FileReader(file));
23        while(reader.ready()) {
24          String line = reader.readLine();
25          if(!line.equals(null)) lines.add(line);
26        }
27        return lines;
28      } catch (Exception e) {
29        System.out.println(e);
30        return null;
31      }
32    }
33
34    protected boolean writeFile(String file, List<String> content) {
35      try {
36        writer = new BufferedWriter(new FileWriter(file));
37        int i = 0;
38        while(i < content.size()) {
39          writer.write(content.get(i));
40                  writer.write("\n");
41          i++;
42        }
43        writer.close();
44        return true;
45      } catch (Exception e) {
46        return false;
47      }
48    }
49
50    public List<String> readWrittenFile(String file) {
51      try {
52        String line = "";
53        List<String> lines = new ArrayList<>();
54        wrReader = new BufferedInputStream(new FileInputStream( "log.txt" ) )
              ;
55              while( wrReader.available() > 0 ) {
56                  char c = (char)wrReader.read();
57                  if(c != '\n') line += c;
58                  else {
59                      line += '\0';
60                      lines.add(line);
61                      line = "";
62                  }
63              }
64              wrReader.close();
65        return lines;
66      } catch (Exception e) {
67        return null;
68      }
69    }
70
71  }
```