UNIVERSIDADE FEDERAL DO CEARÁ
PRÓ-RETORIA DE PESQUISA E PÓS-GRADUAÇÃO
CENTRO DE CIÊNCIAS
PROGRAMA DE MESTRADO E DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

ANTÔNIO MANOEL RIBEIRO DE ALMEIDA

GPS2GR: FORMAÇÃO OTIMIZADA DE COMBOIOS DE VEÍCULOS EM REDES URBANAS COM BASE EM TRAJETÓRIAS DE GPS

FORTALEZA
2018

ANTÔNIO MANOEL RIBEIRO DE ALMEIDA

GPS2GR: FORMAÇÃO OTIMIZADA DE COMBOIOS DE VEÍCULOS EM REDES URBANAS COM BASE EM TRAJETÓRIAS DE GPS

Tese apresentada à coordenação do curso de do Programa de Mestrado e Doutorado em Ciência da Computação do Pró-retoria de Pesquisa e Pós-graduação da Universidade Federal do Ceará, como requisito à obtenção do título de doutor. Área de Concentração: Ciência da Computação

Orientador: Prof. Dr. Javam de Castro Machado

Coorientador: Prof. Dr. José Antônio Fernandes Macedo

FORTALEZA

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A444g Almeida, Antonio Manoel Ribeiro de.
GPS2GR: Formação otimizada de comboios de veículos em redes urbanas com base em trajetórias de GPS / Antonio Manoel Ribeiro de Almeida. – 2018.
122 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação , Fortaleza, 2018.
Orientação: Prof. Dr. Javam de Castro Machado.
Coorientação: Prof. Dr. Jose Antonio Fernandes Macedo.

1. Mineração de Padrões em Trajetórias. 2. Rotas Frequentes. 3. Mapeamento de Trajetórias. 4. Controle de Semáforos. I. Título.

CDD 005

ANTÔNIO MANOEL RIBEIRO DE ALMEIDA

GPS2GR: FORMAÇÃO OTIMIZADA DE COMBOIOS DE VEÍCULOS EM REDES URBANAS COM BASE EM TRAJETÓRIAS DE GPS

Tese apresentada à coordenação do curso de do Programa de Mestrado e Doutorado em Ciência da Computação do Pró-retoria de Pesquisa e Pós-graduação da Universidade Federal do Ceará, como requisito à obtenção do título de doutor. Área de Concentração: Ciência da Computação

Aprovada em: 14/08/2018

BANCA EXAMINADORA

Prof. Dr. Javam de Castro Machado (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. José Antônio Fernandes Macedo (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Gerardo Valdísio Rodrigues Viana
Universidade Estadual do Ceará (UECE)

Prof. Dr. José Maria da Silva Monteiro Filho
Universidade Federal do Ceará (UFC)

Prof. Dr. Sérgio Lifschitz
Pontifícia Universidade Católica do Rio de Janeiro (PUC-RIO)

Este trabalho é dedicado à minha família, pelo apoio incondicional em todos os momentos; à minha mulher, Viviane, que me deu cobertura nas obrigações familiares. Sem vocês, nenhuma conquista teria valor. Dedico aos nossos filhos, Felipe e Davi, para lhes sirvam de inspiração futura, a fim de que sempre lutem pelos seus objetivos.

AGRADECIMENTOS

Aos Professores Dr. Javam de Castro Machado e Dr. José Antônio Fernandes Macedo, por me orientarem nesta tese. Ao coordenador do Laboratório de Sistemas e Banco de Dados (LSBD), onde esta pesquisa foi desenvolvida. Aos meus pais, irmãos e sobrinhos, que, nos momentos de minha ausência dedicados ao estudo superior, sempre fizeram entender que o futuro é feito a partir da constante dedicação no presente!

Agradeço a todos os professores por me proporcionarem o conhecimento, não apenas racional, mas também a manifestação do caráter e afetividade da educação no processo de formação profissional, por tanto que se dedicaram a mim, não somente por terem me ensinado, mas por haverem me feito aprender.

Esta pesquisa foi parcialmente apoiada pela Fundação Cearense de Apoio ao Desenvolvimento Científico e Tecnológico - FUNCAP e também pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES, para sustento financeiro por meio de bolsa de pós-graduação.

Agradecimentos especiais são direcionados ao Laboratório de Sistemas e Banco de Dados[1] da Universidade Federal do Ceará (UFC), ao grupo de pesquisa Datamining e Aprendizagem de Máquina, que contribuíram definitivamente para que eu concluísse este trabalho. E de forma ampla, a todos os colegas pesquisadores do grupo de pesquisa do LSBD pelas discussões às segundas-feiras.

[1] <http://www.lsbd.ufc.br/>

”As invenções são principalmente o resultado de um trabalho teimoso.”

(Santos Dumont)

RESUMO

Propõe um método de otimização clássico (offline) computacionalmente factível que otimize o fluxo global de veículos por uma programação dos semáforos, que incentive a formação de comboios de veículos viajando em "ondas verdes"por rotas frequentes, utilizando como entrada a densidade de fluxo estimado por trajetórias coletadas de GPS's. Para isso, foi utilizado um escalonador que gera um movimento de onda verde nos fluxos mais frequentes da cidade, não se limitando a avenidas ou mudanças de ruas de um conjunto de trajetórias. A descoberta de rotas frequentes utiliza o princípio da conservação mínima do fluxo de tráfego por janela de tempo. As janelas de tempo de larguras heterogêneas permitem obter soluções específicas para cada momento durante o dia de uma cidade. A estimativa de fluxo do segmento de rua é feita mapeando-se as coordenadas de GPS para o segmento de rua e contando cada trajetória que passa no segmento de rua, para se ter precisão na estimativa de tráfego por segmentos de rua e por janela de tempo. A "discretização"temporal é feita agrupando-se as trajetórias na linha do tempo, reunindo os horários que concentram naturalmente muitas trajetórias. Na teste relatada utilizam-se para os experimentos as trajetórias de uma empresa de táxi da cidade de Fortaleza, mas a solução pode usar trajetórias de outras fontes, como Waze. Entende-se que este modelo pode ser a alternativa centralizada de orquestração do trânsito urbano, utilizando os mesmos equipamentos para permitir maior fluxo do tráfego urbano diário, maximizando-o.

Palavras-chave: Mineração de Padrões em Trajetórias. Rotas Frequentes. Mapeamento de Trajetórias. Controle de Semáforos

ABSTRACT

This work proposes a computationally feasible optimization method (offline) that optimizes the global flow of vehicles through traffic signal programming that encourages the formation of vehicle convoys traveling in a green wave by frequent routes using as input the estimated flow density by trajectories collected from GPS's. For this, a schedule was used that generates a green wave movement in the most frequent flows of the city, not being limited to avenues or changes of streets of a set of trajectories. Frequent route discovery uses the principle of minimum traffic flow conservation per time window, time windows of heterogeneous widths allow you to obtain specific solutions for each for each moment throughout the day of a city. The street segment flow estimation is done by mapping the GPS coordinates to the street segment and counting each trajectory that passes in the street segment, in order to have greater precision in estimating traffic by street segments and by the time window. The temporal discretization is done by grouping the trajectories in the timeline, grouping the schedules that naturally concentrate many trajectories. In this work, we are using for the experiments the trajectories of a taxi company in the city of Fortaleza, but the solution can use trajectories from other sources, such as Waze. We understand that this model can be a centralized alternative of urban traffic orchestration, using the same equipment to allow a greater flow of daily urban traffic, maximizing it.

Keywords: Trajectory Pattern Mining. Hot Route. Map-Matching. Traffic-Light Scheduler.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE CÓDIGOS-FONTE

GLOSSÁRIO ABREVIATURAS E SIGLAS

A* *A* é um algoritmo para Busca de Caminho. Ele busca o caminho em um grafo de um vértice inicial até um vértice final. Ele é a combinação de aproximações heurísticas como do algoritmo Breadth First Search (Busca em Largura) e da formalidade do Algoritmo de Dijkstra.*

BML *O modelo de tráfego Biham – Middleton – Levine (BML) é um modelo de fluxo de tráfego de autômato celular auto-organizado bidimensional para tráfego urbano. Consiste em um número de carros representados por pontos em uma rede com uma posição inicial aleatória, onde cada carro pode ser de dois tipos: aqueles que só se movem para baixo e os que só se movem em direção ao direito.*

DBSCAN *O agrupamento espacial baseado em densidade de aplicativos com ruído (DBSCAN) é um algoritmo de clustering de dados proposto por Martin Ester, Hans-Peter Kriegel , Jörg Sander e Xiaowei Xu, em 1996. É um algoritmo de clustering baseado em densidade : de pontos em algum espaço, agrupa pontos que estão intimamente agrupados (aponta com muitos vizinhos próximos), marcando como outliers pontos que ficam sozinhos em regiões de baixa densidade (cujos vizinhos mais próximos estão muito distantes). O DBSCAN é um dos mais comuns algoritmos de clustering e também mais citado na literatura científica.*

GPS *Sistema de posicionamento global, mais conhecido pela sigla GPS (em inglês global positioning system) é um sistema de posicionamento por satélite que fornece a um aparelho receptor móvel a sua posição, assim como informação horária, sob quaisquer condições atmosféricas, a qualquer momento e em qualquer lugar na Terra, desde que o receptor se encontre no campo de visão de três satélites de posicionamento global.*

GPX *GPX, ou GPS eXchange Format (Formato de Intercâmbio GPS) é um esquema XML designado para transferir dados GPS entre aplicações.*

K-MEANS *k-means é um método de quantização vetorial, originalmente de processamento de sinal, que é popular para análise de cluster em mineração de dados . k significa clusters visa particionar n observações em k clusters nos quais cada observação pertence ao cluster com a média mais próxima, servindo como um protótipo do cluster. Isso resulta em um "particionamento" do espaço de dados nas células de Voronoi.*

MISE *Na Estatística , o erro médio quadrático integrado (MISE) é usado na estimativa de densidade . O MISE de uma estimativa de uma densidade de probabilidade desconhecida é dado por f onde f é a densidade desconhecida, f_n é sua estimativa baseada em uma amostra de n variáveis aleatórias independentes e identicamente distribuídas.*

OFF-LINE *Otimização Off-line, campo da teoria de otimização , mais popular em Ciência da Computação e pesquisa operacional , que lida com problemas de otimização clássica, onde se visa obter a solução ideal e com base num conjunto completo de entrada de variáveis.*

ON-LINE *Otimização on-line é um campo da teoria de otimização que lida com problemas de otimização sem conhecimento ou conhecimento incompleto do problema. Normalmente, usa métodos adaptativos provenientes da teoria de análise de sinais. Esses tipos de problemas são denotados como problemas on-line e são vistos como opostos aos problemas de otimização clássicos em que informações completas são assumidas (off-line).*

OSM *OpenStreetMap (OSM) é um projeto de mapeamento colaborativo para criar um mapa livre e editável do mundo, inspirado por sites como a Wikipédia. Traduzindo para português, o nome significa Mapa Aberto de Ruas. Ele fornece dados a centenas de sites na internet, aplicações de aparelho celular e outros dispositivos.*

PDF *Em teoria das probabilidades e estatística, a função densidade de probabilidade ou a probability density function (PDF) em inglês, é uma função que descreve a probabilidade relativa de uma variável aleatória tomar um valor dado. A probabilidade da variável aleatória cair em uma faixa particular é dada pela integral da densidade dessa variável sobre tal faixa - isto é, é dada pela área abaixo da função densidade mas acima do eixo horizontal e entre o menor e o maior valor dessa faixa. A função densidade de probabilidade é não negativa sempre, e sua integral sobre todo o espaço é igual a um. A função densidade pode ser obtida com base na função distribuição acumulada desde da operação de derivação.*

Poisson *Função de probabilidade da distribuição de Poisson para vários valores de λ. Na teoria da probabilidade e na Estatística, a distribuição de Poisson é uma distribuição de probabilidade de variável aleatória discreta que expressa a probabilidade de uma série de eventos acontecem num certo período de tempo se estes eventos ocorrem independentemente de quando sucedeu o último evento.*

SSE *Sum of the Squared Errors é uma medida estatística da discrepância entre os dados e um modelo de estimativa.*

XML *XML é a sigla para Extensible Markup Language, que significa em português Linguagem Extensível de Marcação Genérica. É uma recomendação para gerar linguagens de marcação para necessidades especiais.*

LISTA DE SÍMBOLOS

Ψ	Um conjunto de configurações de temporização de sinal
C_n	Tempos de ciclo de um semáforo n
θ_n	Compensação de interseção n
ϕ_{np}	Fase p de tempo de verde para interseção n
I_{np}	tempo perdido entre o final da fase p até o próximo

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, houve um aumento dramático no número de sistemas de viagem portáteis e de GPS embutidos em smartfones. A proliferação desses dispositivos permitiu a coleta de uma grande quantidade de trajetórias de GPS. Muitas aplicações, como planejadores de rotas, buscadores de rotas e redes sociais georreferenciadas, começaram a usar informações de dados GPS para obter melhor qualidade de serviço.

O congestionamento do tráfego é uma das principais causas de perda de produtividade e diminuição do padrão de vida em ambientes urbanos. Os dados gerados por smartfones com GPS fornecem conjuntos de dados valiosos de onde é possível extrair conhecimentos úteis para encontrar soluções para problemas de trânsito urbano.

O congestionamento do tráfego causa atrasos que aumentam os custos significativos para a sociedade e os negócios, diariamente, e também aumentam as emissões de CO^2 e o risco de acidentes. O estudo (WARBERG *et al.*, 2008) informa que, em 2002, as pessoas que passaram 100 mil horas em filas na infraestrutura rodoviária da Grande Copenhague geraram uma perda econômica de mais de 750 milhões de euros.

A figura 1 mostra uma realidade típica na cena urbana, um cruzamento de avenidas com semáforos não sincronizado.

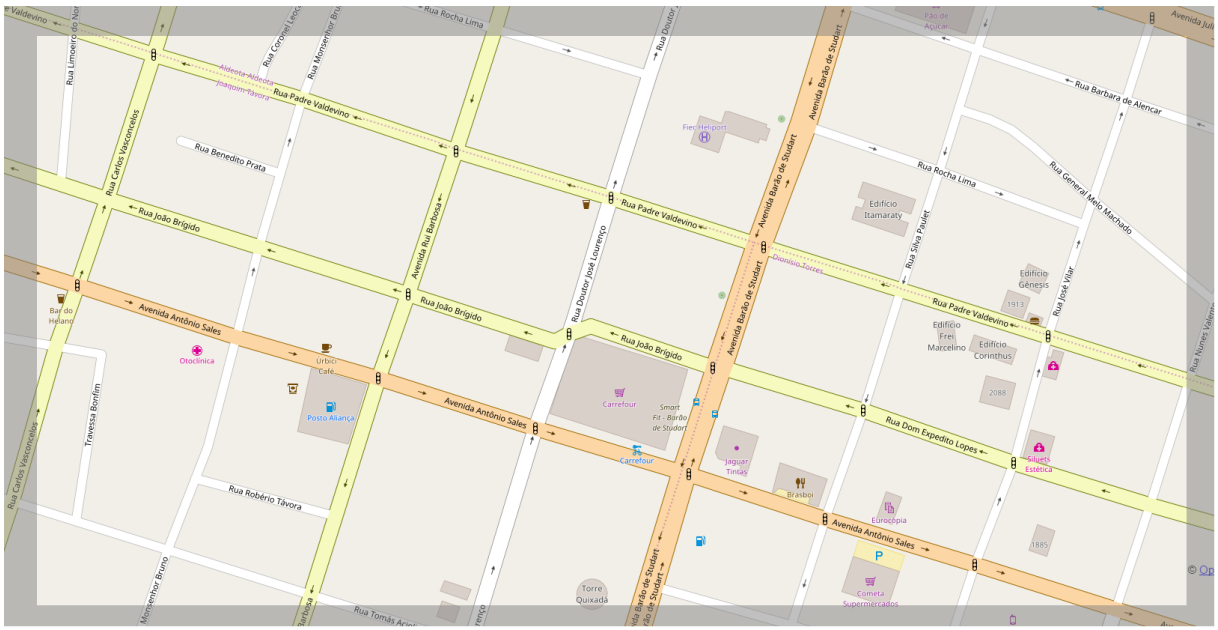


Figura 1 – Cruzamento de avenidas com semáforos nas aproximações

Com o aumento do fluxo de veículos, esta cena muda para a situação da figura 2, onde ocorrem saturação e congestionamentos.

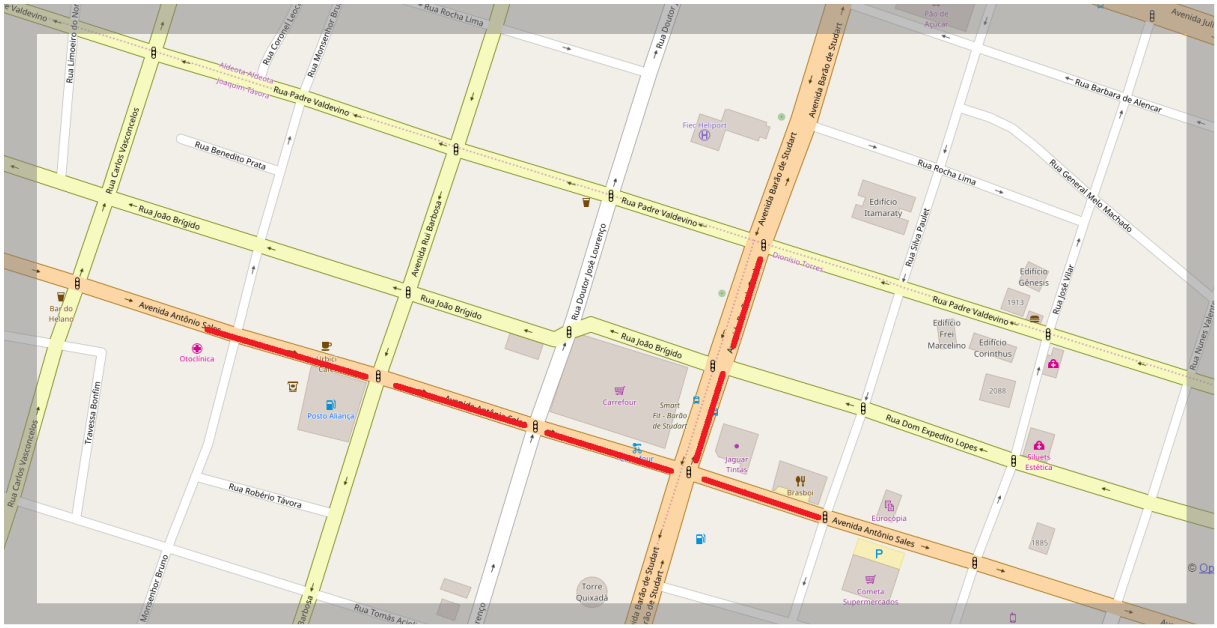


Figura 2 – Efeito do aumento da concentração de veículos é a saturação nas aproximações dos semáforos

Uma solução é a sincronização dos semáforos na avenida de maior fluxo, como na figura 3, em que o fluxo melhora na avenida principal, porém, na via secundária o problema se mantém.

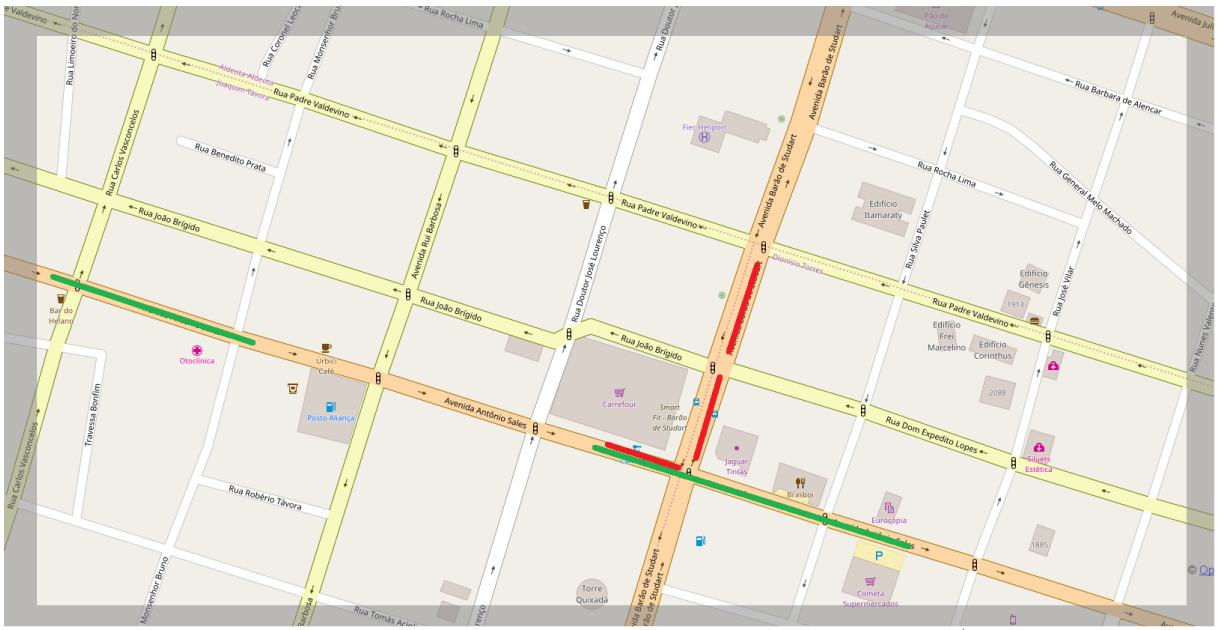


Figura 3 – Solução 1 - Geração de Onda Verde na via de maior fluxo (WARBERG *et al.*, 2008; CORMAN *et al.*, 2009)

Outra solução é instalar semáforos inteligentes, que, mediante sensores instalados na rua, medem a concentração de carros e ajustam os tempos dos semáforos na busca de uma melhoria como na figura 4. Repetindo o ciclo ou retornando a configuração

anterior, caso a situação não melhore.

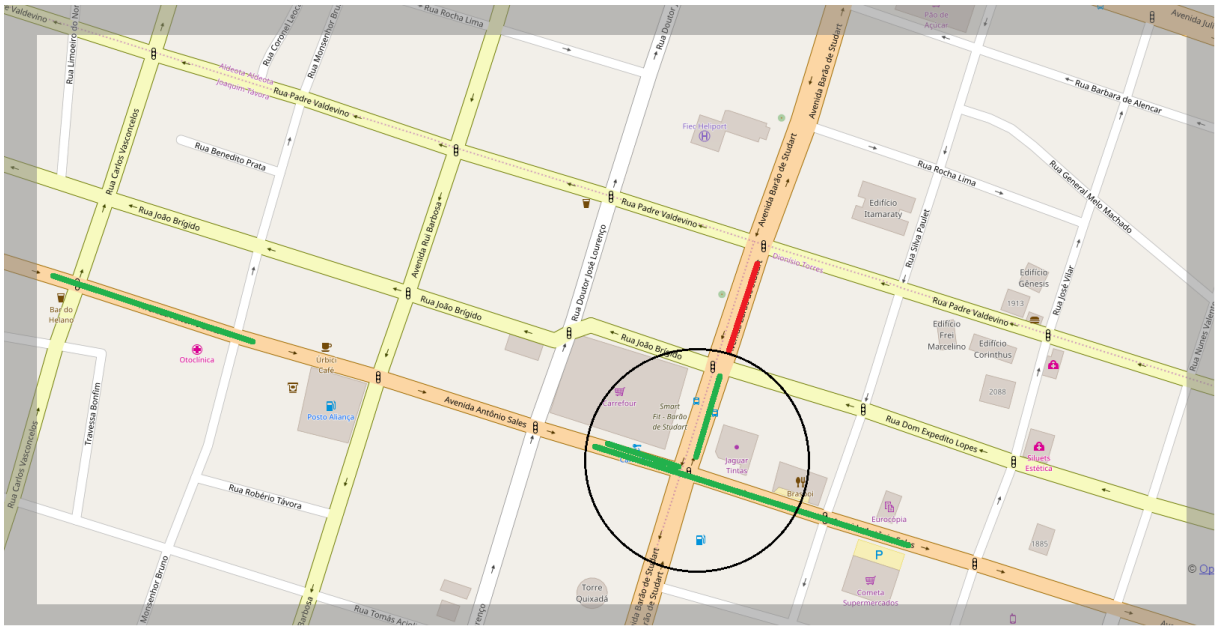


Figura 4 – Solução 2 - Geração de Onda Verde na via de maior fluxo, com semáforos inteligentes (KANUNGO *et al.*, 2014)

Para aliviar o congestionamento, o transporte público pode ser melhorado ou a infraestrutura pode ser expandida. Nas áreas urbanas, este último é, muitas vezes, impossível, em razão da existência de áreas residenciais adjacentes às estradas. A maneira mais sutil de melhorar o desempenho da rede é fazer um melhor uso das estradas, o que pode ser conseguido, em parte, pela configuração adequada, otimização dos parâmetros do sinal de trânsito, aumentando a eficiência de fluxo global dos veículos e reduzindo as perdas (WARBERG *et al.*, 2008).

Nosso problema é como criar um modelo de otimização para programar os semáforos, que incentive a formação de comboios de veículos, seguindo uma ”onda verde” por meio de rotas cujos fluxos sejam mais frequentes. Para isto, fazemos uso, como entrada do problema, da variação de fluxo de veículos, estimada por trajetórias de GPS, tendo como desafio o tratamento de trajetórias capturadas de aparelhos de GPS, que frequentemente contém falhas de amostragem. No uso de aparelhos com a função de GPS, pode ocorrer falha na leitura dos satélites, impedindo a obtenção da coordenada em um determinado ponto, decorrente da ocorrência de nuvens ou obstáculos, como edifícios, que não permitem que dispositivos GPS acessem os satélites, necessários para leitura correta.

Essas falhas produzem trajetórias com lacunas que precisam ser preenchidas para criar um caminho completo. A fim de resolver este problema, aplicamos um algoritmo

de correspondência de mapas que, com a ajuda da técnica A^*, é capaz preenchê-las as lacunas em falta nas trajetórias e aumentar significativamente a precisão da partida (ALMEIDA *et al.*, 2016a).

Como se vê no exemplo da figura 5, podemos observar algumas trajetórias interrompidas e continuando de um local improvável, gerando assim uma falha na trajetória, ou seja, trajetórias com baixa amostragem de pontos ou falhas no registro das coordenadas que necessitam de um método para completar o caminho de maneira correta (MIWA *et al.*, 2012). A maioria das abordagens atuais de correspondência de mapas lida apenas com trajetórias de GPS, com alta taxa de amostragem - tipicamente um ponto a cada dez ou 30 segundos, e estas abordagens tornam-se menos efetivas para trajetórias de baixa taxa de amostragem, aumentando a incerteza de encontrar o caminho correto.

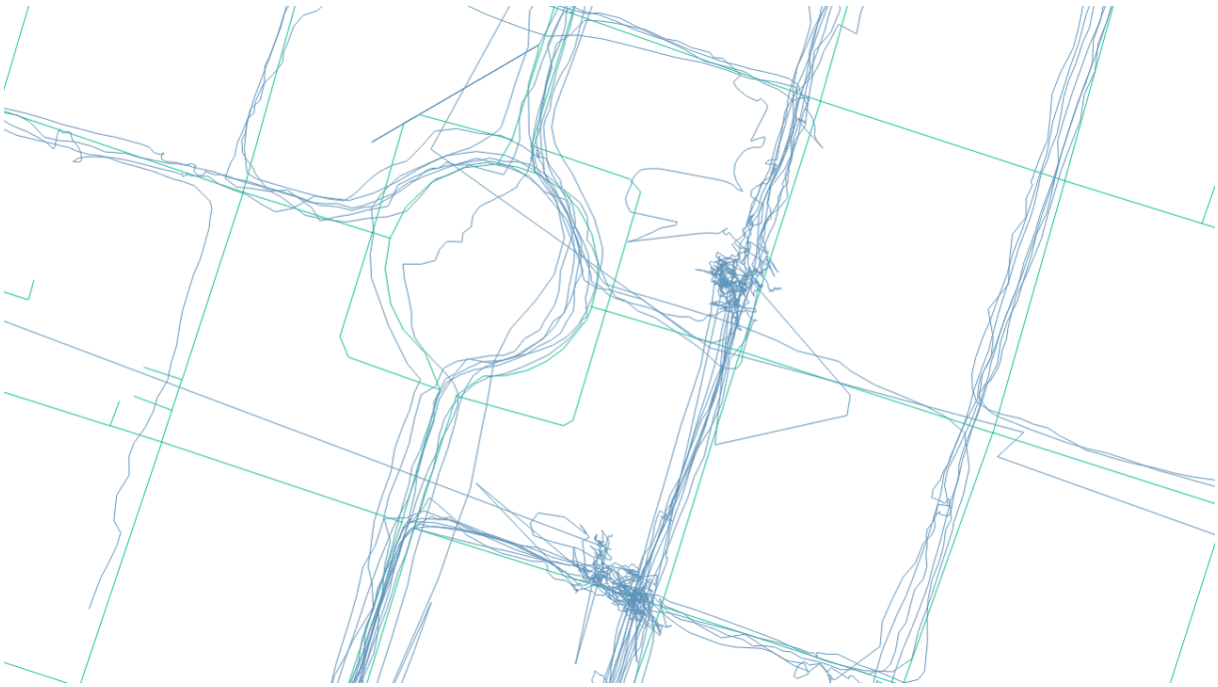


Figura 5 – Pequeno exemplo de trajetórias reais de táxis na cidade de Fortaleza, Ceará, Brasil

Outro desafio é como identificar automaticamente rotas frequentes de um conjunto de caminhos, de modo a identificar o padrão de deslocamento em grupo de veículos de uma cidade. Propomos, nesta tese, estimar a intensidade do tráfego, com base na densidade de caminhos, agrupar em janelas de tempo, formando, assim, o que a literatura denomina Hot Route ou rota frequente.

Devemos, também, considerar os benefícios da escala de tempo de agrupamento, usando técnicas de ”discretização” para permitir que comportamentos específicos sejam

capturados quando se dividem as trajetórias em grupos. Existem técnicas que permitem a ”discretização” do tempo e as trajetórias dentro de cada grupo ou *cluster*.

Um comportamento comum em conjuntos de dados de pontos de trajetória é a ocorrência de séries temporais. Série de tempo é uma sequência de pontos de tempo no espaço, onde cada ponto possui valor discreto. Em um conjunto de dados de pontos de trajetória, esses valores discretos podem ser entendidos como timestamps. De acordo com (FU, 2011), a tarefa de descobrir padrões em uma série de tempo é desafio para escolher a melhor técnica de ”discretização” do tempo em janelas. As etapas envolvidas na solução deste problema exigem técnicas de ”discretização” ou agrupamento.

Abordamos esse problema de agrupamento de séries temporais de trajetórias aplicando quatro técnicas distintas: limites com redução de escala, limites do histograma, limite de agrupamento com *k-means* e limites de agrupamento com DBSCAN. Os dois primeiros métodos têm uma abordagem de ”discretização”, os outros dois estão relacionados a agrupamento de pontos. Os pontos de dados de nossos conjuntos de dados são distribuídos em um espaço tridimensional. A primeira e a segunda dimensões são, respectivamente, a latitude e a longitude de um ponto de trajetória. A terceira dimensão tem a data e a hora da medição da coordenada no GPS. Como nosso objetivo inicial é agrupar trajetórias por janela de tempo, de maneira que trajetórias frequentes em uma janela de tempo possam ser reveladas, apenas a dimensão temporal é utilizada neste etapa do processo. Nossas experiências são todas baseadas na terceira dimensão. Portanto, nossos conjuntos de dados podem ser pensados como séries temporais de pontos de dados em um espaço tridimensional.

A qualidade de cada *cluster* gerado nos experimentos deste trabalho é medida levando-se em consideração o número de *clusters* gerados é a soma de erro dos quadrados (SSE) produzido por parte de cada algoritmo. Além disso, o limite de Sturges (STURGES, 1926) está sendo usado para estimar o número ideal de *clusters* com base no número de pontos de dados. Portanto, temos um limite superior que nos informa quando a técnica está produzindo um resultado aceitável.

Nossos experimentos mostraram que algumas técnicas se comportam melhor para determinados conjuntos de dados. Basicamente, esses conjuntos de dados diferem uns dos outros pelo número de pontos de dados que estão sendo avaliados. Alguns algoritmos podem não ser capazes de lidar com erros de amostragem GPS, ensejando caminhos

correspondentes incorretos e sem significado. Nossa proposta é estender e otimizar o algoritmo de correspondência, ST-Matching, pois ele escolhe bons caminhos, mesmo com trajetórias de GPS com falhas e baixa amostragem, como já mencionado. Propomos, também, estender o algoritmo de busca de caminhos frequentes, FlowScan, por ter conceitos matemáticos simples e convenientes quando se dispõe da informação de densidade de fluxo por segmento de rua.

Definição do Problema

Otimizar o fluxo global de veículos por intermédio de uma programação dos tempos dos semáforos que favoreça o deslocamento em comboio de veículos que seguindo um padrão global das trajetórias identificadas.

Nesta circunstancia, o objetivo é de pesquisar e desenvolver, tendo como entrada a densidade de fluxo estimada por trajetórias coletadas em aplicativos de GPS's, um método off-line, factível computacionalmente, que otimize o fluxo global de veículos de uma cidade, por meio da programação dos semáforos e que incentive a formação de comboios de veículos trafegando em ondas verdes, por meio de rotas frequentes.

A abordagem off-line do problema permite acumular conhecimento por meio do reconhecimento de padrão de trajetórias para cada janela de tempo. Em condições rotineiras e diárias do tráfego de veículos, o fluxo de veículos nas cidades obedece a padrões de rotas e horários. Assim, é necessário a otimização dos tempos dos semáforos para melhor atender a esse padrão de fluxo. Como a otimização do fluxo não pode ser individualizada por veículo, criou-se o conceito de comboio virtual de veículos, que representa uma região da via que se desloca em uma velocidade média, onde os veículos que estiverem neste espaço e se deslocarem na mesma direção e velocidade do comboio, pertencem a este comboio até o fim da rota. Os veículos que estiverem neste espaço-tempo e se mantiverem na rota terão trafego à frente livre, dada a otimização prévia dos semáforos.

Desafios e Hipóteses

1. Desafio: como criar janelas de tempos variáveis para melhor representar a variação de densidade das trajetórias no decorrer do dia?
    - Hipótese: em princípio, janelas de tempo variáveis se ajustam melhor às variações de fluxo de trajetórias durante o dia. Ao se utilizar uma medida de erro, os modelos de janela fixa terão erros maiores do que das janelas variáveis. As janelas variáveis, em hipótese, permitirão gerar menor números de janelas, o

que reduzirá a complexidade das soluções, visto que, para cada janela, haverá solução distinta.

A hipótese é que, se evitarmos as abordagens clássicas de ”discretização” como histograma ou redução de escala, e utilizarmos métodos de clusterização para identificar os agrupamentos de trajetórias na linha do tempo e, após isso, utilizarmos as fronteiras dos agrupamentos para definir os limites das janelas de tempo, teremos janelas de tamanho variável e mais ajustadas à distribuição das trajetórias.

2. Desafio: como automatizar contagem de trajetórias nas ruas, visto que os métodos atuais são de alto custo, por via de contagem manual em campo ou por intermédio da instalação de sensores nas ruas?
    - Hipótese: dado que o grande volume de trajetórias registradas por dispositivos de uso pessoal, como smartfones, é praticamente unânime, caso haja um método que transforme os *Datasets* de trajetórias em contagem de fluxo por segmento de rua, se poderia reduzir muito o custo e o tempo deste processo.

      A hipótese, neste caso, é que, se usando um algoritmo preciso de Map-matching, é possível criar um método de contagem automatizado, por segmento de rua, que aproveita os dados públicos de trajetórias para inferir a densidade relativa de fluxo de veículos, por segmento de rua.
3. Desafio: conforme (WARBERG *et al.*, 2008), o problema da otimização dos semáforos é de alto custo, ao ponto de ser inviável computacionalmente. Portanto deve-se buscar a maneira de reduzir a complexidade do problema. Acreditamos seja oportuno reduzir o contexto de otimização apenas para as ruas e semáforos que demonstrem volume significativo numa janela de tempo. Caso contrário, estes semáforos ficariam fora do problema e, portanto, com ciclo não sincronizado com demais semáforos que estarão no contexto do problema. O desafio, portanto, consiste em como descobrir as rotas mais frequentes de uma janela de tempo.
    - Hipótese: dado que o problema de obter a densidade de fluxo por segmento de rua esteja resolvido, aplicar um algoritmo de preservação de fluxo com valor de corte baixo, de três a cinco veículos, para revelar significativamente as principais rotas de veículos para cada janela de tempo.
4. Desafio: como otimizar os tempos dos semáforos para privilegiar os fluxos nas

rotas mais frequentes, visto que este problema é de programação inteira mista e computacionalmente inviável para ser aplicado às ruas de uma cidade inteira?

- Hipótese: transformar o problema original de otimização inteira mista em um problema de otimização combinatória e introduzir restrições práticas de estudos sobre tempos de semáforos que reduzam a complexidade do problema e o torne viável computacionalmente.

5. Desafio: como comprovar que este método ocasiona soluções otimizadas para fluxo?
   - Hipótese: utilizar ferramenta de simulações de tráfego de veículos, comparando os tempos dos semáforos obtidos com os de outra solução e analisar os tempos médios de deslocamentos.

Consideramos, portanto, a criação de um método para agrupar trajetórias em janela de largura variável de tempo, utilizando o critério de densidade das trajetórias por tempo como parâmetro, a principal contribuição desta tese.

Outras contribuições deste trabalho estão expressas na sequencia.

1. Extensão do método ST-Matching de mapeamento de trajetórias livre por ruas para inferir a densidade de trajetórias, por segmento de rua e por janela de tempo.
2. Extensão do método FlowScan para revelar as rotas frequentes, por janela de tempo.
3. Criação de um método para encontrar a configuração ideal de tempos de semáforo em rotas frequentes, a fim de maximizar o fluxo na forma de onda verde nos caminhos mais frequentes, maximizando o fluxo de tráfego geral.
4. Transformação do clássico modelo de otimização offline do programador de semáforos para produzir a onda verde para um problema do Job Shop Scheduler (JSSP).
5. Uso de uma técnica de otimização combinatória para gerar uma solução ideal para tempos de semáforos.
6. A aplicação de um método de simulação para provar que uma solução ideal foi gerada pelo abordagem descrita.

Eis, sequencialmente, as principais publicações obtidas com este trabalho.

1. DMM: A Distributed Map-matching algorithm using the MapReduce Paradigm (ALMEIDA *et al.*, 2016b). Trabalho publicado no Intelligent Transportation Systems - ITSC 2016, Qualis Computação A1.
2. GPS2GR: Optimized Urban Green Routes Based on GPS Trajectories (ALMEIDA *et al.*, 2017). Trabalho publicado no 8th ACM SIGSPATIAL Workshop on GeoStreaming - 2017, , Qualis Computação A1.
3. Optimization of urban semaphore times turning into JSSP, trabalho aceito para VLDB 2018 - 44th International Conference on Very Large Data Bases, no workshop Big Social Data and Urban Computing - BiDU, Qualis Computação A1.

Esta tese está organizada em seis capítulos.

1. Introdução, onde delineamos o problema o objetivo do trabalho, seus subproblemas e as hipóteses para solucionar cada um deles.
2. Fundamentação Teórica fizemos um resumo da literatura envolvida no tema estudado.
3. Trabalhos Relacionados contém um resumo das abordagens semelhantes para resolver o mesmo problema e uma análise comparativa.
4. GPS2GR, explanamos nossa abordagem, para resolver cada subproblema, os algoritmos, definições matemáticas utilizadas e como os experimentos foram elaborados e executados.
5. Resultados, fizemos uma análise quantitativa e qualitativa dos resultados dos experimentos, bem como explicações sobre cenários de comparações.
6. Conclusões e Trabalhos Futuros contém análise crítica dos resultados obtidos nesta tese, bem como eventuais desdobramentos que podem ser abordados, partindo desta investigação de tese acadêmica.

2 FUNDAMENTAÇÃO TEÓRICA

Procedemos aqui, a um resumo dos conceitos definidos nos textos, artigos, livros e periódicos citados nas referências, para fundamentar a elaboração desta tese.

2.1 Definições preliminares

Definição 1 ***(Ponto de Trajetória)*** *É o elemento básico de registro de uma trajetória, comumente é obtido via medição de GPS, neste caso cada ponto é uma quadra formada pelos atributos id, timestamp, latitude e longitude, onde id representa um número de sequencia dos ponto, timestamp, a registro de data e hora da medição do GPS, latitude, a medida de distância horizontal em relação ao meridiano de Greenwich, e longitude, mediada de distância vertical em relação ao equador, podemos representar o ponto de trajetória como,* $\mathcal{P} = (id, t, h, v)$

Definição 2 ***(Trajetória de GPS)*** *Uma trajetória* $\mathcal{T}$ *é uma sequência de pontos de trajetórias, definimos portanto uma trajetória* $\mathcal{T}$*, como sendo uma sequência ordenada* $\mathcal{T} = (P_1, P_2, ..., P_n)$*, onde n é o número de pontos de uma trajetória.*

Definição 3 ***(Janela de Tempo)*** *É um intervalo fechado dentro do domínio numérico das horas de um dia que agrupa todas as trajetórias cujos os timestamp dos pontos estão dentro deste intervalo. Definimos portanto uma janela* $\mathcal{J}$*, como sendo* $\mathcal{J} = [t_{inicio}, t_{fim}]$*.*

Definição 4 ***(Dataset de Trajetórias)*** *É um conjunto de trajetórias independentes definidas por* $\mathcal{DT} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_k\}$*, onde k é o número de trajetórias do Dataset. O padrão comum de arquivo para Dataset de trajetórias é GPX.*

Definição 5 ***(Rede rodoviária)*** *Uma rede rodoviária é um grafo direcionado* $R(N, S)$*, onde* N *é um conjunto de nós que representam os cruzamentos de ruas e seus pontos terminais, com seus atributos são latitude e longitude, e* S *é um conjunto de arcos que representam segmentos de ruas, onde cada arco possui um atributo com o nome da rua. Assim* $N = \{n_1, n_2, ...n_y\}$*, onde y é o números de cruzamentos e pontos terminais do mapa e* $S = \{(n_i, n_j), ...(n_k, n_l)\}$*. O padrão de arquivo utilizado para os mapas é OSM.*

Definição 6 ***(Rede com Densidade)*** *É uma variante de uma Rede Rodoviária onde se adiciona a cada segmento um atributo que contém a contagem de trajetórias que passam*

por aquele segmento em um determinado tempo. Assim, os atributos dos segmentos S da rede ficam: nome da rua e densidade do fluxo.

Definição 7 ***(Rota Frequente)*** *É um subconjunto contíguo de segmentos de ruas que possuem uma frequência elevada de utilização, ou seja, subconjunto de uma Rede Rodoviária com caminhos que preservam um fluxo mínimo do início ao fim. Portanto* $RF \subset R$*, ou seja,* $RF(N,SF)$*, onde* $SF = \{(n_i,n_j),(n_j,n_K),...\}$*, e o atributo de densidade de cada segmento se mantém acima de um valor mínimo.*

Definição 8 ***(Comboio)*** *Comboio virtual de veículos, representado por uma região da via, que se desloca em uma velocidade média, onde os veículos que estiverem neste espaço e se deslocarem na mesma direção e velocidade do comboio, pertencem a este comboio até o fim da rota. O veículos que permanecerem no comboio terão o benefício na priorização e sincronização dos semáforos, facilitando seu deslocamento. Os veículos que saírem do comboio, ou porque saíram da rota, ou por trafegarem numa velocidade muito acima ou abaixo da velocidade do comboio, podem ficar bloqueado em algum semáforo com luz vermelha.*

Definição 9 ***(Modelo Webster)*** *É um estudo empírico sobre os tempos de semáforos que gerou equações e tabelas que definem um modelo genérico de relação entre os tempos de cada fase dos semáforos (WEBSTER; COBBE, 1966).*

Definição 10 ***(Onda Verde)*** *É a sincronização no espaço-tempo dos semáforos que otimiza o fluxo de veículos em uma via.*

Definição 11 ***(Rota Verde)*** *É a Onda Verde aplicada uma rota frequente, ou seja, sincronização dos semáforos ao longo de uma rota frequente, independente da ruas que compõe esta rota.*

2.2 O problema da ”discretização” de séries temporais em trajetórias

A ”discretização” de séries temporais de trajetórias visa revelar janelas de tempo, *clusters*, que agrupem as trajetórias, analisando apenas a dimensão tempo *timestamp* das trajetórias de modo que o número de clusters e os limites dos clusters representem da melhor maneira a variação da densidade de trajetórias no tempo.

Na figura 6, podemos ter uma noção do desafio de ”discretizar” uma série temporal de trajetórias cujo objetivo é identificar os pontos que demarcam as mudanças de janelas de tempo. Por princípio, as trajetórias entre dois marcos possuem densidades semelhantes e podem ser tratadas em grupo. Portanto a ”discretização” se resume a encontrar estes marcos.

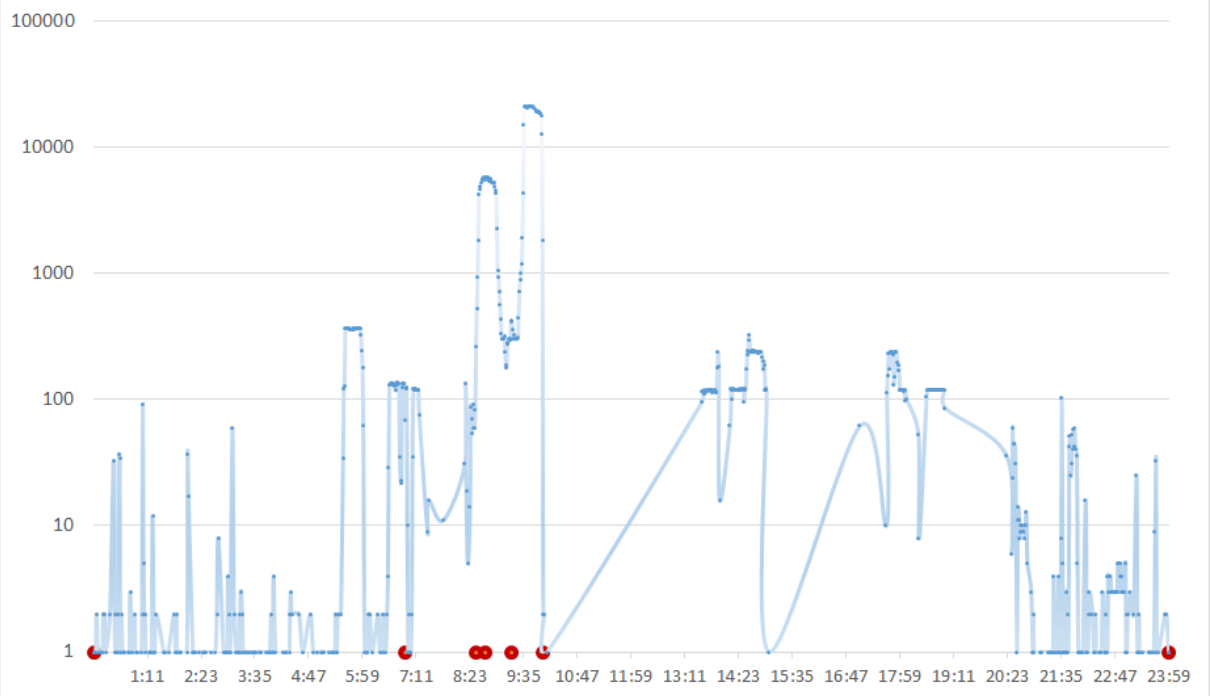


Figura 6 – Exemplo real da distribuição de densidade de trajetórias no decorrer de um dia

A descoberta do conhecimento em séries temporais em trajetórias, utiliza, em geral, a densidade de trajetórias por intervalo de tempo, como medida de agrupamento. Para permitir a descoberta de *cluster*, as representações simbólicas do tempo em horas, minutos e segundos são convertidas em escalas numéricas, permitindo a aplicação de vários métodos de discretização ou agrupamento. A escala numérica do tempo permite também o uso de um critério objetivo de avaliação da qualidade do método através da análise do erro final de cada solução de ”discretização” ou agrupamento. O uso de técnicas de agrupamento sobre séries temporais de trajetórias permite a criação de *cluster* de trajetórias, no âmbito de cada *cluster*, as trajetórias têm características semelhantes de densidade no tempo ou formam grupos espaçados, fato ao qual que podemos dar um tratamento único para todas as trajetórias de um *cluster* e, ao mesmo tempo, ter um tratamento diferente para cada *cluster*.

Neste contexto, a ”discretização” será utilizada para agrupar trajetórias com

densidade homogênea, facilitando, assim, a aplicação de técnica de descoberta de padrões em janelas de tempo onde as trajetórias possuem densidade similar (GIANNOTTI *et al.*, 2007).

Estudos utilizam histogramas para ”discretizar” séries temporais de trajetórias. Estes histogramas possuem ajuste automático da largura das faixas, resultando no menor erro na representação de dados. Oferecida uma quantidade de dados, gerar um histograma que representa a função de densidade de probabilidade (PDF), que permite extrair a maior quantidade possível de informações, não é uma tarefa simples.

Para o caso dos dados que seguem a distribuição de Poisson, o algoritmo CSTH (SHIMAZAKI; SHINOMOTO, 2007) para a seleção de tamanho do compartimento permite esse cálculo automático. O resultado desse algoritmo é a seleção, entre uma variedade de classes, da que melhor se ajusta ao conjunto de dados. CSTH tem por base um histograma espaçotemporal que utiliza o Erro Quadrado Integrado (MISE) como uma métrica para otimizar o ajustamento das classes que representam o conjunto de dados de trajetórias histograma.

Os benefícios da ”discretização” da série temporal para dados de trajetória estão reproduzidos na sequencia.

- Representação mais precisa das variações na densidade das trajetórias durante um período determinado, por exemplo um dia.
- Permitir uma representação mais precisa das rotas quentes e mudanças de tráfego, em decorrência das dinâmicas do tráfego da cidade ao longo do dia.

Outros métodos de ”discretização” foram propostos (SHIMAZAKI; SHINOMOTO, 2007), usando o ajuste do número de classes no histograma, minimizando o MISE. Todos os métodos baseados em histograma, no entanto, usam *clusters* de largura de faixas iguais, e isso não é muito adequado para trajetórias que variam a densidade frequentemente no dia. Portanto, buscamos um método cuja a largura das faixas possa variar de acordo com a densidade de trajetórias.

A análise de agrupamento e a análise de ”discretização” constituem segmentos importantes no terreno da mineração de dados, conforme anota (ZHENG, 2015). Além disso, estas técnicas agrupam objetos próximos ou relacionados uns com os outros. No agrupamento, os objetos que pertencem ao mesmo *cluster*, geralmente, têm algumas características comuns. O mesmo acontece com a análise de ”discretização”, onde os

atributos do objeto são atribuídos a um número finito de subintervalos de um intervalo específico (KAUFMAN; ROUSSEEUW, 2009). É necessário estabelecer uma medida heurística para decidir a qual grupo cada objeto pertence.

Para técnicas de agrupamento aplicadas a conjuntos de dados de pontos de trajetória, em que cada ponto, há uma coordenada de latitude e longitude; a distância euclidiana se encaixa bem. Por outro lado, para abordagens de "discretização" de tempo, precisamos estabelecer o número de valores categóricos, e como o método irá dividir o intervalo especificado.

Neste experimento, a medida heurística utilizada é a distância euclidiana. No entanto, esta medida de distância é aplicada apenas a dimensão de espaço dos pontos de trajetórias. Para nosso objetivo os pontos de dados são tratados como *timestamps* em um espaço unidimensional de tempo, consequentemente, os algoritmos agrupam pontos mais próximos em tempo entre si. Para abordagens de "discretização", o intervalo estabelecido varia de um dia.

De acordo com (TAN MICHAEL STEINBACH, 2005), os métodos de agrupamentos podem ser classificados como hierárquico ou particional. O agrupamento particional não tem sobreposição de grupos, enquanto o hierárquico o possui. Além disso, cada cluster ou grupo tem seu próprio tipo. De acordo com (TAN MICHAEL STEINBACH, 2005) existem quatro de tipos de algoritmos de agrupamento: baseados em protótipos, em gráfico e em densidade e de propriedades compartilhadas. Este trabalho foca em algoritmos que são arrimados em densidade como critério de agrupamento.

2.2.1 Técnicas de "discretização" e Agrupamento

Nesta seção, trazemos os principais métodos de "discretização" (BISHOP, 2006), bem como algoritmos de agrupamento aplicados com a finalidades de "discretização" da dimensão tempo de trajetórias, e como os métodos se comportam com dados de trajetórias. Embora as técnicas de "discretização" e agrupamento sejam diferentes, neste trabalho, utilizamos o resultado do agrupamento para gerar uma "discretização" do conjunto de trajetórias, utilizando apenas a dimensão de tempo das trajetórias. Como os métodos de agrupamento não revelem diretamente os domínios de cada *cluster*, é possível obter estes limites percorrendo os grupos e obtendo os limites inferior e superior de cada agrupamento (LINDEN, 2009).

Existem várias técnicas para ”discretização” e agrupamento de conjuntos de dados, neste ensaio, comparamos os resultados da aplicação dos algoritmos clássicos de ”discretização” com algoritmos de agrupamento para ”discretização” de dados de trajetórias. Os algoritmos de agrupamento podem ter bons resultados, uma vez que se ajustam melhor às variações de densidade de trajetórias ao longo do dia, pois as janelas possuem larguras variadas. A razão disso é que eles são mais precisos em representar a variação da densidade das trajetórias durante o dia. Além disso, essas técnicas podem lidar melhor com as mudanças de tráfego em razão das diversas dinâmicas da cidade no curso diário.

2.2.1.1 Redução de escala

A redução de escala é um método de ”discretização” que usa uma abordagem não supervisionada, ou seja, esta técnica não tem conhecimento das classes de cada conjunto de dados em processamento. Além disso, a redução de escala funciona dividindo o intervalo numérico em um número igual de intervalos (WITTEN, 2005). Portanto, todos os intervalos gerados pelo método de redução de escala possuem igual largura das faixas. Nesta investigação, a dimensão tempo está dentro do domínio de 0 a 24 horas, foi dividida em intervalos por hora ou minuto. Por exemplo, um intervalo de uma divisão diária por horas nos dará 24 intervalos. Os pontos de dados em processamento são atribuídos à respectiva ligação de intervalo de uma hora. O mesmo acontece quando dividimos o intervalo por minutos. Por exemplo, numa divisão do dia por hora nos resultará em 24 intervalos. Os pontos de trajetórias em processamento são atribuídos para o respectivo intervalo de uma hora, eventualmente trajetória cujos pontos pertençam a mais de um intervalo, então, foram consideradas pertencentes a ambos os intervalos. Um processo semelhante acontece quando dividimos o intervalo de um dia por minuto, ensejando portanto, 1440 intervalos num dia.

2.2.1.2 Histograma

Histograma é um método de ”discretização” de distribuições de frequências usadas em conjuntos de dados. Além disso, esta técnica usa a fórmula de Sturges (STURGES, 1926) para descobrir o número de intervalos com base na quantidade de pontos do conjunto de trajetórias, conforme descrito na fórmula 2.1, onde n é o tamanho da amostra e k o número de classes. Esse método é baseado em uma aproximação da

distribuição binomial, onde cada elemento pode pertencer ou não a uma classe. Neste trabalho, aplicamos diretamente esse método em nossos conjuntos de dados, como sendo o limiar para número de janelas de tempo para todos os métodos de discretização.

$$k = 1 + 3,322(\log_{10} n) \tag{2.1}$$

2.2.1.3 K-means

K-means é um algoritmo de agrupamento que possui um esquema baseado em protótipo. Este tipo de esquema funciona agrupando objetos semelhantes a um objeto-modelo, comumente chamado de centroide. K-means é um método supervisionado de agrupamento onde é necessário informar, *a priori*, o número de classes k.

Os fundamentos do algoritmo podem ser explicados como mostrados em (TAN MICHAEL STEINBACH, 2005). Tomamos aleatoriamente, os centroides iniciais K. Em seguida, cada ponto do conjunto de dados é atribuído a um *cluster* que tenha o centroide mais próximo, utilizando uma medida euclidiana. Então os centroides são recalculados com base nos pontos de cada *cluster*. O processo se reinicia interativamente até que os centroides não mudam mais.

A ideia de proximidade entre um ponto e seu centroide pode ser pensada como uma métrica que estabelece a distância entre dois pontos. Neste experimento, usamos a distância euclidiana, na dimensão do tempo já convertida para escala numérica, como medida para comparar a similaridade entre pontos.

Além disso, o último passo do algoritmo precisa de uma medida de qualidade. De acordo com (JAIN, 2010), a soma do erro quadrado é usada para medir a qualidade de um cluster e o objetivo é minimizá-la. Portanto, para recalcular os centroides, escolhemos a menor soma de erro quadrado entre os clusters gerados, e os novos modelos de objeto serão os seus centroides atuais.

Nesta tese, aplicaremos o k-means, variando o número de classes de dois até o limite da formula de Sturges, verificando a variação do erro SSE entre cada interação do k-means. O k-means será aplicado apenas a dimensão tempo convertida para escala numérica de 0 a 24, semelhante ao exemplo na figura 7. Portanto, a distância euclidiana é simplesmente a diferença absoluta entre o tempo de dois pontos de trajetórias. Exemplo p_1

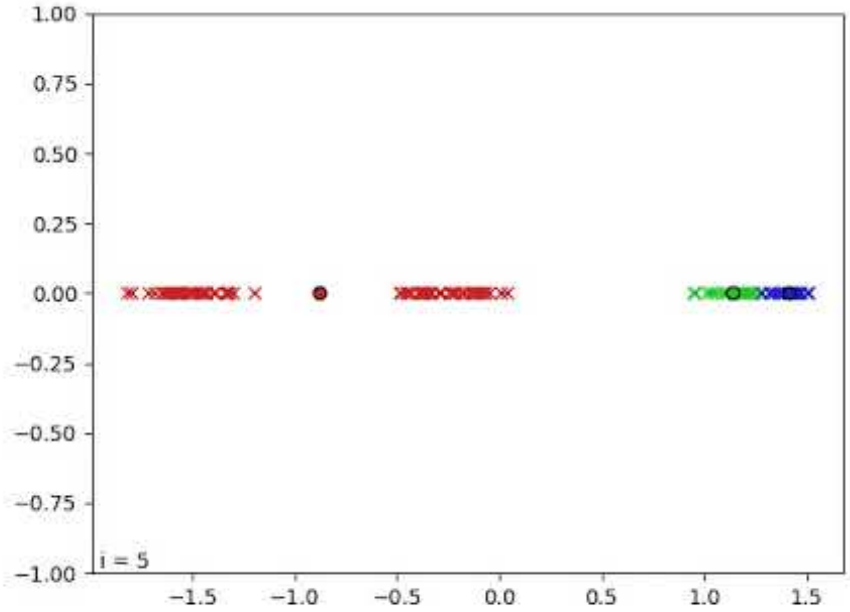


Figura 7 – Exemplo de aplicação do k-means em uma dimensão

= 2,2345 e ponto p_2=3,1223 a distância entre os dois pontos é $|p_1 - p_2| = |2,2345 - 3,1223|$ $= 0,8878$

2.2.1.4 DBSCAN

Os algoritmos de "clustering" espacial DBSCAN (ESTER *et al.*, 1996) é baseado em análise de densidade dos pontos do conjunto de entrada, dado um conjunto de pontos em algum espaço, o algoritmo agrupa pontos que estão perto, marcando como outliers pontos que ficam sozinhos em regiões de baixa densidade (cujos vizinhos mais próximos estão muito distantes)(ESTER H. KRIEGEL, 1996).

Além disso, os elementos de cada agrupamento podem ser classificados como: pontos principais, pontos de borda e pontos de ruído (TAN MICHAEL STEINBACH, 2005), conforme podemos observar na figura 8. Os pontos principais estão dentro de algum agrupamento e todos os outros pontos podem ser alcançados a partir dele. Se um ponto não pode ser alcançado de qualquer outro ponto, neste caso, é considerado um ponto de ruído. Finalmente, os pontos de fronteira estão na extremidade de um agrupamento, embora, eles ainda pertençam ao grupo.

Conforme mencionado, existem dois invariantes, ou parâmetros de entrada, necessários para que esse algoritmo funcione, o número mínimo de pontos por agrupamento

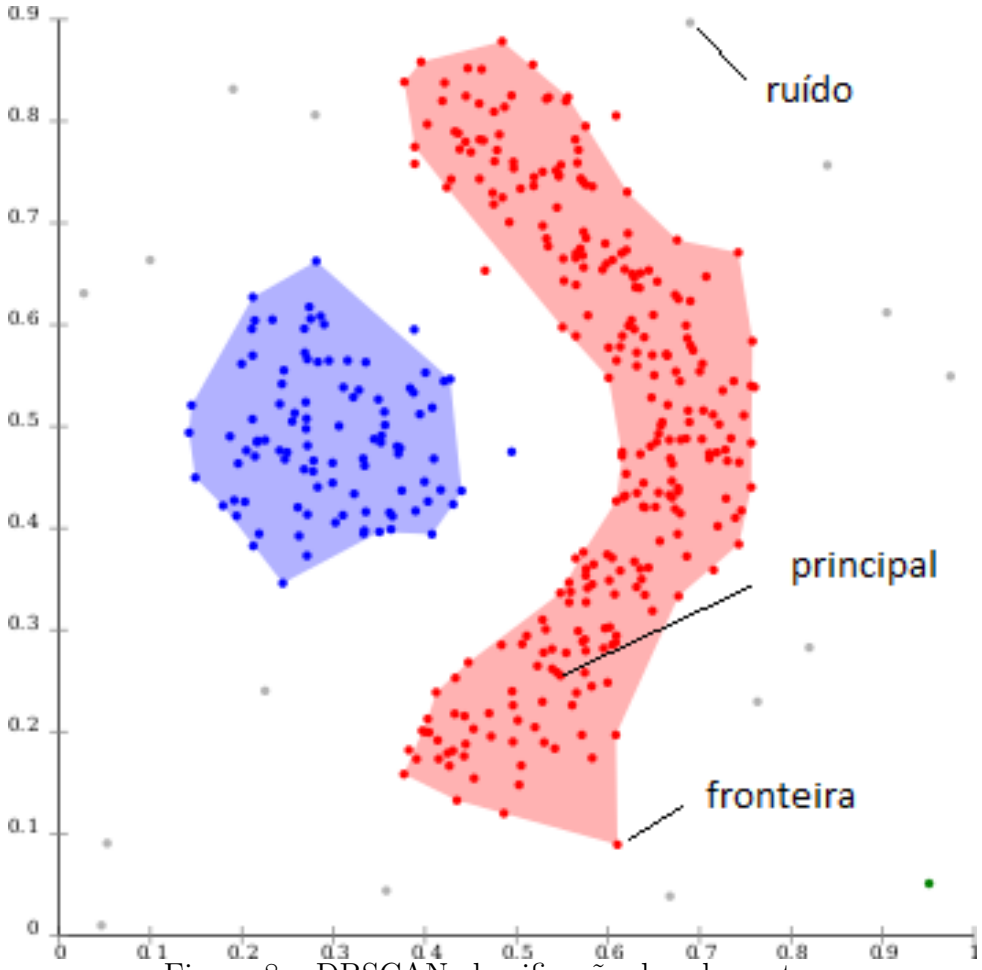


Figura 8 – DBSCAN classificação dos elementos

(MinPoints) e uma distância de máximo Eps. De acordo com (ESTER H. KRIEGEL, 1996), a distância máxima eps de um ponto p pode ser definida da seguinte maneira:

$$Neps(p) = \{q \in D \mid dist(p,q) \leq Eps\}, \tag{2.2}$$

onde os pontos de D podem pertencer ao grupo de p. O dinamismo do DBSCAN faz com que os pontos mudem de agrupamento a cada iteração. Por exemplo, um ponto pode ser considerado um ponto de borda para algum agrupamento, no entanto, em outra iteração, ele pode se tornar um ponto central.

Uma das etapas principais do DBSCAN é a seleção de seus parâmetros. Nesta tese, estamos trabalhando com uma série temporal proveniente de dados de trajetórias, a dimensão de tempo foi transformada em numéria e, portanto, está dentro de um intervalo de [0, 24]. Para encontrar bons parâmetros, variamos o Eps durante longos experimentos até que a soma do erro quadrado produzido se estabilize. O mesmo procedimento foi feito para o parâmetro $MinPoints$. Empiricamente, encontramos o fato de que trajetórias distantes de 43 minutos no tempo podem ser grupadas com menor erro. Em escala

numérica, equivale a $eps = 0{,}03$. De igual modo, com nosso conjunto de trajetórias, o menor erro foi obtido com numero de pontos de trajetórias desde 100, dai utilizamos $MinPoints = 100$.

2.3 O problema da junção espacial (Map-matching)

Métodos locais ou incrementais tentam encontrar a correspondência local de geometrias (GREENFELD, 2002). Os métodos incrementais usam duas medidas de similaridade para avaliar as bordas candidatas, uma para similaridade a distância e outra para a similaridade de orientação. A medida de similaridade combinada é calculada como a soma das pontuações individuais. A complexidade do tempo é $O(n)$, uma vez que encontramos bordas adjacentes para cada amostra, onde n é o número de pontos GPS a serem correspondidos. O método do recorte adaptável usa o algoritmo Dijkstra (FUHAO; JIPING, 2009) para estabelecer o caminho mais curto no gráfico de espaço livre local. Ele é executado em $O(mn \log m)$, onde m e n são, respectivamente, o número de bordas na rede rodoviária e de pontos de GPS.

Os algoritmos ST-Matching (ZHENG *et al.*, 2010) e UrbMatch (YING *et al.*, 2014) trabalham com uma matriz de probabilidade das rotas viáveis, escolhendo ao final a rota de maior pontuação. Nestes métodos, cada ponto de trajetória é uma posição que deve ser mapeada para um ponto correspondente na rede rodoviária. Considerando erros de precisão do GPS, o ponto mais próximo não é necessariamente o correto. Todo caminho deve ser considerado para evitar uma correspondência potencialmente errada. Em vez de mapear cada ponto para o nó mais próximo em um mapa digital (um método conhecido como correspondência ponto a ponto), um conjunto de caminhos possíveis é mantido e o caminho mais provável é escolhido com base na probabilidade. Isso é chamado de técnica de múltiplas hipóteses (MTH), usado em algumas estratégias de correspondência de mapas (SCHUESSLER; AXHAUSEN, 2009; PYO *et al.*, 2001).

Inicialmente, cada ponto de trajetória deve ser mapeado para seu conjunto de pontos candidatos, que são suas projeções em segmentos de ruas próximos. Isto envolve encontrar os vértices n mais próximos ao referido ponto de trajetória na rede viária, pois quaisquer pontos próximos são possíveis correspondências e devem ser levados em conta. Para uma rede rodoviária muito grande e/ou um grande número de pontos de trajetória, esta tarefa pode ser computacionalmente cara. Em seguida, encontramos os

pontos candidatos para cada ponto de trajetória, mapeando-os para sua projeção de segmento de linha, calculada como a distância entre um ponto de um segmento de rua.

Depois que os pontos candidatos foram recuperados, deparamos várias possibilidades de caminho para a solução. De qualquer ponto candidato $c_{i,j}$ (representando o j-ésimo ponto candidato no timestamp i), é possível mover para qualquer ponto candidato $c_{i+1,k}$. Para determinar o caminho mais provável percorrido por uma trajetória, devemos adicionar probabilidades ao nosso modelo.

Para representar melhor este modelo, é usado um gráfico candidato (figura 27) com todos os caminhos possíveis percorridos por uma trajetória. Neste gráfico, cada ponto candidato é um vértice e cada par de pontos candidatos consecutivos de timestamp forma uma aresta. Cada vértice tem uma pontuação associada, relacionada à probabilidade de uma trajetória estar realmente naquele ponto candidato. Ver algoritmo 1.

Algoritmo 1: ST-Matching

Input: Road network G
Input: trajectories T:$p_1 \rightarrow p_2 \rightarrow ... \rightarrow p_n$
Output: the matched sequence P:$c_1^{j1} \rightarrow c_2^{j2} \rightarrow ... \rightarrow c_n^{jn}$ in G

```
1 Initialize lList as an empty list;
  /* a list of candidate points                                   */
2 G' ← {}
3 for i = 1 to n do
4 |   s = GetCandidates(p_i, G, r);
  |   /* candidates within radius r                               */
5 |   lList.add(s);
6 end
7 G'_t = ConstructGraph(tList);
  /* constructs graph G'_t                                        */
8 return FindMatchedSequence(G'_t);
```

Do mesmo modo, cada borda possui um escore associado a ela, relacionado à probabilidade de uma trajetória que vai do seu vértice fonte ao seu vértice-alvo. Essas probabilidades são calculadas de acordo com o algoritmo ST-Matching, utilizando as coordenadas geográficas e calculando a distância euclidiana entre os pontos; o algoritmo ST-Matching original ignora completamente o aspecto temporal (timestamp) dos pontos das trajetórias.

As distâncias mais curtas entre vértices próximos na rede rodoviária precisam ser computadas como parte do cálculo da probabilidade de transição. Como o resultado

dessa fase é fixo para a mesma rede viária, e seu cálculo pode ser bastante custoso, dependendo da dimensão da rede, essa etapa é processada antecipadamente e armazenada em um sistema de arquivos distribuído, para posterior utilização.

Depois que o gráfico dos caminhos candidatos é construído, é possível percorrê-lo para encontrar o caminho ideal, ou seja, a sequência de pontos candidatos com a maior pontuação combinada. A pontuação é calculada como a soma da probabilidade de observação do ponto candidato $c_{i,j}$ e a probabilidade de transmissão da borda de $c_{i,j}$ para $c_{\mathrm{i}\,+\,1,k}$ (para qualquer k). No final, o caminho com maior pontuação é selecionado e recuperado, conforme exemplo na figura 9.

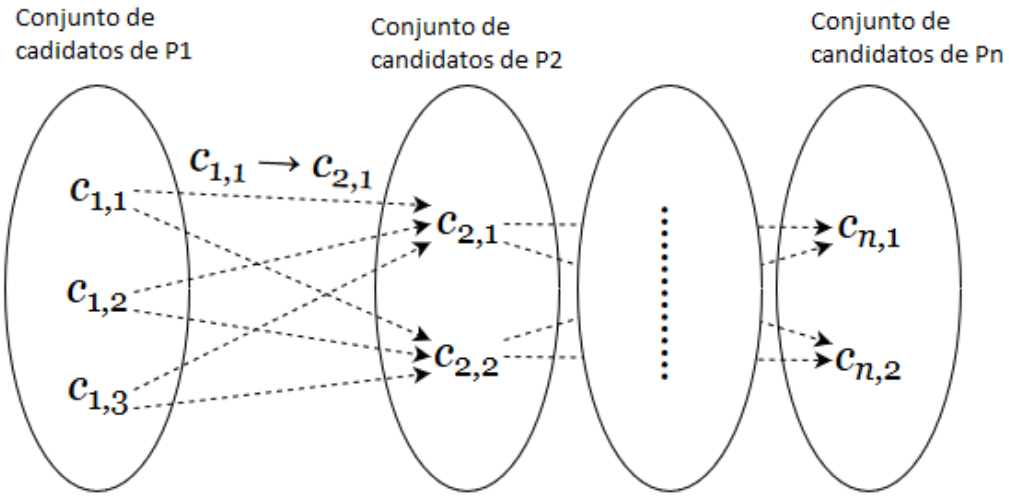
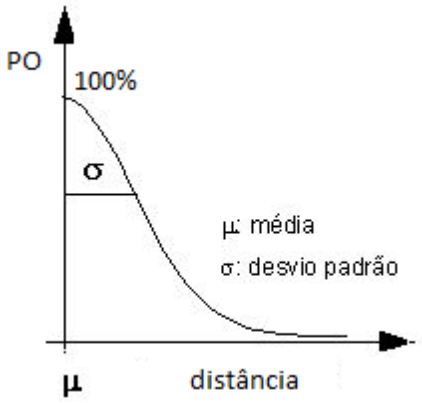


Figura 9 – Evolução do algoritmo ST-Matching para escolha dos caminhos candidatos

Para ponto da trajetória P_i, o algoritmo seleciona o conjunto de pontos de segmento de rua candidatos C_i^k. Por padrão o ST-Matching seleciona até cinco pontos candidatos. Para cada ponto candidato C_i^k, será calculada uma probabilidade de observação, é definida como uma distribuição normal sobre a distância euclidiana do ponto candidato C_i^k e o ponto da trajetória P_i. Nessa distribuição, a média μ será 0, ou seja, se a distância for zero, a probabilidade será máxima, 100%, e o desvio-padrão σ é 20, ou seja, para distâncias maiores do que 20 metros a probabilidade será mínima. Entre dois pontos candidatos de conjuntos distintos $C_{i,j}$ e $C_{k,l}$, foi definida a probabilidade de transmissão como a razão entre a distância euclidiana e o número de segmentos de rua do menor caminho entre $C_{i,j}$ e $C_{k,l}$. Finalmente, o escore final de cada caminho candidato é definido como a multiplicação de todas as probabilidade de observação e de transmissão, de cada caminho candidato.

Em resumo:

- $P_i \Longleftrightarrow (C_{i,1}, C_{i,2}, ..., C_{i,k})$, default k=5
- Probabilidade de observação $PO(C_{i,1}) = Normal(\mu = 0, \sigma = 20, dist(C_{i,1}, P_i))$



- Probabilidade de transmissão $PT(C_{i,j}, C_{k,l}) = \frac{dist(C_{i,j}, C_{k,l})}{|shortpath(C_{i,j}, C_{k,l})|}$
- $Score(C_{1,i}, C_{2,j}, ..., C_{n,x}) = \prod_{k=1}^{n} PO(C_{k,x}) * PT(C_{k,x}, C_{k+1,x})$

Em (ZHENG *et al.*, 2010) é proposto um método de correspondência baseado em segmento para atribuir valores de confiança a diferentes pontos de amostragem. Ele coincide primeiro com segmentos de alta confiança e, em seguida, combina segmentos de baixa confiança usando fronteiras anteriormente mescladas. Em geral, quando uma nova posição é combinada, um método local ou incremental considera apenas uma pequena porção da trajetória próxima à posição. Este método funciona rápido e se desenvolve bem quando a frequência de amostragem é muito alta. À medida, porém, que a taxa de amostragem diminui, a probabilidade de se obter uma lacuna de trajetória aumenta, tornando-se proeminente e causando degradação de precisão significativa. Em contraste, considerando um aumento razoável na complexidade do tempo no algoritmo original, propomos uma alteração no algoritmo ST-Matching, tornando-o mais eficaz para a taxa de amostragem mais baixa e para grande volume de trajetórias. Propomos uma modificação na poda do algoritmo ST-Matching, fazendo uma poda preliminar no conjunto de ruas, considerando somente o universo de ruas próximas a trajetórias (CutOut) e fazendo a busca de candidatos utilizando um índice kd-tree (CutIn). Ver figura 10, além disso, utilizamos o algoritmo A^* para completar as trajetórias com falhas ou baixa amostragem, como descreveremos no próximo capítulo.

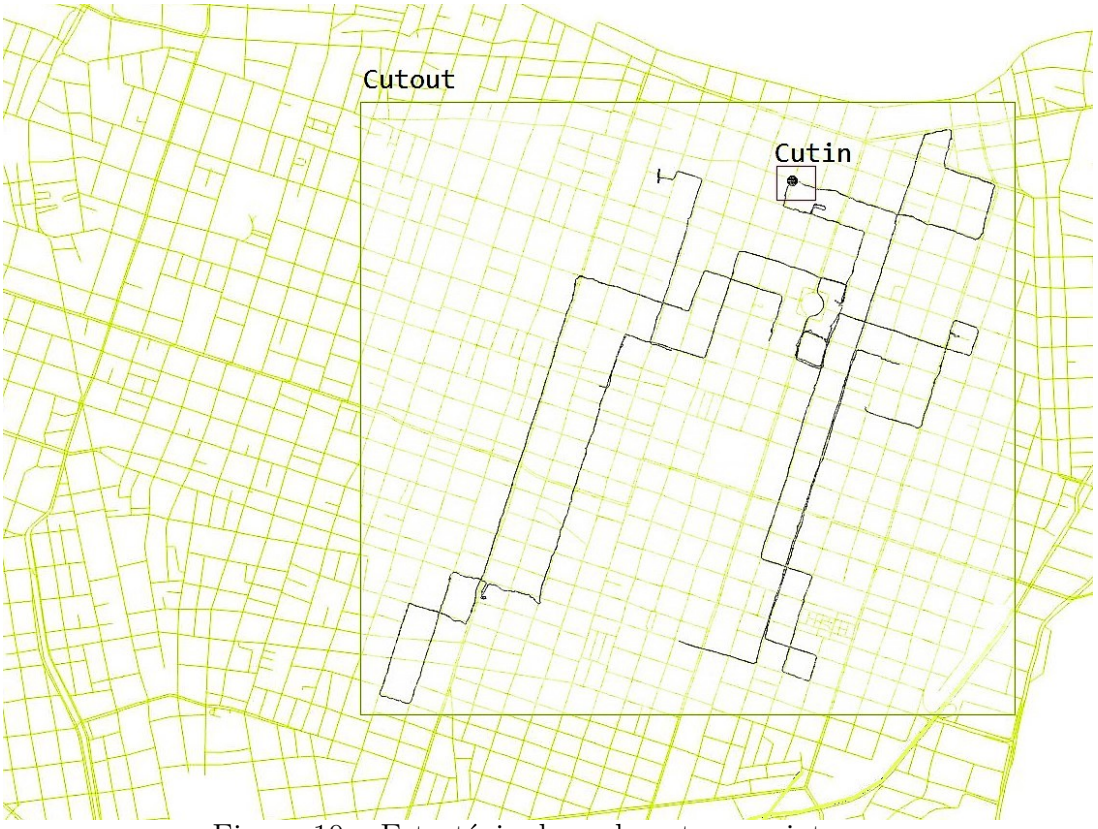


Figura 10 – Estratégia de poda externa e interna

2.4 O problema de descoberta de rotas frequentes

O problema de descobrir rotas frequentes foi examinado no trabalho de (LI *et al.*, 2007), para o caso em que os objetos estão limitados a se mover em uma rede conhecida. Uma rota frequente neste contexto é uma sequência de arestas, não necessariamente adjacentes, que compartilham determinada quantidade de tráfego.

Neste ensaio, abordamos o problema de descobrir rotas frequentes em uma rede rodoviária urbana. Uma rota frequente, portanto, revela um padrão geral de fluxo de tráfego. O conjunto de rotas frequentes oferece uma visão direta dos padrões de tráfego de uma cidade. Gestores de planejamento urbano podem usá-las para melhorar o fluxo de tráfego; donos de lojas e anunciantes podem aplicá-las para posicionar melhor seus empreendimentos; sistemas de segurança pública podem se usar delas para maximizar a cobertura de patrulhamento (WEI *et al.*, 2012). Nossa contribuição neste contexto é propor a criação de rotas frequentes por janela de tempo, considerando o fato de que uma cidade pode modificar seus padrões de tráfego no decorrer do dia.

Uma característica importante do tráfego no mundo real é a complexidade. Em vez de grupos naturais, os veículos viajam em velocidades e tempos diferentes, mesmo

quando estão na mesma rota. Por exemplo, em um bairro residencial, muitas pessoas saem para trabalhar de manhã e viajam para o seu local de trabalho, usando, aproximadamente, as mesmas rotas. Semáforos e outros obstáculos na trajetória podem uni-los ou separá-los em toda a rota.

Encontrar rotas frequentes (padrões de fluxo de tráfego) em uma rede rodoviária é um problema importante. Eles são benéficos para os planejadores da cidade, departamentos de polícia, promotores imobiliários e muitos outros. Conhecer as rotas frequentes permite à cidade direcionar melhor o tráfego ou analisar as causas do congestionamento. No passado, esse problema foi amplamente abordado com o conhecimento do domínio da cidade. Nos últimos anos, entretanto, informações detalhadas sobre veículos na rede rodoviária estão disponíveis. Com o desenvolvimento e adoção de GPS e outros sensores de localização, uma enorme quantidade de trajetórias de objetos móveis está sendo coletada e pode ser usada para encontrar rotas frequentes. Este é um problema desafiador em virtude da natureza complexa dos dados. Se os carros viajassem em *clusters* organizados, seria simples usar um algoritmo de *clustering* para encontrar as rotas frequentes. No mundo real, todavia, os objetos se movem de maneiras imprevisíveis. Variações na velocidade, no tempo, na rota e em outros fatores fazem com que eles viajem em grupos. Essas propriedades tornam o problema difícil para uma abordagem ingênua. Em (LI *et al.*, 2007) propõe o algoritmo FlowScan baseado em densidade. Em vez de agrupar os objetos em movimento, os segmentos de estrada são agrupados com base na densidade do tráfego comum que eles compartilham.

O FlowScan, portanto, utiliza a densidade de trajetórias por segmento de rua como um métrica de fluxo. Este algoritmo identifica onde nascem os fluxos, acompanha os caminhos que mantêm um mínimo de fluxo até que o fluxo se encerre.

Para entender o FlowScan, consideremos as definições de $NEps$, que representa o conjunto de caminhos vizinhos a partir de um segmento de rua r, cujo tamanho do caminho, ou seja, o número de segmentos é menor ou igual a constante eps. O início de uma rota frequente é cada segmento r de rua cuja a soma das densidades dos segmentos vizinhos, menos a densidade do próprio segmento r, é maior do que uma constante $minTraffic$. Por fim, o método define como critério de parada de uma rota frequente quando a condição de manutenção não for mais satisfeita.

Em resumo:

- Caminhos vizinhos $NEps(r) = \{s \in E | ForwardNumHops(r, s) \leq Eps\}, Eps \geq 0$
- Início de uma rota $|traffic(r) - \sum traffic(x)| \geq minTraffic, x \in \{NEps(r)\}$
- Condição de preservação de fluxo $|traffic(r) \cap traffic(s)| \geq minTraffic, s \in NEps(r)$

Algoritmo 2: FlowScan

Input: Road network G
Input: object trajectories T
Input: Eps
Input: minTraffic
Output: Hot routes R

```
1 Initialize R to {}
2 Let H be the set of hot houte starts in G according to T
3 for ever hot route start h in H do
4 |   r = new HotRoute initialized to <h>
5 |   Add ExtendHotRoute(r) to R
6 end
7 return R
```

2.5 O problema da onda verde

A mobilidade é uma das preocupações mais significativas nas principais cidades do mundo, em razão do crescimento da frota de veículos e da limitação de capacidade das vias urbanas. Podemos, então, afirmar que, na maioria das cidades o fluxo de veículos ultrapassa a capacidade das vias. O trabalho (BROCKFELD *et al.*, 2001) estudou a otimização dos semáforos em um modelo de autômato celular para o tráfego da cidade. Esse estudo demonstra o efeito benéfico da estratégia de controle de sinal no tráfego de veículos para aumentar a vazão com a mesma capacidades das vias. Além disso, os autores mostraram que o tráfego controlado por semáforos pode ser reduzido a um problema mais simples de uma matriz de células onde as ruas e suas interseções são tratadas igualmente, tudo baseado no modelo Biham–Middleton–Levine BML. O ensaio citado inspirou a propor a formação de comboios virtuais de veículos no lugar de células. Na nossa abordagem, os comboios são agrupamentos que se deslocam no espaço-tempo, enquanto no modelo (BROCKFELD *et al.*, 2001) as células são espaços fixos, que pode ou não conter veículos. Esse mesmo trabalho concluiu que, a partir de uma simples estratégia sincronizada, a capacidade da rede depende fortemente do ciclo de tempos dos semáforos. Já o estudo

(SASAKI; NAGATANI, 2003) investigou o fluxo de tráfego controlado por semáforos em rota única, usando o modelo de velocidade ideal. Nesse trabalho os autores diferem na relação entre a capacidade da estrada e o limite de saturação. Estudo de (NAGATANI, 2007) descreve o tráfego de veículos, por meio de uma sequência de semáforos em uma rodovia e ofereceu um modelo dinâmico em que um veículo é impedido de passar por outros veículos.

Esta tese considera o fluxo de veículos passando por uma série finita de semáforos. Cada veículo se move com velocidade média v, se o caminho não estiver bloqueado por outros veículos e não houver sinais de trânsito não sincronizados, e esta velocidade v será calculada com base nas trajetórias de GPS mapeadas por segmento de rua. Consideramos o tráfego de veículos na rede unidimensional. Nosso objetivo consiste em incentivar a formação de comboios de veículos mediante a orquestração de semáforos, onde os veículos tendem a viajar juntos, todos com a velocidade em torno da velocidade média da rota v. Na estratégia sincronizada, todos os semáforos de uma rota mudam sequencialmente de vermelho para verde e vice-versa em um período fixo. A soma dos tempos de cada semáforo é chamada de tempo de ciclo; o tempo de ciclo será ajustado entre o valor mínimos e máximo, permitindo que exista uma solução ótima para que isto ocorra em todas as rotas, visto que, quando se aumenta o tempo de ciclo, aumentam-se proporcionalmente os tempos de folga nos cruzamentos. Na figura 11, vemos um semáforo com tempos de ciclos sincronizado para cada fase.

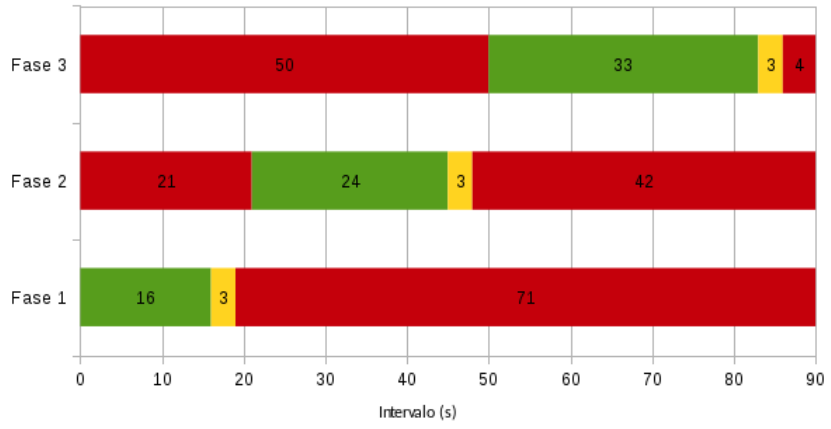


Figura 11 – Exemplo de ciclo de um semáforo com três fases

Na estratégia da onda verde, o semáforo muda seu estado com um certo atraso entre as fases do semáforo de duas interseções sucessivas. O tempo de atraso é chamado

de tempo de viagem. A mudança de semáforo se espalha para trás, como uma onda verde. Se um veículo chega a um semáforo com sinal verde, basta que se mantenha na velocidade média da via para que os próximos semáforos estejam verdes também. Se eventualmente um veículo perde a onda verde, ou acabou de entrar na rota e ficou bloqueado num sinal vermelho, assim que o semáforo ficar verde novamente ele estará em outra onda verde, ver figura 12.

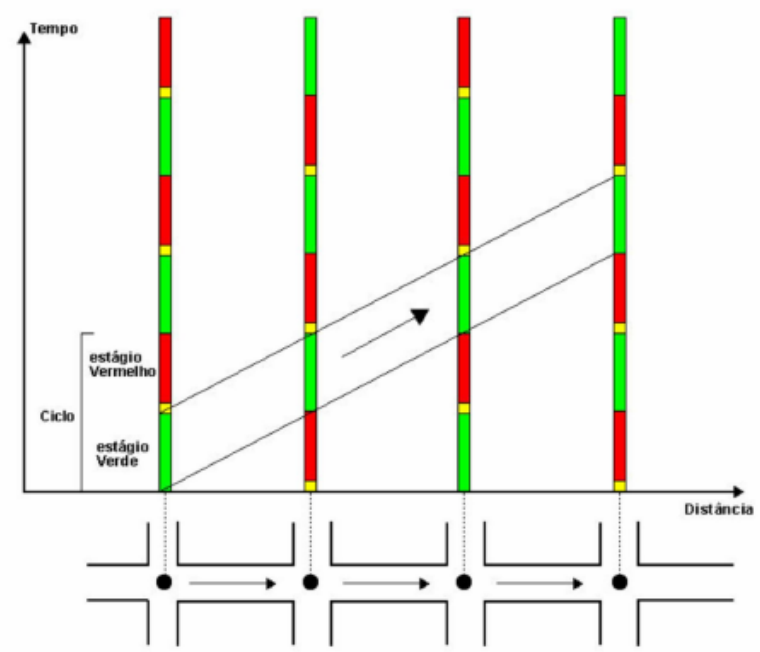


Figura 12 – Abordagem de propagação da janela de tempo do ciclo

Já na figura 13 podemos observar com maior clareza o tênue equilíbrio da onda verde em relação à velocidade dos veículos. Se o veículo andar numa velocidade muito superior ao que foi planejado para esta rota, ele estará à frente da onda verde no tempo e, portanto, poderá ficar bloqueado em um semáforo à frente. Caso contrário, se ele estiver numa velocidade muito inferior aquela destinada à rota, ele perderá a onda verde e ficará retido no próximo semáforo.

O problema clássico de otimização dos tempos dos semáforos (WARBERG *et al.*, 2008) para produzir um efeito de onda verde em um caminho é representado pelo

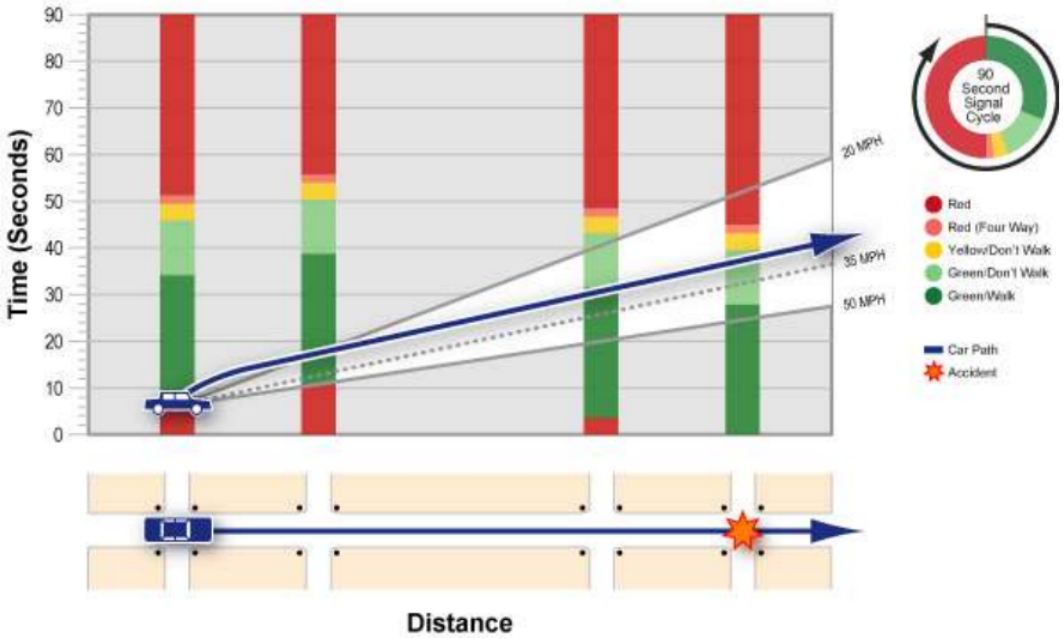


Figura 13 – Exemplo de um gráfico de caminhos de onda verde com a sincronização do espaço x tempo

seguinte modelo matemático:

$$
\begin{aligned}
\min \quad & TT(\Psi, q^*(\Psi)) = \sum_{l=1}^{L} q_l \cdot t_j(\Psi, q^*(\Psi)) \\
& subject \quad to: C_{min,n} \leq C_n \leq C_{max,n}, \\
& 0 \leq \theta_n \leq C_n - 1 \quad \forall_n, \\
& \phi_{min,np} \leq \phi_{np} \leq \phi_{max,np} \quad \forall_{n,p}, \\
& C_n = \sum_{p=1}^{P_n} (\phi_{np} + I_{np}) \quad \forall_n
\end{aligned}
\tag{2.3}
$$

Como o problema de otimizar as ondas verdes em vias públicas é de complexidade NP-Hard, as abordagens tradicionais utilizam métodos adaptativos (WARBERG *et al.*, 2008) para encontrar boas soluções, fazendo ajustes nos parâmetros mediante de avaliação e retroalimentação por meio de medições online, semelhantes aos métodos de processamento de sinal.

2.5.1 Sistemas de otimização de rotas em lote

Nesta seção, trazemos as tendências para sistemas de otimização off-line pesquisadas. Para que o modelo seja solucionável sem uma ferramenta de avaliação externa, como o TRANSYT ou SUMO, o modelo deve conter restrições funcionais que representam bem

$n \in \{1, ..., N\}$	Índices de intersecção
$p \in \{1, ..., P_n\}$	Índices de fases na interseção n
Ψ	Um conjunto de configurações de temporização de sinal
C_n	Tempo de ciclo para interseção n
θ_n	Compensação de interseção n
ϕ_{np}	Fase p do tempo verde para interseção n
I_{np}	Intervalo perdido de tempo após a fase p até o próximo cíclo
$l \in \{1, ..., P_n\}$	Índices de links
$q_l \in \mathbf{q}*(\Psi)$	Fluxo de link de equilíbrio do usuário, em função dos parâmetros de temporização do sinal
$t_l(\Psi, \mathbf{q}*(\Psi))$	Tempo de viagem no link l considerando os tempos de sinal e a resposta do usuário

Tabela 1 – Notação para o sinal de trânsito e modelo de rede

as restrições do tráfego, como, em um exemplo não permanecer verde em fases distintas do mesmo semáforo, implementar tempo de vermelho extra para evitar acidente, denominado vermelho global etc. Considerando um modelo de otimização típica envolvendo um tempo de ciclo comum, divisões e compensações verdes, está claro que, mesmo com alguma "discretização" de tempo, o espaço de busca de uma solução é muito amplo, especialmente quando se considera alguma modalidade de simulação para obter o valor objetivo. Os sistemas que operam dessa maneira são mais adequados para uso off-line, considerando que alguns tempos de execução relatados são tão altos quanto várias horas, mesmo para redes pequenas.

2.5.2 Job Shop Scheduller Solver

Um problema comum de programação é o Job Shop Scheduller, na qual diversas tarefas são processadas em várias máquinas. Cada trabalho consiste em uma sequência de tarefas, que devem ser executadas em determinada ordem, e cada tarefa deve ser processada em máquina específica. Por exemplo, o trabalho poderia ser a fabricação de um só item de consumo, como um automóvel.

O problema é programar as tarefas nas máquinas para minimizar a duração da programação - o tempo que leva desde que os trabalhos são iniciados pela primeira vez até que todos os trabalhos sejam concluídos.

Existem várias restrições para o problema da Job Shop Scheduler:

- nenhuma tarefa para um trabalho pode ser iniciada até que a tarefa anterior desse

trabalho seja concluída;

- uma máquina só pode trabalhar em uma tarefa por vez; e
- uma tarefa, uma vez iniciada, deve ser executada até a conclusão.

Abaixo está um exemplo simples de um problema de Job Shop Scheduler, em que cada tarefa é rotulada por um par de números (m, p), onde m é o número da máquina em que a tarefa deve ser processada e p é o tempo de processamento da tarefa, a quantidade de tempo que requer. A numeração de trabalhos e máquinas começa em 0.

- job 0 = [(0, 3), (1, 2), (2, 2)]
- job 1 = [(0, 2), (2, 1), (1, 4)]
- job 2 = [(1, 4), (2, 3)]

No exemplo, o trabalho 0 possui três tarefas. A primeira tarefa, (0, 3), deve ser processado na máquina 0 em 3 unidades de tempo. A segunda (1, 2) há de ser processado na máquina 1 em 2 unidades de tempo e assim por diante. Ao todo, existem oito tarefas.

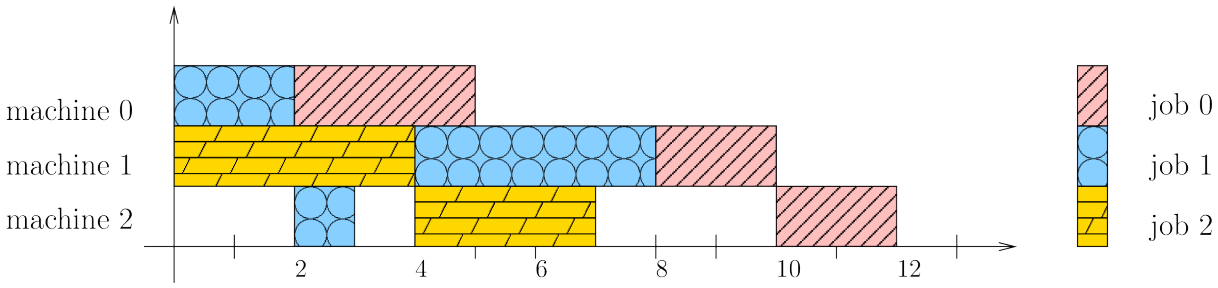


Figura 14 – Solução 1 - JSSP

Uma solução para o problema da job shop é a atribuição de uma hora de início para cada tarefa, que atende às restrições dadas acima. A figura 14 mostra uma solução possível para o problema:

Verificamos se as tarefas de cada trabalho estão agendadas em intervalos de tempo não sobrepostos, na ordem dada pelo problema. O tamanho dessa solução é 12, que é a primeira vez em que todos os três trabalhos são concluídos. Podemos observar na figura 15, porém, que existe uma solução melhor onde o *makespan* é 11.

Variáveis e restrições para o problema

Esta seção descreve como configurar as variáveis e restrições para o problema. Primeiro, deixe a tarefa (i, j) denotar a j-ésima tarefa na sequência do trabalho i. Por exemplo, tarefa (0, 2) indica a segunda tarefa do trabalho 0, que corresponde ao par (1, 2) na descrição do problema. Em seguida, define $t_{i,j}$ como a hora de início da tarefa (i,

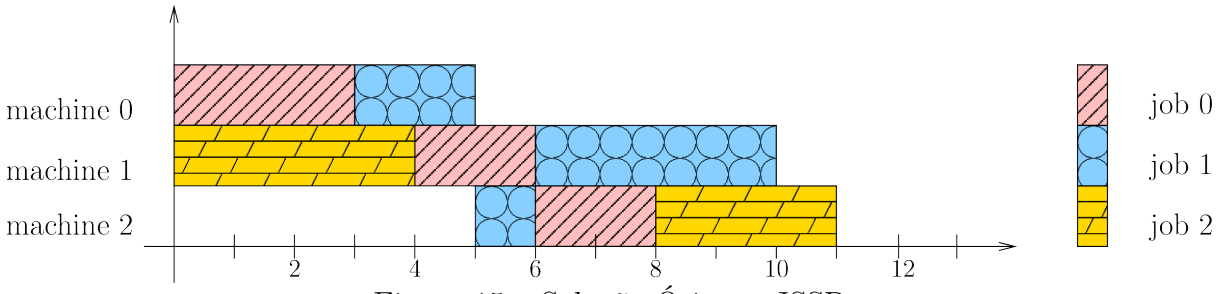


Figura 15 – Solução Ótima - JSSP

j). O $t_{i,j}$ são as variáveis do problema loja de trabalho. Encontrar uma solução envolve determinar valores para essas variáveis que atendem ao requisito do problema. t_i.

Existem dois tipos de restrições para o problema do Job Shop Scheduler:

- Restrições conjunturais - surgem da condição de que, para quaisquer duas tarefas consecutivas no mesmo trabalho, a primeira deve ser concluída antes que a segunda possa ser iniciada. Por exemplo, tarefas (0, 2) e (0, 3) são consecutivas para o trabalho 0. Como o tempo de processamento da tarefa (0, 2) é 2, o tempo de início da tarefa (0, 3) deve ser pelo menos 2 unidades de tempo após o horário de início da tarefa 2. Como resultado, obtém-se a seguinte restrição: $t_{0,2} + 2 \leq t_{0,3}$.
- Restrições disjuntivas - elas surgem da restrição de que uma máquina não pode trabalhar em duas tarefas ao mesmo tempo. Por exemplo, a tarefas (0, 2) e (2, 1) são processadas na máquina 1. Como os tempos de processamento são 2 e 4, respectivamente, uma das seguintes restrições deve conter: $t_{0,2} + 2 \leq t_{2,1}$ (se a tarefa (0, 2) estiver programada antes da tarefa (2, 1)) ou $t_{2,1} + 4 \leq t_{0,2}$ (se a tarefa (2, 1) estiver programada antes da tarefa (0, 2)).

O gráfico 16 ilustra as variáveis e restrições no problema do Job Shop Scheduler.

Os nós no gráfico representam as tarefas. As linhas sólidas com setas, que conectam tarefas consecutivas para o mesmo trabalho, correspondem a restrições conjuntivas. As setas apontam na ordem em que as tarefas devem ser processadas.

As linhas tracejadas, que conectam tarefas para a mesma máquina, correspondem a restrições disjuntivas. Essas linhas não possuem setas, porque a ordem dessas tarefas não é dada. De fato, atribuir setas a linhas - o que é o mesmo que atribuir a ordem das tarefas em cada máquina - é essencial para resolver o problema.

Em resumo neste capítulo abordamos que:

- a ”discretização” do tempo é importante para reduzir a complexidade do problema sem perder a especificidade de cada momento da cidade;

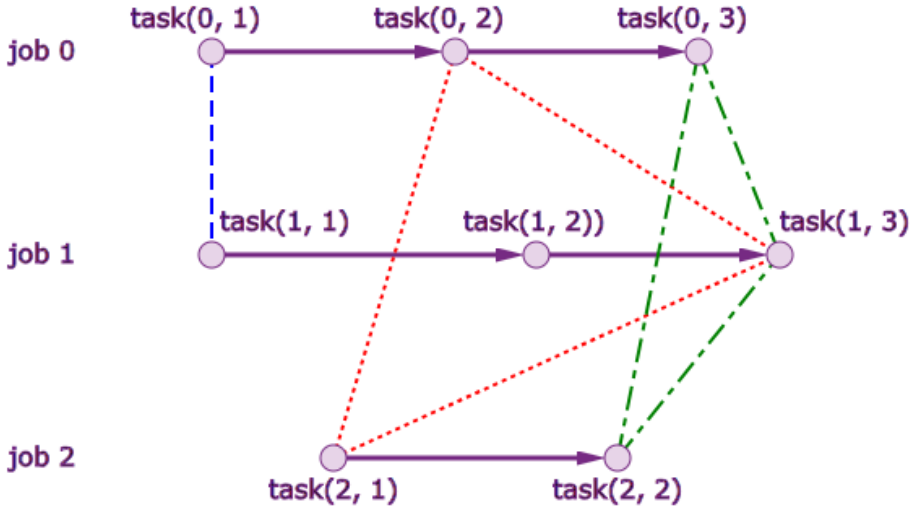


Figura 16 – Restrições do problema Job Shop Scheduler

- os algoritmos de map-matching são confiáveis para se inferir o numero de trajetórias por ruas;
- o algoritmo FlowScan por calcular com precisão os fluxo mais frequentes utilizando o critério de densidade de trajetórias, pela rua;
- o modelo de otimização para obter a onda verde é um grande desafio computacional; e
- um solver para o problema de otimização Job Shop Scheduller.

3 TRABALHOS RELACIONADOS

Sistemas adaptativos de controle de tráfego visam a coordenação de interseções via semáforos para otimizar algum índice de desempenho. Por exemplo, o atraso médio ou o número de paradas (ou uma combinação), ou para reduzir a necessidade de supervisão constante e ajuste de interseções.

Os sistema adaptativos realizam ajustes dinâmicos dos tempos de ciclo, sequências de fase e divisões verdes de acordo com o tráfego detectado, bem como do tráfego previsto, reagindo, assim, dinamicamente a esses aspectos do tráfego, que não podem ser capturados pelas rotinas de otimização estática usadas para gerar planos de hora do dia.

Os sistemas adaptativos, porém, fazem isso às cegas num processo de simulação que busca uma solução sempre melhor do que a anterior, não garantindo, assim, que se atinja um ótimo global.

Neste capítulo, descreveremos os principais trabalhos relacionados ao que é proposto nesta tese, fazendo um resumo técnico ao final, apontando detalhes das abordagens de cada um dos modelos.

3.1 RHODES

A abordagem do RHODES (MIRCHANDANI; WANG, 2005) para a otimização de sinal de tráfego é hierárquica, com três camadas de detalhes, como mostra a Figura 17.

A camada de rede executa o carregamento dinâmico de rede, o que envolve a observação de alterações nos dados de fluxo agregado de toda a rede em razão de variações nas matrizes de origem e destinos. Esta camada fornece estimativas de fluxos de *link* para o nível médio em números aproximados, por exemplo, veículos por hora. A camada intermediária de setor considera setores da rede, por exemplo, uma via arterial. Essa camada de controle de fluxo de rede funciona no nível de detalhe dos pelotões e das velocidades médias. O tempo verde é alocado a fases para acomodar os movimentos dos pelotões e, assim, a coordenação das interseções é feita nesse nível. No nível mais baixo, está o controle de interseção, onde os veículos são manipulados individualmente (uma camada microscópica). Aqui, os tempos verdes e a ordem de fases sugeridos pela camada intermediária são bem ajustados.

Um sistema de controle de tráfego adaptativo deve operar rapidamente para

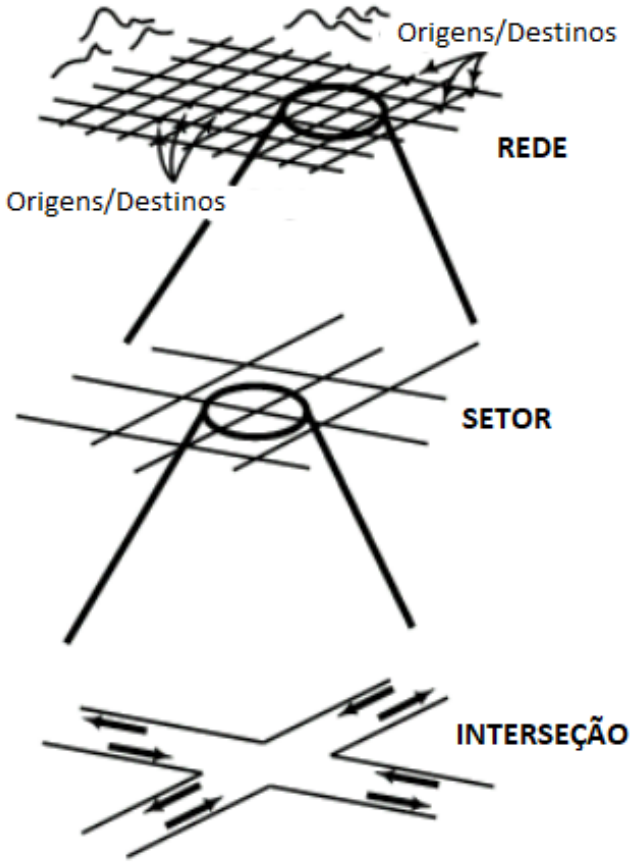


Figura 17 – Os três níveis de detalhe: rede, setor e interseção do RHODES

adaptar os sinais ao tráfego em tempo real. A plataforma RHODES tem boas oportunidades de decomposição e é conectável, ou seja, o nível superior é uma caixa preta alimentando o nível mais baixo com previsões e otimizações. Até a escrita desta tese, o nível superior do RHODES não havia sido implementado e, portanto, apenas os níveis intermediário e inferior são descritos aqui.

Na figura 18, verificamos o modelo adaptativo que, por via de retroalimentação colhida por sensores, procura melhorar a solução anterior, implementado na camada de interseção.

O RHODES foi implementado e avaliado mediante a simulação com CORSIM (YANG; ZHOU, 2004) como parte da avaliação para a inclusão da FHWA[1] no RT-TRACS. O RT-TRACS é um esforço para escolher e padronizar um sistema de otimização de sinal de tráfego de desempenho de pico para redes de tráfego dos Estados Unidos. A simulação é feita para uma interseção arterial de 9 interseções com aumento constante e, em seguida,

[1] Federal Highway Administration (FHWA) é uma divisão do Departamento de Transportes dos Estados Unidos especializada em transporte rodoviário.

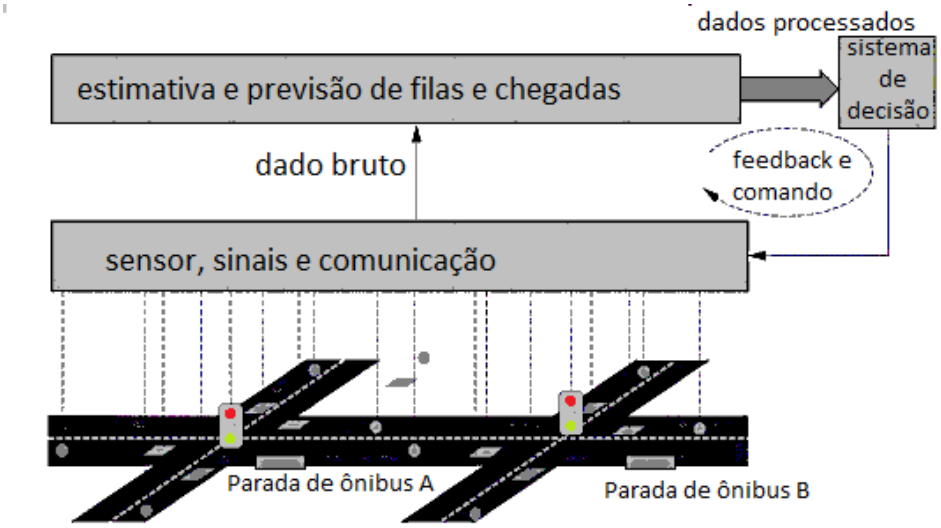


Figura 18 – Um diagrama simplificado da operação de RHODES.

diminuição do tráfego por um período de 2 horas. Este é um caso de teste FHWA e o sistema de controle de tráfego de linha de base é o controle semi-atuado[2] baseado nos resultados das ferramentas on-line, incluindo TRANSYT (ROBERTSON, 1969) e PASSER, que representam o melhor que se pode fazer de uma abordagem on-line e podem ser consideradas um concorrente difícil. O teste mostra que o RHODES é mais capaz de explorar a capacidade do sistema arterial. Enquanto não houver congestionamento, a taxa de transferência corresponderá à demanda e, na comparação, o RHODES pode simplesmente levar mais carga antes de alvo de congestionamento. Os sistemas adaptativos reais devem se sobressair no caso de baixa demanda, já que o excesso de capacidade permitirá que o RHODES, de forma teórica, atenda a cada veículo. O efeito, comparado ao controle semiatuado, é convincente com redução de 50% nos atrasos para baixa demanda e redução de 30% para alta demanda. Espera-se que este efeito desapareça quando a demanda atingir a capacidade da artéria, caso em que, para ambos os sistemas na comparação, a produtividade só pode ser melhorada aumentando o tempo verde em toda a artéria e mantendo a coordenação adequada. A simulação foi executada várias vezes e, tanto para produtividade quanto para atraso, está claro que o RHODES é mais consistente do que o controle semiatuado e oferece menos variabilidade de execução para execução.

[2] semiatuado: modelo misto, hora off-line com otimização, hora online com sensores.

3.2 Phase-by-Phase

O método Phase-by-Phase (PP) foi desenvolvido para superar várias deficiências, que pareciam difundidas em sistemas adaptativos.

- Comprimento de ciclo fixo e/ou passo fixo para variação do comprimento do ciclo.
- Utilização de dados de demanda agregados somente.
- Coordenação fixa de sinais numa artéria e por via de uma rede.

O esquema geral proposto no modelo Phase-by-Phase para melhorar o fluxo é a otimização de interseções isolada e rastreamento mais refinado de veículos. A otimização foi tornada independente da determinação do estado da rede, ou seja, detecção e predição, e, como tal, alguns desses assuntos são principalmente discussões e propostas de melhorias.

Uma estratégia de otimização Phase-by-Phase (MACHEMEHL; SHENODA, 2007) é usar meta-heurística Tabu (HERTZ; WERRA, 1990) para determinar ondas verdes para interseções isoladas, fase por fase. Na forma proposta, o sistema Phase-by-Phase utiliza um processo de Poisson para definir os tempos entre as interseções. Alternativamente o método pode usar análise de séries temporais ou um processo de Poisson com média variável para definição dos tempos. O uso de detecções feitas em tempo real também pode ser usado, como no RHODES. O desempenho do Phase-by-Phase é fortemente dependente da capacidade do sistema de previsão escolhido para gerar previsões adequadas. Na figura 19, podemos ver a abordagem aproximativa variando o tempo de ciclo baseado em uma distribuição de Poisson, buscando assim uma boa solução.

H, tempo de ciclo da interseção, deve estar na ordem do tempo de ciclo desejado e λk são os marcos de mudança de estado e, portanto, temos um plano completo para o sinal. O tempo de vermelho pode ser calculado com base no plano e nas chegadas previstas. Na Figura 19 esta ideia é apresentada.

3.3 DOGS

DOGS é um sistema de otimização de trafego de dupla abordagem, off-line e on-line. É uma extensão do sistema DOG, proposta pela Solução Técnica de Tráfego da Dinamarca (TTS), (LAURITZEN, 2004), que fornece aumentos de capacidade de tráfego baseados em critérios numa artéria. As primeiras 3 letras podem ser traduzidas diretamente do dinamarquês para a Otimização Dinâmica de ondas verdes (Green) e a a

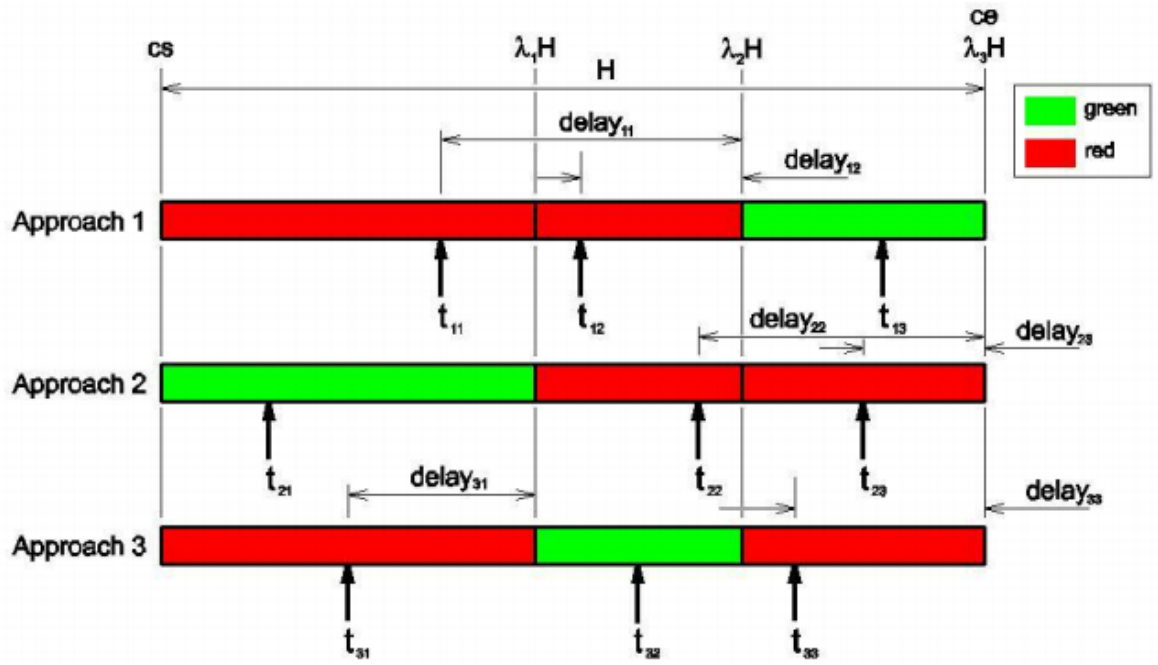


Figura 19 – Abordagens Phase-by-Phase para uma interseção, com valores de ciclo.

letra S adicionada contempla o aspecto de coordenação. Assim, o DOG é um otimizador sensível ao volume de tráfego para interseções simples e o DOGS acrescenta a coordenação entre várias interseção de uma via. O DRD[3] implementou o DOGS atuando sobre várias rodovias na Dinamarca. O DOGS é um sistema baseado em critérios que dependem de um tempo de ciclo comum para coordenação. A área de aplicação pretendida são os sinais de trânsito das artérias, que denotam elevada flutuação na demanda. Na figura 20, parte da rodovia O3 de Harvel, subúrbio de Copenhagen, que está sob a gestão do DOGS, na figura as setas, e os números indicam a direção do fluxo e a contagem da demanda no momento da observação. A contagem neste acaso é obtida por sensores na via e o DOGS atua de maneira adaptativa, ajustando os tempos de ciclos dos semáforos da região para ”otimizar” o fluxo de veículos, produzindo o efeito de onda verde neste setor.

O objetivo do DOGS é aumentar a capacidade do sistema arterial em períodos de alta demanda e reverter para planos otimizados off-line e pré-planejados em situações de baixo tráfego. O aumento de capacidade é realizado aumentando o tempo de ciclo comum e alocando o tempo verde extra, por ciclo, às fases em toda a artéria. Isso causará mais atrasos nas estradas secundárias, mas pode impedir que as filas cheguem à interseção anterior, ou até mesmo evitar filas em casos de congestionamento de luz. O DOGS também é capaz de dar prioridade aos ônibus, estendendo o tempo verde quando os ônibus

[3] DRD-Danish Road Directorate é uma agência de governo, responsável pela rede rodoviária nacional da Dinamarca

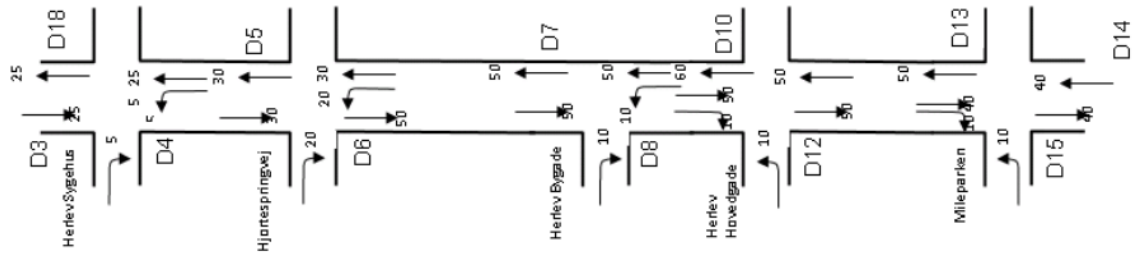


Figura 20 – Layout do fluxo no DOGS. As setas e os números indicam a direção do fluxo e exemplos de contagem de veículos típica.

estão próximos de uma interseção. O DOGS é um sistema puramente sensibilizado pelo volume de tráfego corrente (online) e não trabalha com modelos de predição. Apesar da exequibilidade do sistema, este é um ponto que coloca alta dependência na implementação de engenheiros de tráfego, uma vez que o trafego por meio da artéria deve ser informado manualmente quando no startup do sistema, bem como nas manutenções. Um ponto que minimiza a falta de modelos de previsão da demanda é o fato de que as artérias sob o controle DOGS são relativamente pequenas, e previsões estáticas podem ser feitas por um engenheiro de tráfego experiente. Além disso, como o DOGS só atua sob condições de alta carga, onde qualquer previsões é menos valiosas, pois a conclusão óbvia neste caso é que ela está sobrecarregada.

3.4 SUMO

O projeto SUMO[4] disponibiliza uma ferramenta de simulação de tráfego de código aberto, incluindo componentes de modelagem de demanda, que facilita a investigação de propostas para a otimização de rotas e semáforos (BEHRISCH *et al.*, 2011; KRAJZEWICZ *et al.*, 2012). Sua implementação começou em 2001, com uma primeira versão de código aberto em 2002. Originalmente, havia duas razões para disponibilizar o trabalho como código aberto sob a licença pública GNU (GPL). A primeira foi a intensão de apoiar a comunidade de simulação de tráfego com uma ferramenta gratuita na qual algoritmos próprios podem ser implementados. Grande desvantagem é a comparabilidade quase inexistente dos modelos ou algoritmos implementados, e uma plataforma de simulação comum, aqui considerada benéfica. A segunda razão para fazer a simulação de código aberto foi o intento de obter apoio de outras instituições.

A figura 21 mostra um modelo de simulação de um semáforo com todas as

[4] SUMO-Simulação da Mobilidade Urbana

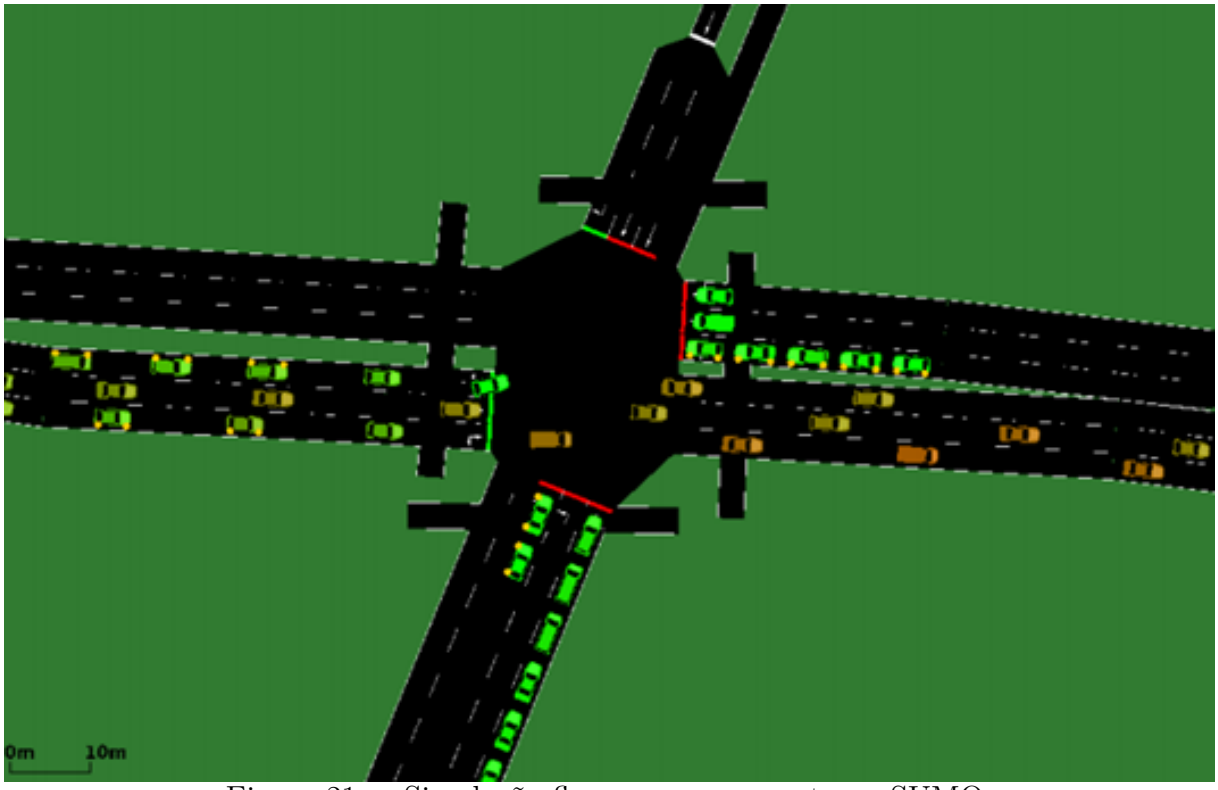


Figura 21 – Simulação fluxo em cruzamento no SUMO.

regras de segurança, como intertravamento, tempo de vermelho de segurança, além da simulação do fluxo de veículos e sua plotagem na via.

O SUMO é uma plataforma de simulação de fluxo de tráfego microscópico, intermodal e multimodal, com espaço contínuo e tempo discreto. O caso não é apenas de uma simulação de tráfego, mas sim de um conjunto de aplicativos, que ajudam a preparar e executar a simulação de um cenário de tráfego. Como o aplicativo de simulação SUMO, que é incluído no pacote, usa formatos próprios para redes rodoviárias e demanda de tráfego, ambos os formatos precisam ser importados ou gerados com base em fontes de tipos diferentes. Tendo em vista a simulação de áreas de grande escala como a principal aplicação para o SUMO, muito esforço foi EMPREGADO no projeto e implementação de heurísticas que determinam atributos ausentes, mas necessários.

No SUMO, as informações de cada veículo são dadas explicitamente. Um veículo é definido pelo menos por identificador único, o horário de partida e a rota do veículo pela rede. Com rota queremos dizer a lista completa de arestas conectadas entre o início e o destino de um veículo. Se necessário, cada veículo pode ser descrito com detalhes, usando propriedades de partida e chegada, como a pista a ser usada, a velocidade ou a posição exata em uma borda. Cada veículo pode obter um tipo atribuído, que descreve as

propriedades físicas do veículo e as variáveis do modelo de movimento usado.

A ferramenta SUMO executa uma simulação discreta no tempo. O comprimento padrão do passo é 1s, mas este valor pode ser definido para ser mais baixo, até 1ms. Internamente, o tempo é representado em microssegundos, armazenado como valores inteiros. A duração máxima de um cenário é limitada a 49 dias. O modelo de simulação é espaço-contínuo e internamente, a posição de cada veículo é descrita pela pista em que o veículo está e a distância desde o início desta faixa. Ao percorrer a rede, a velocidade de cada veículo é calculada usando o chamado modelo de acompanhamento de carros (BEHRISCH *et al.*, 2011). Geralmente, os modelos que acompanham o carro calculam a velocidade de um veículo investigado (ego); observando a velocidade desse veículo, sua distância até o veículo líder (líder) e a velocidade do líder. O SUMO usa por padrão uma extensão do modelo de acompanhamento de veículos estocástico desenvolvido por Stefan Krauß (KRAUB, 1998). Os responsáveis pelo SUMO afirmam que o modelo de Krauß foi escolhido em razão de sua simplicidade e de sua alta velocidade de execução.

O SUMO, portanto, oferece uma plataforma completa de teste de abordagem para o problema tratado nesta tese. Por esse motivo foi que escolhemos a solução linha de base para comparação nos experimentos aqui.

Além disso, o SUMO:

- é uma plataforma aberta, acessível, fácil de configurar e pôr em ação;
- permite trabalhar com mapas e cenários de qualquer região do Planeta;
- não depende de uma infraestrutura em campo, como sensores para captar o comportamento em tempo real do transito; e
- obtém boas soluções com estratégias de força bruta (simulação).

3.5 Resumo

Na tabela 2, oferecemos um resumo das características das propostas RHODES, PP e DOGS que sugerem soluções focadas em obter bons resultados para escopo local de um semáforo ou um trecho de uma via. É provável que do conjunto de soluções ótimas independentes no global, uma pode interferir negativamente na outra, ou seja, nenhum destes métodos fornece uma otimização com a visão global. O método RHODES não oferece solução de otimização, mas um método adaptativo pelo qual se busca uma solução interativamente sendo realimentada com informações dos sensores no campo. Os métodos

PP e DOGS não têm uma visão completa da cidade, o que pode ser prejudicial, pois a solução encontrada pode ser intensamente impactada por um semáforo que está fora do contexto considerado.

Escolhemos o parâmetro Aplicação, que identifica o quão robusta e escalável é a solução, visto que a maioria tem aplicação a um conjunto restrito do problema, como um cruzamento ou trecho de rua. Outro parâmetro escolhido foi o método que permite identificar soluções de otimização ou aproximativa. A Abordagem identifica a maneira de operar a solução entre off-line ou online. Esta última precisa de informações em tempo real do campo. E, finalmente, o parâmetro Dados que descreve o modo de obtenção dos dados de tráfego que são a entrada para o problema.

Título	Autor	Aplicação	Método	Abordagem	Dados
RHODES	Mirchandani et al.	Cidade	Adaptativo	Online	Sensores
Phase-by-Phase	Machemehl at al.	Cruzamento	Poisson	Online	Manual
DOGS	Lauritzen et al.	Avenida	Otimização	Misto	Sensores
SUMO	Behrisch et al.	Cidade	Simulação	Offline	Manual
GPS2GR	Almeida et al.	Cidade	Otimização	Offline	GPS

Tabela 2 – Resumo dos trabalhos relacionados

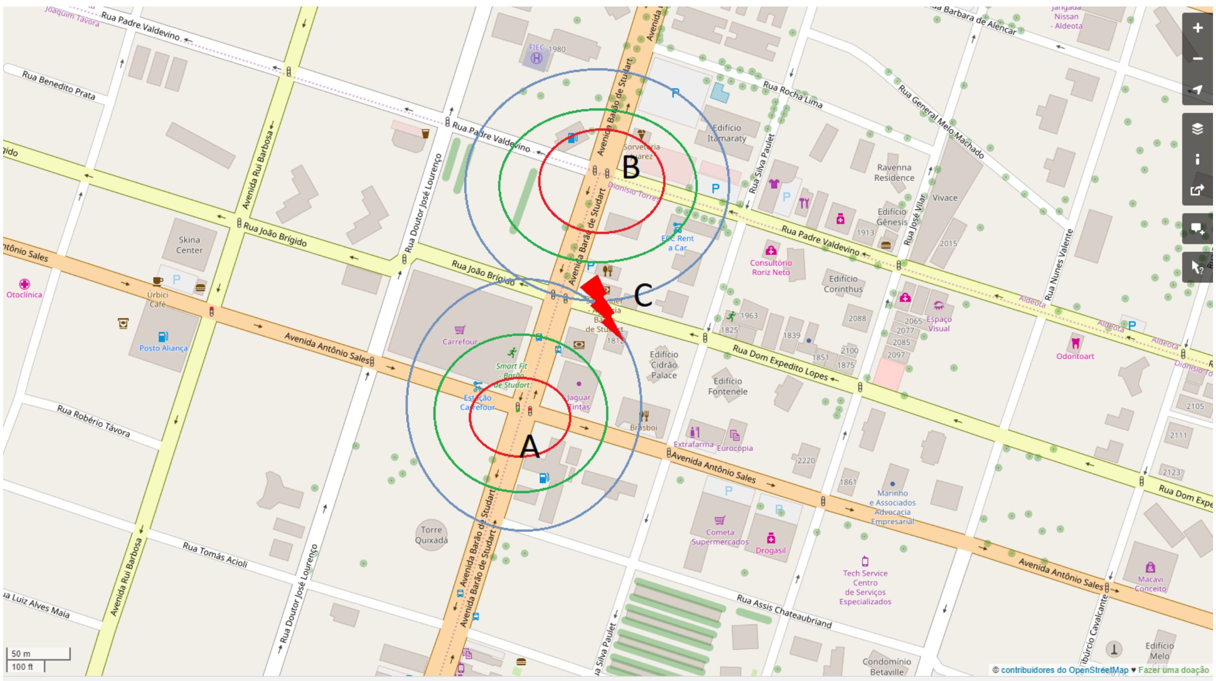


Figura 22 – Exemplo de abordagem de otimização local.

A conjunção de submodelos de otimização aplicada a subconjuntos das ruas de uma cidade, não necessariamente, corresponde à otimização no contexto da cidade inteira,

pois a otimização em subconjuntos das cidades pode gerar parâmetros conflitantes entre tempos de semáforos que são influenciados por dois subconjuntos de soluções. Na figura 22, podemos observar que os pontos de otimização local A e B podem ensejar situações de conflitos em semáforos influenciados por A ou B. No exemplo, é o caso do semáforo C no mapa.

Em resumo, neste capítulo, abordamos.

- os principais trabalhos relacionados a esta tese RHODES, PP, DOGS e SUMO;
- apresentamos os parâmetros de comparação das soluções; e
- no próximo capítulo, trataremos em detalhes da nossa abordagem.

4 GPS2GR

Nossa proposta com o GPS2GR (ALMEIDA *et al.*, 2017) é de transformar um dataset de trajetórias de GPS de veículos em região urbana, em um agendamento sincronizado dos semáforos para formar um comportamento esperado do fluxo de veículos conhecido como onda verde. Procuramos fornecer uma solução de otimização de fluxo em rotas frequentes que são reveladas com base em trajetórias de GPS, de acordo com a variação da densidade do caminho durante o dia. O GPS2GR enseja uma solução otimizada por janela de tempo.

O framework GPS2GR é composto por quatro algoritmos: Time Window - TW, Flow Density Inference - FDI, Hot Route by Time window - HRT e Green Route - GR, conforme descrevemos cada um a seguir. O objetivo global do GPS2GR é otimizar o fluxo de veículos nas cidades, tomando como base datasets de trajetórias de veículos obtidas de GPS.

O algoritmo TW seleciona o melhor método de "discretização" do tempo, com base na densidade de trajetória por tempo, dentre os métodos tradicionais como redução da escala e histograma e técnicas de agrupamento supervisionado aplicadas para discretização com k-means e DBSCAN. O k-means é o método hierárquico popular que trabalha com os centroides para cada grupo e a distância entre o centroide e seus elementos. Já o DBSCAN trabalha com densidade de pontos e portanto foi escolhido com este objetivo.

O algoritmo FDI estima a densidade do fluxo de veículos por seguimento de rua com base na contagem de trajetórias de GPS que passam próximo ao segmento, aplicando o Map-Matching, em cada janela de tempo.

O algoritmo HRT é uma extensão do algoritmo FlowScan para calcular as rotas frequentes por janela de tempo, utilizando, como entrada, a densidade dos fluxos obtidas no algoritmo FDI.

O algoritmo GR enseja uma programação dos tempos dos semáforos sincronizados, de modo a criar condições para que, quando esta programação for aplicada às ruas, incentive a formação de comboios de veículos que trafegam por uma rota frequente, sempre com sinal verde. Isto é possível pois, o agendamento dos semáforos é otimizado com base no deslocamento em comboios de veículos. A principal contribuição no GR é transformar o problema original de programação inteira mista, que é de solução muito custosa, em um problema de otimização combinatória denominado Job shop scheduling.

GPS2GR estende o método de Map-Matching com baixa amostragem (ST-Matching) (ZHENG *et al.*, 2010), estende o método para a detecção de rotas frequente (FlowScan) (LI *et al.*, 2007) e implementa o algoritmo TW utilizando conceitos conceitos da técnica de "discretização" do tempo (CSTH) (ZHANG; LIN, 2004).

Melhoramos o algoritmo do ST-Matching introduzindo um índice k-d tree para a etapa de busca de candidatos próximos *Cut in*, e também introduzimos uma poda no espaço de busca *Cut out*, acelerando, assim, o processo de contagem de densidade no algoritmo FDI.

Finalmente, este trabalho fornece uma solução de otimização para o fluxo de veículos em comboios, onde a estratégia proposta transforma o problema de programação inteira mista (MIP) em um problema de otimização combinatória (JSSP).

Resumo da solução proposta

- TW - algoritmo que calcula as janelas de tempo com base na densidade de trajetórias por fração de tempo. Diversos métodos de discretização sobre o Dataset de trajetórias são utilizados e o que apresenta menor erro quadrático é escolhido.
- FDI - faz a inferência da densidade de tráfego com o apoio da versão estendida do algoritmo ST-Matching para identificar as trajetórias que passam em cada segmento de rua, por janela de tempo.
- HRT - Extensão do algoritmo FlowScan para gerar rotas frequentes, por janela de tempo.
- GR - gera otimizadamente os tempos de ciclo dos semáforos nas rotas frequentes, por janela de tempo.

A Figura 23 mostra a arquitetura proposta em módulos, descrevendo a ordem de execução dos módulos, os dados de entrada e de saída de cada módulo e o resultado final com a programação otimizada dos semáforos.

4.1 TW - Time Window

Tradicionalmente, o histograma é usado para criar janelas de tempo a fim de agrupar variáveis de registro de data e hora, com vista a facilitar a identificação de padrões ou regras (SHIMAZAKI; SHINOMOTO, 2007).

Nossa abordagem propõe o uso de técnicas de agrupamento para avaliar as

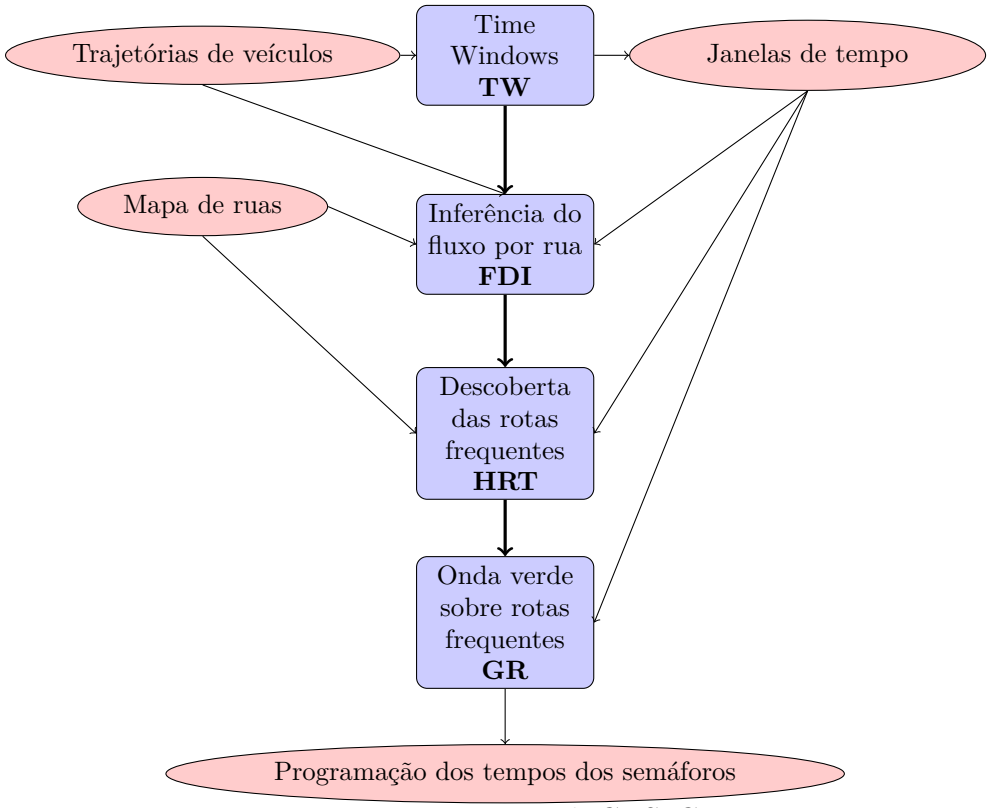


Figura 23 – Framework GPS2GR

trajetórias, usando a variável *timestamp* das trajetórias, comparando a medida de erro com outros os métodos tradicionais, como de redução de escala e histograma. Assim, obtém janelas de tempo que irão influenciar todos as outras etapas nesta solução.

Observamos, por meio de experimentos, que a técnica de agrupamento utilizando k-means gera agrupamento com menor erro do que a técnica de histograma, uma vez que o agrupamento pode gerar janelas de tempo com larguras variadas, e que a densidade das trajetórias tende a variar durante todo o dia. O número de janelas que dividem o dia em intervalos de tempo ideais torna-se a principal decisão para a "discretização" do tempo. Para realizar estes testes, utilizamos a soma do erro quadrado (SSE) e definimos um critério de parada como uma variação menor do que 10% em relação ao erro do ciclo anterior.

$$SSE = \sum_{i=1}^{n} (X_i - \bar{X})^2. \tag{4.1}$$

Em nossa abordagem, X_i é o *timestamp* do ponto de trajetória e $\bar{X}$ é o centroide da classe gerada. Partimos de um conjunto de dados de trajetórias de veículos de Fortaleza (Brasil) e aplicamos diretamente o algoritmo TW. A saída do TW é a "discretização" do

conjunto de trajetórias dependendo da densidade no curso do dia. Ele procura pelo número de clusters que melhor se ajusta ao conjunto de dados. O critério de parada adotado é quando SSE aumenta ou sua variação é menor do que K% de flutuação SSE residual, entre SSE do método de redução de escala e SSE do método de histograma.

Algoritmo 3: TW - Time Windows

Input: set of trajectory points T
Output: set of clusters better adjusted for trajectories

```
1  sturges ← Log2(size(T.nodes))+1;
2  priorSSE ← MaxValue;
3  minClusters ← sturges * 25%;
4  maxClusters ← sturges * 120%;
5  Sort(T.nodes);
   /* Find the best cluster number for the dataset                    */
6  for k in (minClusters..maxClusters) do
7  |  sse ←k-means(T.nodes, k).error;
8  |  if w > priorSSE OR w > 90% * priorSSE then
9  |  |  bestClusters ← k − 1;
10 |  |  break
11 |  end
12 |  priorSSE ← sse;
13 end
   /* Defining the boundaries of each cluster                        */
14 Clustering ←k-means(T.nodes, bestClusters).Clustering;
15 clusters ← {0};
16 k ← Clustering[0];
17 for c in Clustering do
18 |  if c <> k then
19 |  |  clusters.add(T.nodes[c]);
20 |  |  k ← c
21 |  end
22 end
23 clusters.add(1);
24 return clusters[]
```

Os experimentos preliminares demostram que, na "discretização", uma flutuação residual (K) do SSE em relação ao anterior consideramos que atingiu um grau de estabilidade satisfatório, pois, nos experimentos em que reduzimos K, o número de *clusters* aumentou significativamente.

Nos *clusters* obtidos por via deste método de largura variável, alguns tem janela de minutos e outros janela de várias horas, variando de acordo com a densidade do conjunto de dados das trajetórias utilizadas, conforme tensionávamos nas hipóteses da

tese.

- Na linha 7 do algoritmo 3, a variável w armazena o resultado do SSE da última execução do k-means com k agrupamentos.
- Na linha 8 do algoritmo 3, para quando o SSE aumentar em relação a execução anterior ou a variação é menor ou igual a 10% do SSE anterior.

No código fonte 1, podemos observar um exemplo de resultado do algoritmo TW. Nesse exemplo foram geradas cinco janelas de tempo (bucket); casa janela possui um horário de inicio (start) e fim (end). Nesse exemplo, pode-se observar que a ultima janela de tempo E é a mais larga, com quase 14 horas.

Código-fonte 1 – Exemplo da saída do algoritmo TW

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <time-config>
3    <buckets>
4      <bucket name="A" start="00:00" end="09:05"/>
5      <bucket name="B" start="09:05" end="09:42"/>
6      <bucket name="C" start="09:42" end="09:51"/>
7      <bucket name="D" start="09:51" end="10:01"/>
8      <bucket name="E" start="10:01" end="23:59"/>
9    </buckets>
10 </time-config>
```

4.2 FDI - Flow Density Inference by Time Windows

Tradicionalmente, a estimativa da densidade de veículos nas ruas é feita por contagem física no local ou por meio da integração com sensores de semáforos, como ilustrado na figura 24. Propomos um método alternativo de estimar a densidade, baseado em dados de trajetórias de veículos por intermédio de GPS. Na sequência, aplicamos o algoritmo FDI para estimar a densidade do fluxo conforme rua e cluster. Isso é feito anotando a contagem em que cada caminho passou durante o dia. Como saída, obtemos um mapa de ruas com anotação por densidade de tráfego da janela de tempo (clusters).

Na correspondência de mapas, cada ponto de trajetória é uma posição que

Figura 24 – Contagem manual de trafego urbano

deve ser mapeada para um ponto correspondente na rede rodoviária. Considerando erros de precisão do GPS, o ponto mais próximo não é necessariamente o correto. Portanto, toda a trajetória deve ser considerada para evitar uma correspondência potencialmente errada. Assim, em vez de mapear cada ponto para seu nó mais próximo no mapa digital (um método conhecido como correspondência ponto a ponto), um conjunto de caminhos possíveis é mantido e o caminho mais provável é escolhido com base na probabilidade. Isso é conhecido como técnica de múltiplas hipóteses (MTH) e é utilizado em algumas estratégias de correspondência de mapas (SCHUESSLER; AXHAUSEN, 2009), (PYO *et al.*, 2001).

Inicialmente, cada ponto de trajetória deve ser mapeado para seu conjunto de pontos candidatos, que são suas projeções em segmentos de estrada próximos. Isso envolve encontrar os n vértices mais próximos daquela trajetória de ponto na rede viária. Em seguida, encontramos os pontos candidatos para cada ponto de trajetória mapeando-os para sua projeção de segmento de linha (calculada como a distância entre um ponto de um segmento de estrada, mostrado na subseção II) em qualquer aresta que saia ou entre vizinhos mais próximos.

Depois que os pontos candidatos foram recuperados, disparamos várias possibilidades de caminho para a solução. De qualquer ponto candidato $c_{i,j}$ (representando o ponto candidato j-ésimo no *timestamp* i), é possível se deslocar para qualquer ponto candidato $c_{i+1,k}$. Para determinar o caminho mais provável percorrido por uma trajetória, devemos adicionar probabilidades ao nosso modelo.

Figura 25 – Janela de tempo 00h00m a 02h32m

Figura 26 – Janela de tempo 09h13m a 10h45m

Para melhor representar este modelo, um grafo candidato com todos os caminhos possíveis tomados por uma trajetória é utilizado. Neste grafo, cada ponto candidato é um vértice e cada par de pontos candidatos consecutivos de *timestamp* constitui uma aresta. Cada vértice tem uma pontuação associada, relacionada à probabilidade de uma trajetória estar realmente naquele ponto candidato. De igual modo, cada borda possui uma pontuação associada, relacionada à probabilidade de uma trajetória que vai do seu vértice de origem ao seu vértice alvo. Essas probabilidades são calculadas de acordo com o algoritmo ST-Matching, mas este algoritmo ignora a dimensão temporal.

As distâncias menores entre vértices próximos na rede rodoviária precisam ser computadas como parte do cálculo da probabilidade de transição. Como o resultado dessa fase é fixo para a mesma rede de estradas e pode ser bastante caro, dependendo de sua dimensão, essa etapa é processada antecipadamente e armazenada em um sistema de arquivos distribuído, de onde pode ser acessada depois de ler.

Após o grafo candidato ser construído, é possível percorrê-lo para encontrar o caminho ideal, aquele com a maior pontuação combinada. Este passo é essencialmente uma pesquisa inicial que identifica os vértices com os menores *timestamp*.

Algoritmo 4: FDI - Flow Density Inference

```
   Input: road network G, set of trajectories T
   Input: Time Windows TW
   Output: G' set of maps with traffic annotation
 1 G' ← {}
 2 for w in TW do
 3     for t in T do
 4         if t.timestamp ⊂ w then
 5             G'_t ←STMatching(G, t); /* call the STMatching algorithm      */
 6             for s in G'_t do
 7                 s.density ← s.density + 1; /* add one to density count on the
                       segment of the route                                    */
 8             end
 9         end
10     end
11 end
12 return G'
```

- Na linha 5 do algoritmo 4, ocorre a chamada ao algoritmo ST-Matching original.
- Na linha 7 do algoritmo 4, é feita a contagem do número de trajetórias que passam

em cada segmento de uma via no mapa.

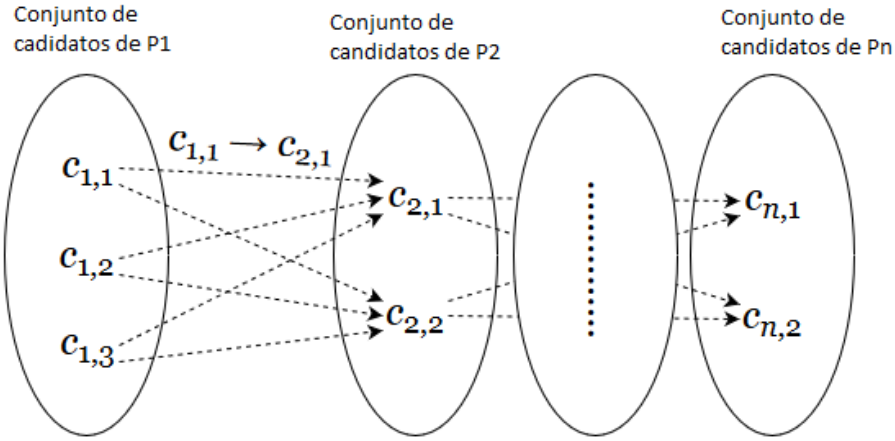


Figura 27 – Exemplo do conjunto de caminhos candidatos

4.3 HRT - Hot Route by Time windows

Por fim, o algoritmo HRT se utiliza o algoritmo FlowScan para identificar as rotas frequentes utilizando como critério de classificação a densidade de fluxo obtida pelo algoritmo FDI. Como saída, obtemos um mapa de rotas de maior densidade, uma para cada janela de tempo (clusters). Usamos esse subconjunto de pistas como entrada para a próxima etapa, na qual aplicaremos algoritmos para veículos de geração de comboios e otimização de semáforos com vistas a incentivar a onda verde em comboios que não geram conflitos em subseções.

O algoritmo HRT usa o conceito de vizinhança definido como as bordas próximas às etapas de Eps, geralmente $Eps = 3$. A função $ForwardNumHops(r, s)$ qualcula o número passos de um caminho possível entre r e s.

$$NEps(r) = \{s \in E | ForwardNumHops(r, s) \leq Eps\}, Eps \geq 0. \tag{4.2}$$

O algoritmo HRT começa identificando os segmentos de rua que se encaixam no conceito Hot Route Start, ou seja, o segmento de rua x, onde a densidade de veículos que passam pelo segmento $traffic(r)$ é maior do que a soma da densidade do segmento antecedente $x \sum traffic(x)$, em que essa diferença é maior do que uma quantidade mínima

de tráfego, $minTraffic$.

$$|traffic(r) - \sum traffic(x)| \geq minTraffic, x \in \{NEps(r)\} \tag{4.3}$$

A rota frequente é construída com origem no *Hot Route Start* e caminha no decurso dos possíveis destinos, escolhendo sempre o caminho mais promissor, ou seja, de maior fluxo, enquanto a densidade mínima se mantém, de acordo com a Equação 4.4. Quando esta condição não é mais satisfeita para qualquer uma das rotas possíveis, a rota frequente é encerrada.

$$|traffic(r) \cap traffic(s)| \geq minTraffic, s \in NEps(r) \tag{4.4}$$

Para cada janela de tempo, o algoritmo HRT produz um submapa com as rotas de maior preservação do fluxo (Rot Routes). Esses arquivos serão a entrada para a próxima fase do problema, que otimizará os semáforos das Hot Routes para aumentar o fluxo dessa maneira.

Algoritmo 5: HRT - Time dependent Hot Route

```
   Input: set of road network G with traffic annotation
   Input: Time Clusters TC
   Output: R[] set of trajectories with annotation
1  R ← {}
2  for c in TC do
3  |  r ← FlowScan(G[c], Eps, minTraffic);
4  |  R.add(r);
5  end
6  return R[]
```

- Na linha 3 do algoritmo 5, repete para cada janela de tempo.
- O algoritmo 5 chama o FlowScan na linha 3.

Na figura 28, podemos observar um mapa da rotas frequentes da primeira janela de tempo do dia.

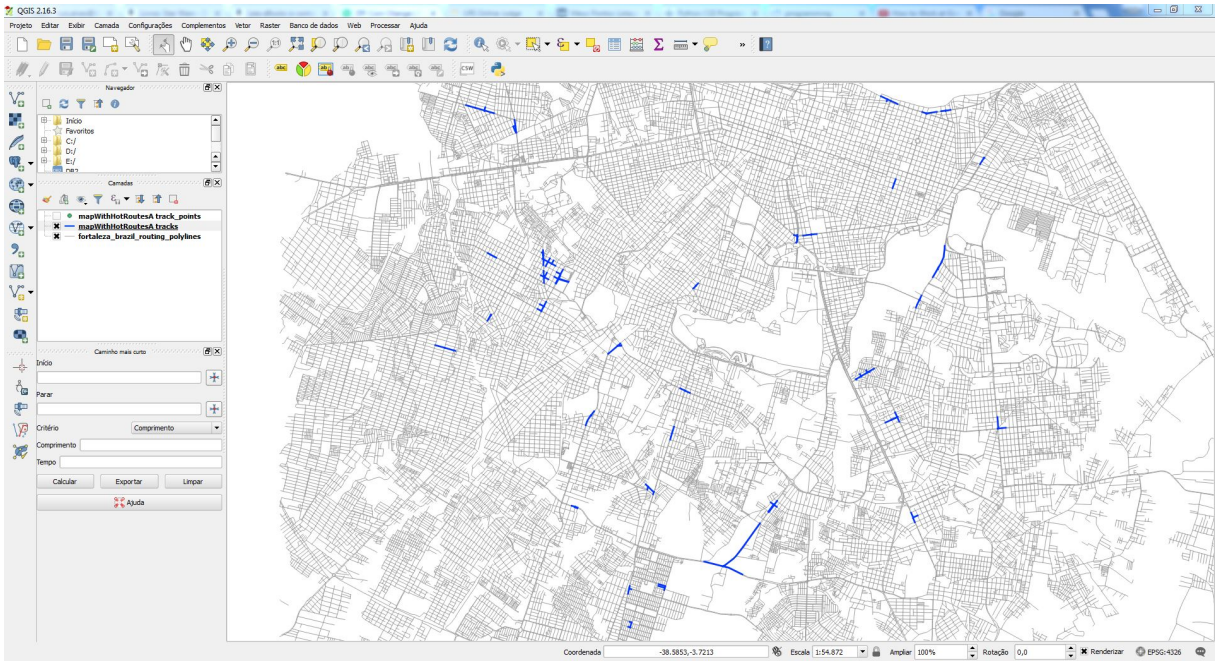


Figura 28 – Rotas frequentes janela de tempo 00h00m a 2h32m

4.4 GR - Green Route by Time Windows

Propomos um método otimizado que resolve o problema das ondas verdes, transformando-o de um problema de otimização clássico para um problema de Job Shop Scheduller. Esta estratégia permite então, o uso do método de Branch and Bound para encontrar uma solução ótima para o problema. Utilizamos no algoritmo GR o solver do Google Optimization Tools[1] para C# (GOOGLE, 2017).

Objetivo para o problema

Seja $p_{i,j}$ o tempo de processamento da tarefa (i, j). Então o tempo final da tarefa (i, j) é $t_{i,j} + p_{i,j}$. Portanto, a duração de uma solução para o problema da oficina é $\max_{i,j} t_{i,j} + p_{i,j}$, onde o máximo é assumido em todas as tarefas. O objetivo do problema é minimizar o tamanho de todas as soluções possíveis.

Geramos um conjunto de comboios de veículos nas rotas mais frequentes e buscamos com o solver, uma solução ótima para este comboio viajar em rotas frequentes, sem ensejar conflito nos cruzamentos com nenhuma outra rota frequente.

No algoritmo GR, propomos a formação de comboios virtuais de veículos em uma rota frequente e criamos um método que otimiza o deslocamento desses comboios em toda a rota frequente, sincronizando os semáforos com este objetivo. Com este modelo, a sequência de semáforos que um comboio tem que atravessar é vista como um conjunto de

[1] <https://developers.google.com/optimization/scheduling/job_shop>

máquinas em um problema de programação de Job Shop Scheduller, onde os comboios correspondem a tarefas. A travessia de um semáforo por um comboio é chamada de operação o_i e requer um tempo de execução p_i (CORMAN *et al.*, 2009; HIGGINS *et al.*, 1996).

4.5 Modelo Webster para ciclos de semáforos

De acordo com o modelo de Webster para semáforos urbanos (WEBSTER; COBBE, 1966), o tempo amarelo de um semáforo isolado é dado pela equação 4.5. Para situações ordinárias, podemos considerar o tempo de reação do motorista $t_r = 1s$ e uma desaceleração média dada por $a = 2,8m/s$. Portanto, o tempo do amarelo depende da velocidade na via de aproximação (V).

$$T_a = T_r + \frac{V}{2a} \tag{4.5}$$

Por simplicidade, o padrão definido na tabela 3 é adotado neste trabalho.

Via de aproximação	Tempo de amarelo
Rua - 40Km/h	3s
Avenida - 60Km/h	4s
Via expressa - 80Km/h	5s

Tabela 3 – Yellow time

O tempo vermelho geral t_{gr} de uma aproximação é uma função das seguintes variáveis:

- velocidade V na aproximação;
- largura L da faixa a ser cruzada (da distância a ser coberta pelo veículo dentro da aproximação);
- tamanho do veículo c; e
- tempo de folga t_f (tempo mínimo decorrido entre o início do tempo da faixa transversal e o momento da entrada de um veículo na velocidade da via).

Considerando as situações usuais, foram adotados os seguintes parâmetros:

- estradas secundárias (ruas) com largura máxima de 9m e velocidade de 40 km/h;
- rotas preferenciais (avenidas) até 30m de largura e velocidade de 60 km/h;

- comprimento de 5m do veículo; e
- tempo de folga $t_f = 1,2s$.

Via de aproximação	Via a ser cruzada	t_{gr}
Qualquer	Rua	$0s$
Rua	Avenida	$2s$
Avenida	Avenida	$1s$
Via expressa	Via expressa ou Avenida	$0.4s \rightarrow 1s$

Tabela 4 – Tempo de vermelho total

De acordo com o modelo de Webster, devemos adotar ciclos submúltiplos de tempo ou período de programação, para permitir a mudança de planos sem cortes abruptos no tempo do verde, além de não adotar t_c a menos de 30 segundos e somente em casos especiais aceitar t_c maior que 120 s

$$T_c = T_g + T_y + T_r + T_{gr} \tag{4.6}$$

O conceito de tempo verde mínimo T_{g-min} está associado a definição do modelo Webster de tempo desperdiçado para aceleração do veículos, e deve ser respeitado sob pena de causar grandes congestionamentos. A tabela 5 mostra os valores usuais de T_{g-min}.

Via de aproximação	Tempo mínimo verde T_{g-min}
Rua - 40Km/h	12s
Avenida - 60Km/h	15s
Via expressa - 80Km/h	17s

Tabela 5 – Tempo mínimo de verde

O problema Job-Shop Scheduling (JSSP)

O JSSP é um problema de otimização combinatória NP-completo e é definido como segue. Considere um conjunto de n tarefas (jobs) e um conjunto de m processadores (máquinas), onde cada tarefa é composta de um conjunto de operações cuja ordem de execução é definida. Para cada operação, queremos saber em qual máquina ela deve ser executada e o tempo necessário para que ela seja concluída. Existem algumas restrições em relação a tarefas e máquinas.

- Uma tarefa não pode ser atribuída à mesma máquina duas vezes.
- Não há ordem de execução em relação a distintas operações de tarefa.

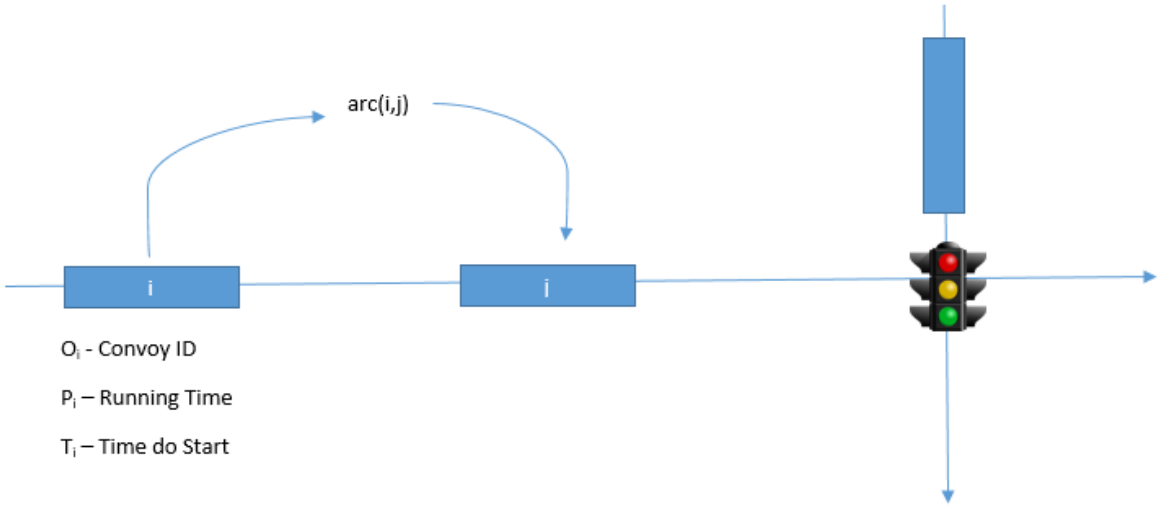


Figura 29 – Detalhe da adaptação do problema de Onda Verde à JSS

- Uma operação não pode ser interrompida.
- Cada máquina pode processar apenas uma tarefa (operação) de cada vez.

O objetivo do JSSP é distribuir tarefas entre máquinas, para que o tempo de processamento necessário para concluir a execução de todas elas (também conhecido como Makespan) seja o menor possível. Em outras palavras, queremos determinar a sequência de máquinas que minimizam o Makespan.

Nessa abordagem, fizemos uma transformação do problema de otimização dos semáforos para um problema do Job Scheduler, onde os semáforos são os processadores e os comboios de veículos são as tarefas.

Modelo Matemático do problema original de otimização dos semáforos

- **Job-shop (J_m)**: Em uma job shop com m machines (semáforos), cada job representa um comboio de carros que tem uma rota particular e predeterminada a seguir (rota frequente)
- **Tempo de processamento (p_{ij})**: O p_{ij} representa o tempo de processamento do Job j na máquina i.
- **Makespan (C_{max})**: O makespan (tempo de processamento), definido como Max $(C_1, ..., C_n)$, a hora final do último trabalho processado no sistema. Um tempo mínimo de processamento, geralmente, implica o uso máximo da(s) máquina(s) (semáforos).

Semáforo duplo	1	2	3	4	5	6	7	8	9	10	11	12	13
Fase 1	G					Y	GR	R					
Fase 2	R						GR	G					Y

Legenda		
	G	Tempo de verde
	Y	Tempo de amarelo
	GR	Tempo de vermelho global
	R	Tempo de vermelho global

Figura 30 – Tempos de ciclo em semáforo com duas fases.

Semáforo triplo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Fase 1	G					Y	GR	R													GR
Fase 2	R						GR	G					Y	GR	R						
Fase 3	R													GR	G					Y	GR

Legenda		
	G	Tempo de verde
	Y	Tempo de amarelo
	GR	Tempo de vermelho global
	R	Tempo de vermelho global

Figura 31 – Tempos de ciclo em semáforo com três fases.

4.6 Transformação do problema de otimização dos semáforos para o problema de agendador de tarefas (JSSP)

Para transformar o problema de otimização da onda verde em um problema de JSS fazem-se necessárias as seguintes transformações a seguir descritas.

1. Os semáforos são mapeados para os processadores.
2. Os comboios são formados para reproduzir o ciclo de Webster na rota frequente.
3. Os comboios são mapeados para os Jobs.
4. O tempo de verde do semáforo é mapeado para o tempo de execução do job no semáforo(processador).
5. As regras do modelo Webster são restrições de problemas.
6. Utilizamos o solver do Google Optimization Tools para o problema de JSSP, para encontrar uma solução ideal que minimize o *makespan*.
7. A solução dá ensejo a um conjunto de Jobs que pode executar sem conflitos nos processadores, ou seja, um conjunto de comboios de veículos; que trafegam na velocidade média da rota, passará em todos os semáforos abertos [2].
8. Os comboios de veículos viajam na velocidade média da pista para aquela janela de

[2] Veículos nos limites do comboio podem, eventualmente, deixar o comboio e ficar retidos em algum semáforo pelo caminho, como é o caso de veículos com velocidade menor ou maior do que a do comboio.

tempo.

Algoritmo 6: GR - Green Way over Hot Route

```
   Input: R[] set of trajectories with flow annotation
   Output: S set of schedduler for all traffic light
 1 Processors ← {};
 2 Jobs ← {};
 3 w ←WebsterModel(R[]);
 4 w.greentime ← w.mingreentime;
 5 (w.greentime++ <= w.maxgreentime) Processors ← DTL(R[], w);
 6 Jobs ← MC(R[], w);
 7 s ←JSSP.Solver(Processors, Jobs);
 8 if s!= null then
 9 |  break;
10 end
   /* repeats by increasing the green time, in search of a solution   */
11 return s;
   /* s variable contains an optimal solution                        */
```

- Na linha 11 do algoritmo 6, a variável s contém a solução ótima da programação do semáforos das rotas frequentes (apenas os tempos de duração da fase verde).

Algoritmo 7: MC-MakeConvoys

```
   Input: R[] set of trajectories with flow annotation
   Input: w Webster model variables
   Output: Jobs list of tasks (convoys)
 1 Convoys ← {};
 2 ConvoysSize ← w.getTotalTimeRoute() / w.CycleTime;
 3 while n ≠ ConvoysSize do
 4 |  c ← new Convoy();
 5 |  c.hotRoute ← this;
 6 |  c.processingTime ← w.GreenTime;
 7 |  c.timeInHotRoute ← w.CycleTime;
 8 |  Convoys.add(c);
 9 end
10 return Convoys;
```

- A rotina 7 é chamada pelo algoritmo GR para criar os comboios, calculando o número de comboio com base no tempo total da rota dividido pelo tempo de ciclo.
- A rotina 8 é chamada pelo algoritmo GR para encontrar os semáforos na rota que precisam ser sincronizados.

No arquivo de saída 2, podemos observar que cada semáforo é identificado por

Algoritmo 8: DTL-DiscoveryTrafficLights

```
   Input: R[] set of trajectories with flow annotation
   Input: w Webster model variables
   Output: Jobs list of processors (traffic lights)
 1 Processors ← {};
 2 foreach s ← R.Segments do
 3     foreach n ← s.Nodes do
 4         Found ← True;
 5         if n.TrafficSignal == True then
 6             t ← new Processor(n.Id);
 7             foreach p ← Processors do
 8                 Result ←checkIfProcessorCloser(p, t);
 9                 if t.Id ≠ p.Id ∧ Result > 30 then
10                     Found ← False;
11                 end
12             end
13             if !Found then
14                 p ← new Processor();
15                 Processors ← p;
16             end
17         end
18     end
19 end
```

um *id*, informação proveniente do Mapa (OSM). O arquivo contém a programação dos tempos de abertura e fechamento do tempo verde no curso do dia. Todos os semáforos das rotas frequentes terão sua programação definida e otimizada para o padrão das trajetórias identificadas.

Código-fonte 2 – Fragmento da saída do algoritmo GR

```
<?xml version="1.0" encoding="UTF-8"?>
<scheduler>
  <traffic-light id="0">
    <green start="00:00:00" end="00:00:17" />
    <green start="00:00:34" end="00:00:51" />
    <green start="00:01:08" end="00:01:25" />
    <green start="00:01:42" end="00:01:59" />
    ...
    <green start="23:59:03" end="23:59:00" />
  </traffic-light>
</scheduler>
```

5 RESULTADOS

5.1 Detalhes de implementação

Toda a implementação do framework foi feita na linguagem C#, porque alguns dos algoritmos utilizados para compor a solução foram originalmente escritos nesta linguagem e gentilmente cedidos pelos autores. Todos os códigos dos algoritmos implementados estão disponíveis no repositório online no endereço <https://github.com/antoniomralmeida/TPM> (ALMEIDA, 2018).

Na Tabela 6, mostramos as entradas e saídas de cada etapa do processo, bem como o encadeamento do processo.

Algoritmo	Entrada	Saída
TW	Trejetórias(GPX)	Janelas de Tempo(XML)
FDI	Trajetórias,Mapa(OSM),Janejas de Tempo	Contagem(OSM)
HRT	Contagem,Janela de Tempo, Mapa	Rotas frequentes(GPX)
GR	Rotas frequentes,Mapa, Janeja de Tempo	Programação semafórica (XML)

Tabela 6 – Encadeamento dos dados de entrada e saída, por etapa

Todos os experimentos realizados nesta tese, demonstram que a divisão das trajetórias em janelas de tempo com largura variável é uma excelente abordagem para encontrar uma solução customizada para realidades distintas do tráfego em uma cidade.

5.2 Origem do DataSet de trajetórias

Para fins de teste, utilizamos dados de trajetória reais cedidos gentilmente por Taxi Simples, uma companhia de táxi com sede em Fortaleza, Brasil (BRAGA, 2016). Este conjunto de dados teve uma taxa de amostragem de alta frequência. Os mapas no formato XML OSM foram obtidos de OpenStreetMap, analisados como grafos de rede de ruas, usando um analisador de XML escrito em C# (ZHENG *et al.*, 2010). O conjunto de dados utilizado coletou trajetórias de táxi da cidade de Fortaleza no período de 14/11/2015 a 09/05/2016, contendo 22.791.003 pontos de trajetórias de GPS.

5.3 Configuração dos Experimentos

Para as experiências, utilizamos um computador Intel (R) Core (TM) i7-4770S @ 3.10GHz com memória RAM de 16GB, Cache L1 de 256KB, 1.0 MB Cache L2 e 8.0 MB Cache L3.

Os dados de trajetórias de táxi (BRAGA, 2016) foram obtidos e descompactados. Tomamos aleatoriamente o dia 6/5/2016 para realizar as experiências deste trabalho. Apenas esse dia contém 749.049 pontos de trajetória do GPS. O formato dos arquivos foi convertido de CSV para GPX, o mapa da cidade de Fortaleza foi baixado do site Open Street Map (OPENSTREETMAP,).

Para demonstrar a eficácia de nossos algoritmos, realizamos a sequência das seguintes etapas nos dados brutos das trajetórias de GPS na cidade de Fortaleza. Em seguida, o GPS2GR foi executado, conforme descrito na figura 23 e os resultados foram registrados e analisados (BEHRISCH *et al.*, 2011).

5.4 Experimentos

Para avaliar a discretização no GPS2GR, comparamos os resultados de três maneiras diferentes de "discretização" da escala de tempo das trajetórias e medimos o número de classes geradas juntamente com o erro observado, usando o critério de soma dos erros dos quadrados (SSE).

- "discretização" da escala de tempo em horas e minutos, arredondando (truncando) a escala do tempo para apenas horas e minutos.
- "discretização" com elaboração de histograma de frequência das trajetórias em colunas de largura fixa.
- "discretização" com base nas fronteiras do agrupamento usando DBSCAN.
- "discretização" com base nas fronteiras do agrupamento usando k-means.

Observamos na Tabela 7 e nas Figuras 32 a 34 que os métodos são reduzidos a minuto e histograma produz um número de classes superior ao limite de Stugers; entre outros métodos K-means [k = 6], apresentou o menor erro, por isso foi escolhido. Portanto, as trajetórias neste conjunto de dados foram divididas em seis conjuntos independentes para as etapas a seguir.

Na figura 34, o erro final (SSE) do método k-means foi entre o erro do método

Method	N.Cluster	SSE	Status
Down. Minute	81	94,34	Above the Sturges
Down. Hour	7	182,48	
Histogram	21	75,60	Above the Sturges
DBSCAN	2	282,24	
k-means[5]	5	139,20	
k-means[6]	6	126,42	Minor error
Stugers	19		

Tabela 7 – ”discretização” com DataSet 400.000 pontos de trajetórias.

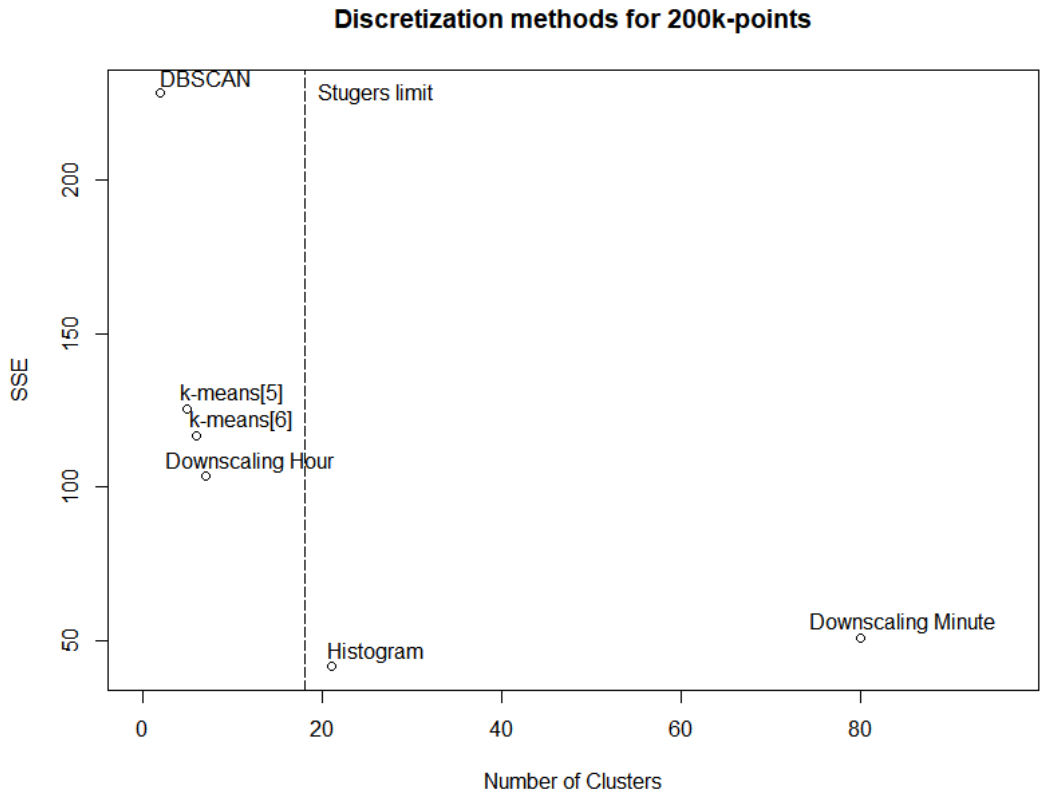


Figura 32 – “discretização” com DataSet 200mil pontos de trajetórias.

do histograma e o erro do método de truncamento no tempo.

Todas as abordagens utilizadas neste estudo mostraram resultados muito promissores e alta aplicabilidade. Com base nas trajetórias em tempo real dos dados brutos, podemos obter o comportamento do tráfego da cidade em cada fatia de tempo em que o comportamento é linear. A visualização de consulta pode ser utilizada como meio para descobrir o fluxo de maior densidade (rotas quentes) e influenciar a elaboração de políticas públicas para o tráfego urbano.

Nos experimentos, usamos o *software* de simulação de mobilidade urbana (SUMO) (versão 0.29.0) (CENTER, 2017). Essa ferramenta é de código aberto e tem uma

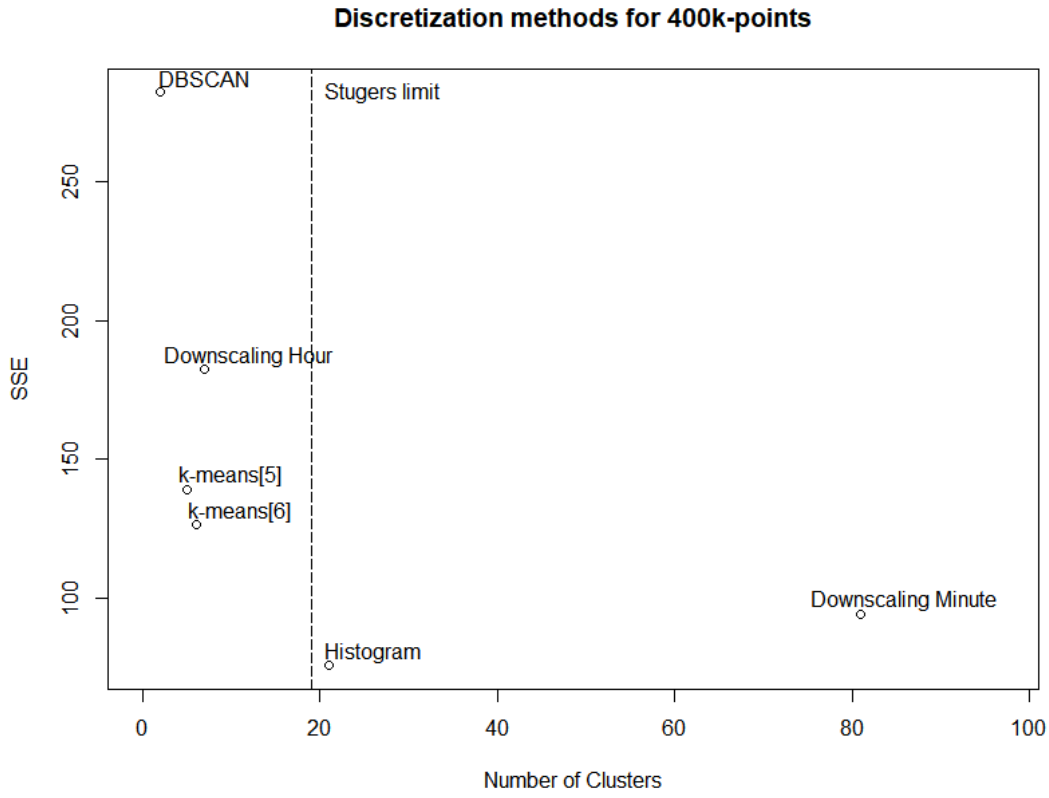


Figura 33 – ”discretização” com DataSet 400mil pontos de trajetórias.

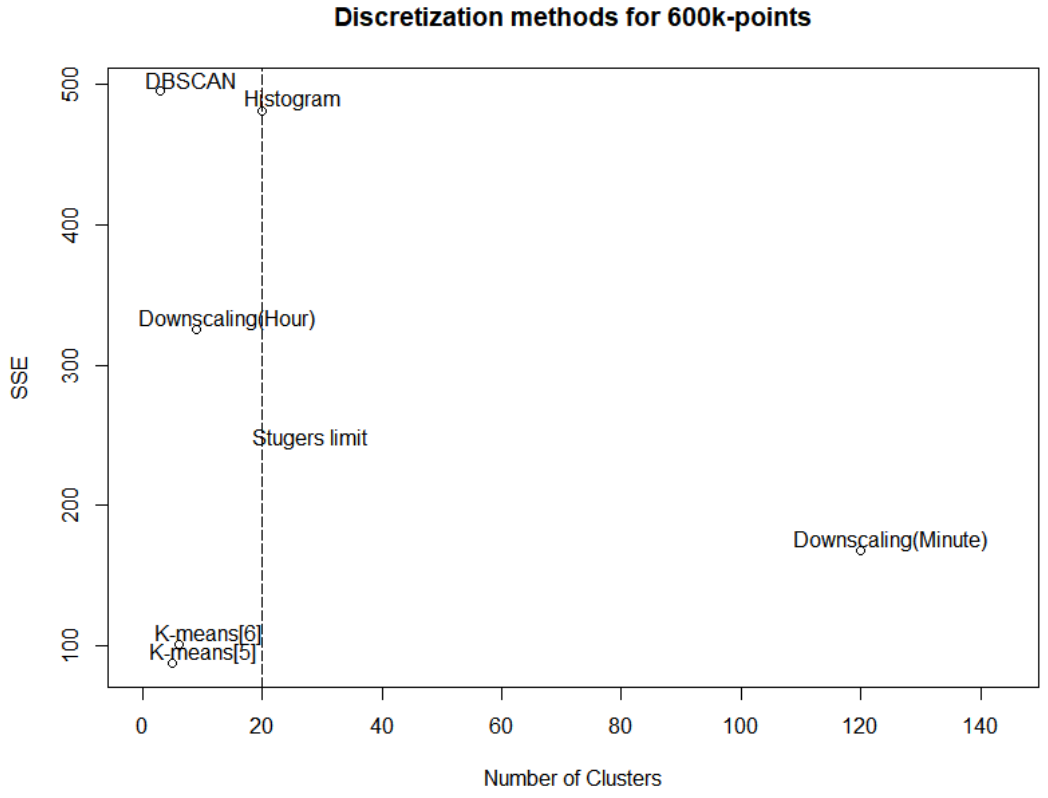


Figura 34 – ”discretização” com DataSet 600mil pontos de trajetórias.

Figura 35 – Resultado do FDI para janela de tempo[1]

Figura 36 – Resultado do FDI para janela de tempo[2]

Figura 37 – Resultado do FDI para janela de tempo[3]

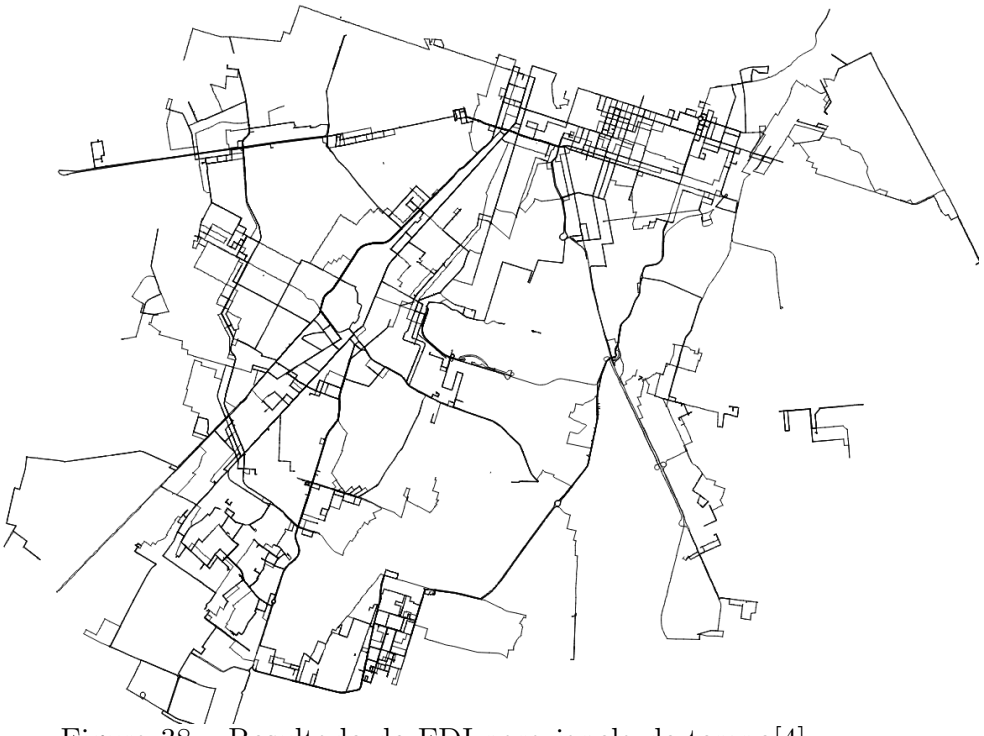


Figura 38 – Resultado do FDI para janela de tempo[4]

Figura 39 – Resultado do FDI para janela de tempo[5]

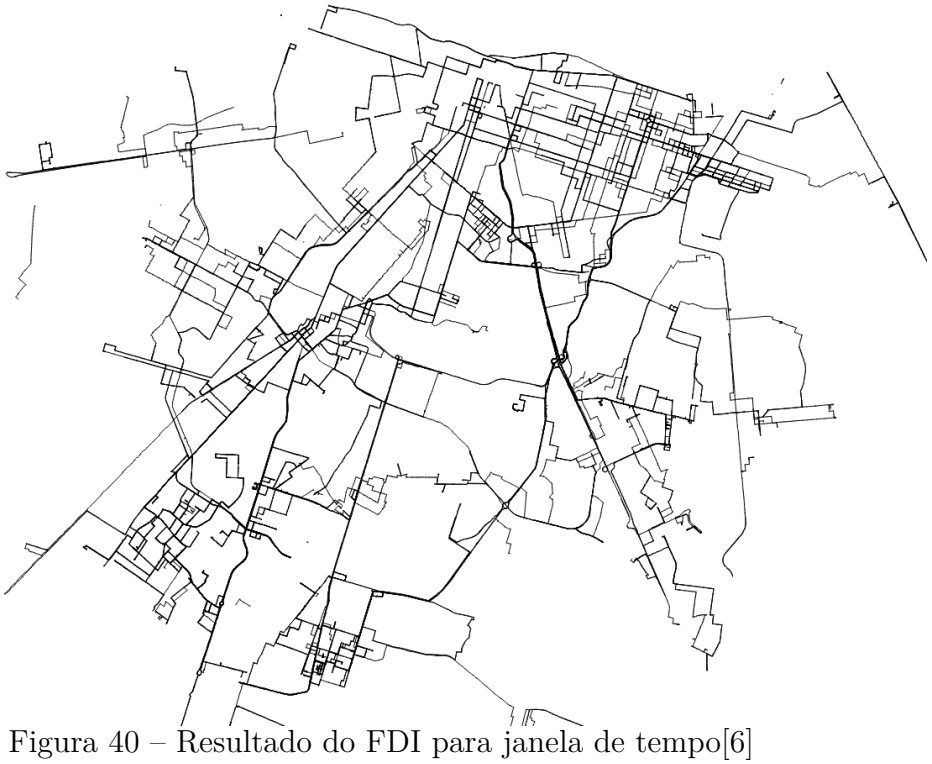


Figura 40 – Resultado do FDI para janela de tempo[6]

Figura 41 – Resultado do FDI para janela de tempo[7]

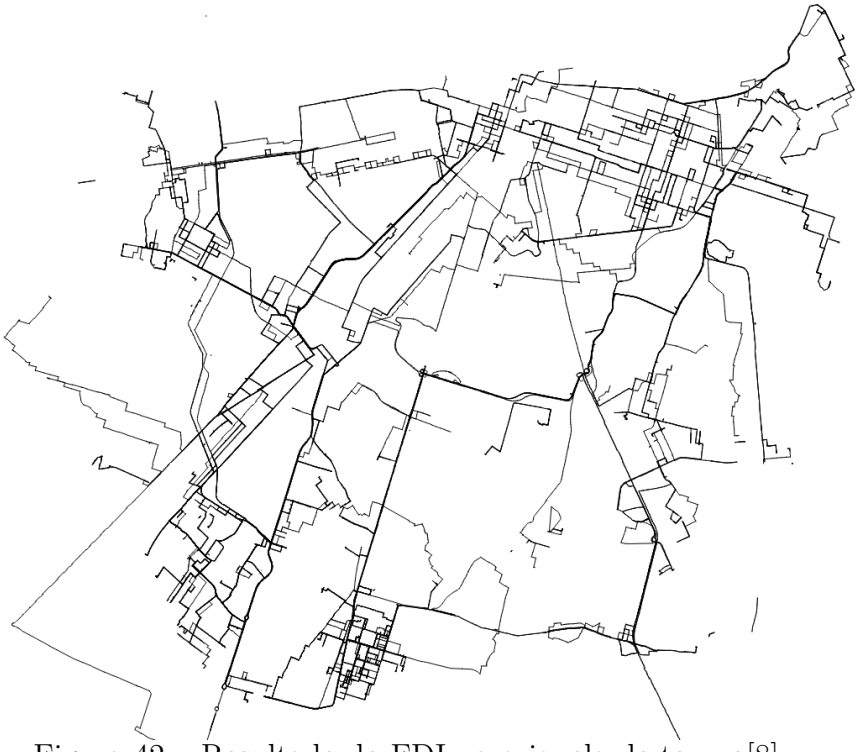


Figura 42 – Resultado do FDI para janela de tempo[8]

Figura 43 – Hot Routes geradas por HRT na janela de tempo[4]

notória eficiência em grandes conjuntos de rotas de dados. Além disso, para mostrar a eficácia do algoritmo Green Route, três simulações foram realizadas em um processador Intel (R) Core (TM) i7-4770S a 3.10GHz.

5.5 Avaliação

A avaliação dos nossos resultados foi baseada nos números de clusters gerados por parte de cada algoritmo. Além disso, levamos em conta a soma do erro quadrado (SSE) de cada solução. A fórmula de Sturges é usada para nos informar se a solução está em bom intervalo. De acordo com (STURGES, 1926), a fórmula é capaz de calcular o número de intervalos para um conjunto de observações. No nosso caso, as observações são os pontos de trajetória de dados no espaço. Utilizamos no primeiro experimento um banco de dados de 200mil pontos de dados, conforme descrito na Figura 32. Embora o método Histograma tenha obtido o menor SSE, está fora dos limites estabelecidos pela fórmula de Sturges. A mesma situação acontece com Downscaling por minuto, no entanto, seu resultado é pior, porque o número de *cluster* é enorme.

Continuando nossa avaliação na Figura 32, podemos ver que o método de

redução de escala, por sua vez, obteve excelentes resultados; está dentro dos limites declarados pela fórmula de Sturges, e tem um SSE considerável. A mesma situação acontece para o método K-means, que contém cinco ou seis centroides. Finalmente, apesar de o DBSCAN ter gerado um excelente número de *clusters*, e estar nos limites propostos, seu SSE não é aceitável.

No segundo experimento, descrito na Figura 33, o K-means alcançou melhores resultados. Precisamente, o algoritmo K-means com seis centroides obteve um SSE decente e um grande número de *clusters*. Por outro lado, desta vez, a redução de escala por hora apresentou um desempenho fraco em comparação com os resultados da figura 32. As abordagens restantes ficaram mais próximas de seus últimos resultados.

Em conclusão, a eficácia do algoritmo K-means pode ser confirmada no último experimento descrito na Figura 34. Desta vez, diferentemente do resultado da figura 33, o algoritmo K-means com cinco centroides alcançou os melhores resultados. O downscaling por hora não melhorou o método histograma, pois permaneceu no limite da fórmula de Sturges. O erro do histograma, no entanto, ainda é muito grande. Mais uma vez, a redução de escala por minuto e o DBSCAN não tiveram melhorias em relação à experimentação.

Observando os resultados das figuras 35 a 42, podemos constatar que a estratégia de segregação das trajetórias em janelas de tempo permitiu a configuração bastante distintas das rotas frequentes, o que vai permitir a geração de soluções.

Para o algoritmo GR, o *baseline* escolhido foi o SUMO, pois enseja uma boa solução, embora seu algoritmo de Monte Carlo não seja performático. Nas tabelas 8 e 9, podemos observar o resultado da simulação de fluxo, comparando os tempos entre o SUMO livre para encontrar os melhores tempos dos semáforos e com os tempos encontrados pelo GR, comparando com os tempos obtidos com a programação dos semáforos do *framework* GPS2GR. Não utilizamos o como *baseline* o RHODES ou DOGS pois dependem de dados obtidos de sensores para seu funcionamento, sendo inviável a experimentação. Já a comparação com método Phase-by-Phase não se justificou pois não utiliza método de otimização, mais um método aproximativo de Poisson.

Para comparação das abordagens, utilizamos três principais índices de qualidade de uma programação de semáforos com simulação. Este parâmetros são gerados pelo SUMO ao final da etapa de simulação.

- Tempo de espera - Tempo médio em segundos que os veículos ficaram retidos, parados,

em congestionamentos.

- Duração - Tempo médio da duração das viagens dos veículos na simulação
- Tempo perdido - Tempo médio perdido aos se dirigir em velocidade menor que o possível para via.

Resultados da Simulação	Cenário I - 791 veículos		
	SUMO	GR	%
Número de veículos #	791	791	
Número de Hot Routes #	1	1	
Distância total percorrida (km)	84,94	84,94	
Métricas de qualidade			
Tempo de espera (s)	11,6	11,32	-2,41
Duração (s)	110,95	110,53	-0,38
Tempo perdido (s)	34,2	33,8	-1,17

Tabela 8 – Resultado de simulação para o Cenário I

Resultados da Simulação	Cenário II - 6.291 veículos		
	SUMO	GR	%
Número de veículos #	6291	6291	
Número de Hot Routes #	3	3	
Distância total percorrida (km)	653,26	653,26	
Métricas de qualidade			
Tempo de espera (s)	28,49	28,11	-1,33
Duração (s)	247,5	246,74	-0,31
Tempo perdido (s)	77,71	77	-0,91

Tabela 9 – Resultado de simulação para o Cenário II

Como podemos observar na figura 44, embora as respostas encontradas pela solução SUMO já sejam muito próximas da solução ótima, esta foi obtida por um processo computacional muito mais custoso, conforme a Tabela 10 o algoritmo GR levou três minutos para encontrar uma solução ótima, a solução SUMO levou 36 horas de processamento no mesmo conjunto de dados para encontrar uma solução boa, ver tabela 11. Vale ressaltar que a etapa de simulação é exatamente igual tanto na abordagem com GPS2GR e SUMO, em ambos os cenários a ferramenta SUMO vai criar fluxos virtuais de veículos na mesma quantidade e sobre as mesmas rotas definidas em cada cenário.

A justificativa para a diferença no tempo de processamento das duas abordagem está na etapa de “Encontrar tempos dos semáforos” que no caso da abordagem GPS2GR é feita em 48 minutos, conforme podemos observar na tabela 10 e na abordagem SUMO

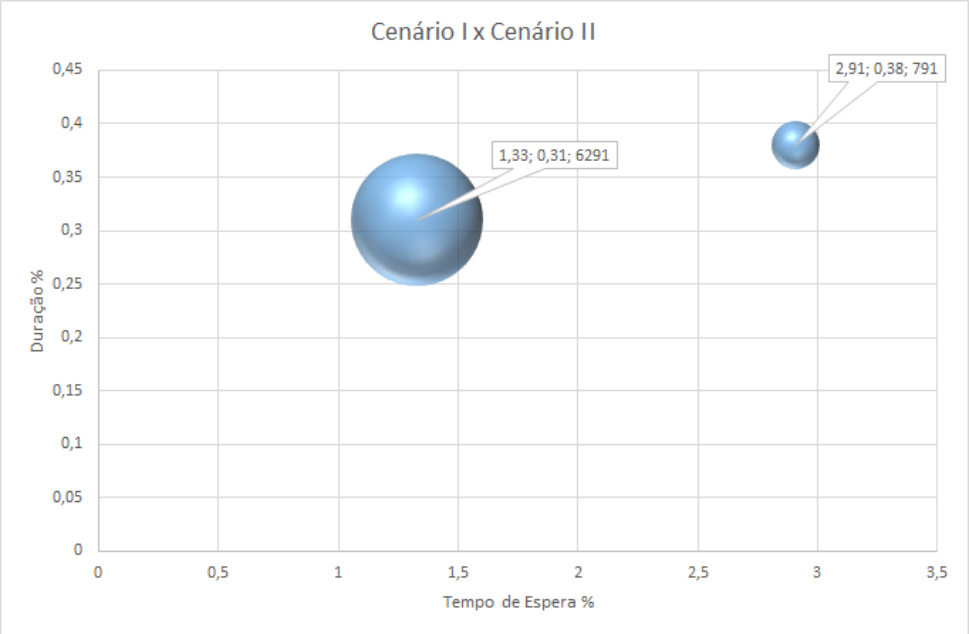


Figura 44 – Comparativos do cenário I com o cenário II

chegou a 36 horas no cenário II. Deve-se destacar também que o cenário II foi definido com base nos limites operacionais da ferramenta SUMO, enquanto a abordagem GPS2GR concluiu um cenário muito mais robusto, considerando todas as rotas da cidade de Fortaleza.

Etapa	Tempo de execução em minutos
Time Window –TW	20
Flow Density Inference - FDI	10
Hot Route by Time - HRT	15
Green Route - GR	3
TOTAL	48

Tabela 10 – GPS2GR Tempo de execução- Dataset II Taxi Simples

Característica	GPS2GR	SUMO
Encontrar tempos dos semáforos	Calculado pelo GR	Definido pelo SUMO
Fluxo de veículos	Simulado	Simulado
Tempo de execução - Cenário I	5 minutos	20 horas
Tempo de execução - Cenário II	20 minutos	36 horas

Tabela 11 – Tempo de execução das simulações

Avaliação das Hipóteses

1. Hipótese: a hipótese inicial de que se evitarmos a abordagem clássica de histograma ou redução de escala, utilizando métodos de clusterização para identificar os agrupamentos de trajetórias na linha do tempo e, após isso, utilizarmos as fronteiras

dos agrupamentos para definir os limites das janelas de tempo, teremos janelas de tamanho variável e mais ajustadas à distribuição das trajetórias.

- Comentário: tendo em vista os experimentos realizados com algoritmo TW, ficou comprovada a abordagem utilizando os limites variáveis dos agrupamentos das trajetórias, que ofereceu o melhor equilíbrio entre número de classes e menor erro quadrático - SSE. Além do mais, os agrupamento possuem larguras variáveis, que, do ponto de vista do problema, é benéfico, pois aglutina grupos de trajetórias de comportamento semelhante, contribuindo assim para reduzir a complexidade geral do problema.

2. Hipótese: a hipótese neste caso é que se usando um algoritmo preciso de Map-matching, pode-se criar um método de contagem automatizado, por segmento de rua, que aproveita os dados públicos de trajetórias para inferir a densidade relativa de fluxo de veículos para o segmento de rua.
    - Comentário: o algoritmo FDI foi testado e avaliado e seu resultado ensejou como saída um mapa de rede com densidade relativa com base no fluxo de trajetórias que passaram pelo segmento de rua na janela de tempo definida pelo algoritmo TW. As alterações introduzidas no algoritmo original ST-Matching surtiram efeito positivo na redução do tempo de execução para o mesmo conjunto de dados utilizado.
3. Desafio: dado que o problema de obter a densidade de fluxo, por segmento de rua, esteja resolvido, utilizando um algoritmo de preservação de fluxo com valor de corte baixo, de 3 a 5 veículos, pode revelar de modo significativo as principais rotas de veículos neste período.
    - Comentário: o algoritmo HRT não só revelou uma fotografia do fluxo da cidade, como também deixou claro que a estratégia de separar o problema em janela de tempo revela radiografias do fluxo completamente diferente da mesma cidade em cada janela de tempo. Também observamos que o fluxo residual fora das rotas frequentes é muito baixo, normalmente um a dois veículos em um segmento de rua.
4. Hipótese: transformar o problema para otimização combinatória e introduzir restrições práticas de estudos sobre tempos de semáforos que reduzam a complexidade do problema e o torne viável computacionalmente.

- Comentário: o algoritmo GR implementou a transformação do problema original de otimização em um problema de otimização combinatória. Utilizamos um solver de Job Scheduler para encontrar a solução ótima. Normalmente o GR executou em poucos minutos.

5. Hipótese: utilizar ferramenta de simulações de tráfego de veículos comparando os tempos dos semáforos obtidos com os de outra solução e analisar os tempos médios de deslocamentos.
    - Comentários: os experimentos com SUMO demonstraram que a solução obtida com otimização foi ainda melhor do que a solução conseguida com força bruta. Concluímos, portanto, que a solução ótima gerada em 0,93% do tempo de processamento, em relação a força bruta (SUMO), oferece uma solução viável e factível para ser utilizada diariamente para processar os dados de trajetórias dos dias anteriores e projetar os tempos de semáforos para o dia seguinte.

6 CONCLUSÕES E TRABALHOS FUTUROS

Fomos capazes de concluir uma implementação dos métodos propostos da solução GPS2GR e realizar experimentos em todos as fases do processo que comprovaram sua correção e viabilidade computacional.

A "discretização" temporal por métodos de agrupamento utilizados no algoritmo de TW forneceu benefícios a consultas espaço-temporais, criando subproblemas que reduzem a complexidade da solução e permitem a geração de soluções específicas para cada janela de tempo. O TW também reduz a escala de variáveis temporais e facilita a visualização, permitindo a identificação de padrões em conjuntos de trajetórias.

Os experimentos mostraram que o algoritmo FDI possui alta precisão e escalabilidade, podendo ser aplicado a dados em trajetórias de fluxo contínuo, para aplicações onde a correspondência de trajetórias precisa ser realizada com precisão e rapidez em tempo real. O FDI se mostrou um método prático para estimar a densidade de tráfego, por segmento de rua, usando trajetórias de GPS, substituindo os métodos convencionais de contagem manual em campo ou contagem por sensores instalados na via. O algoritmo HRT identificou longas e significativas rotas frequentes, bem como gerou mapas de rotas frequentes completamente distintos para cada janela de tempo.

Finalmente, o algoritmo GR convergiu e ensejou soluções otimizadas dos tempos de abertura e fechamento dos semáforos nas rotas frequentes (YING *et al.*, 2014). Observamos que a solução de otimização dos tempos dos semáforos, obtida pelo algoritmo GR foi melhor que a obtida pelos métodos comparados, nos dois cenários observados, já que sua solução de otimização foi sempre melhor do que as boas soluções ensejadas pelos métodos heurísticos. A otimização na rota pode resolver problemas clássicos de contenção de veículos no final de uma onda verde em apenas uma avenida, já que leva em consideração a otimização da rota frequente, isto é, otimiza os semáforos do caminho até a dispersão dos veículos.

Como a abordagem do GPS2GR é off-line, não há como reagir a eventos inesperados que ocorram na via, porém, se houver uma mudança em via que modifique sensivelmente o fluxo por mais de um dia, o GPS2GR já refletirá a mudança no replanejamento do dia seguinte.

6.1 Trabalhos Futuros

Como contribuição para futuros trabalhos, sugerimos a aplicação da solução GPS2GR numa situação real em campo, comparando os resultados com a circunstância anterior da implantação do método, assim poderíamos demonstrar o ganho real da otimização dos semáforos versus uma situação caótica e pouco otimizada do campo.

Também é possível fazer uma extensão no GPS2GR de modo que permita a execução com dados em tempo real, possibilitando que situações atípicas no campo, como acidentes ou obras, possam ser contempladas em uma nova solução de otimização dos semáforos, considerando assim os fatos novos do campo. O GPS2GR, porém, não teria a dupla abordagem presente no DOGS, que opera em modo off-line e, em momentos de contenção do tráfego, alterna para online, passando a obter dados diretamente dos sensores em campo. O GPS2GR operaria sempre em modo off-line porém com o tempo reduzido entre a realidade e a geração de uma nova solução otimizada.

É possível também evoluir o modo de distribuir o planejamento para os semáforos nas ruas, de modo consistente. O principal desafio é assegurar que todos recebam o novo plano e o ativem de maneira simultânea, evitando inconsistências entre semáforos. Há uma hipótese que considera a possibilidade de utilizar o algoritmo *two-phase commit* para este fim. Este aspecto pode ser explorado em outros trabalhos científicos.

REFERÊNCIAS

ALMEIDA, A.; LIMA, M.; MACEDO, J.; MACHADO, J. Dmm: A distributed map-matching algorithm using the mapreduce paradigm. IEEE, 2016.

ALMEIDA, A. M.; LIMA, M. I.; MACEDO, J. A.; MACHADO, J. C. Dmm: A distributed map-matching algorithm using the mapreduce paradigm. In: IEEE. **Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on**. [S.l.], 2016. p. 1706–1711.

ALMEIDA, A. M. R. **GitHub repository of project**. 2018. Disponível em: <https://github.com/antoniomralmeida/TPM>. Acesso em: 15 jul. 2018.

ALMEIDA, A. M. R.; LEITE, J. L. A.; MACEDO, J. A. F.; MACHADO, J. C. Gps2gr: Optimized urban green routes based on gps trajectories. In: **Proceedings of the 8th ACM SIGSPATIAL Workshop on GeoStreaming**. [s.n.], 2017. (IWGS'17), p. 39–48. ISBN 978-1-4503-5492-9. Disponível em: <http://doi.acm.org/10.1145/3148160.3148167>. Acesso em: 15 jul. 2018.

BEHRISCH, M.; BIEKER, L.; ERDMANN, J.; KRAJZEWICZ, D. Sumo–simulation of urban mobility: an overview. In: THINKMIND. **Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation**. [S.l.], 2011.

BISHOP, C. M. **Pattern Recognition and Machine Learning (Information Science and Statistics)**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN 0387310738.

BRAGA, M. **Taxi Dataset Fortaleza, Ceará**. 2016. Disponível em: <https://s3.amazonaws.com/txsimples_tracking/index.html>. Acesso em: 23 set. 2016.

BROCKFELD, E.; BARLOVIC, R.; SCHADSCHNEIDER, A.; SCHRECKENBERG, M. Optimizing traffic lights in a cellular automaton model for city traffic. **Physical Review E**, APS, v. 64, n. 5, p. 056132, 2001.

CENTER, G. A. **Simulation of Urban MObility**. 2017. Disponível em: <http://sumo.dlr.de/index.html>. Acesso em: 01 abr. 2017.

CORMAN, F.; D'ARIANO, A.; PACCIARELLI, D.; PRANZO, M. Evaluation of green wave policy in real-time railway traffic management. **Transportation Research Part C: Emerging Technologies**, Elsevier, v. 17, n. 6, p. 607–616, 2009.

ESTER H. KRIEGEL, J. S. X. X. M. A density-based algorithm for discovering clusters in large spatial database with noise. In: MAGNENAT-THALMANN, N.; THALMANN, D. (Ed.). **Proceeding of the 2rd ACM SIGKDD**. [S.l.]: ACM SIGKDD, 1996. p. 226, 231.

ESTER, M.; KRIEGEL, H.-P.; SANDER, J.; XU, X. *et al.* A density-based algorithm for discovering clusters in large spatial databases with noise. In: **KDD Knowledge Discovery and Data Mining**. [S.l.: s.n.], 1996. v. 96, n. 34, p. 226–231.

FU, T. A review on time series data mining. In: **Engineering Applications of Artificial Intelligence**. [S.l.]: Elsivier, 2011. p. 164, 181.

FUHAO, Z.; JIPING, L. An algorithm of shortest path based on dijkstra for huge data. In: IEEE. **Fuzzy Systems and Knowledge Discovery, 2009. FSKD'09. Sixth International Conference on**. [S.l.], 2009. v. 4, p. 244–247.

GIANNOTTI, F.; NANNI, M.; PINELLI, F.; PEDRESCHI, D. Trajectory pattern mining. In: ACM. **Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2007. p. 330–339.

GOOGLE. **Google Optimization Tools**. 2017. Disponível em: <https://developers.google.com/optimization/>. Acesso em: 03 fev. 2017.

GREENFELD, J. S. Matching gps observations to locations on a digital map. In: **Transportation Research Board 81st Annual Meeting**. [S.l.: s.n.], 2002.

HERTZ, A.; WERRA, D. de. The tabu search metaheuristic: how we used it. **Annals of Mathematics and Artificial Intelligence**, Springer, v. 1, n. 1, p. 111–121, 1990.

HIGGINS, A.; KOZAN, E.; FERREIRA, L. Optimal scheduling of trains on a single line track. **Transportation research part B: Methodological**, Elsevier, v. 30, n. 2, p. 147–161, 1996.

JAIN, A. K. Data clustering - 50 years beyond k-means. In: **Pattern Recognition Letters**. [S.l.]: Elsivier, 2010. p. 651, 666.

KANUNGO, A.; SHARMA, A.; SINGLA, C. Smart traffic lights switching and traffic density calculation using video processing. In: IEEE. **Engineering and computational sciences (RAECS), 2014 recent advances in**. [S.l.], 2014. p. 1–6.

KAUFMAN, L.; ROUSSEEUW, P. J. **Finding groups in data: an introduction to cluster analysis**. [S.l.]: John Wiley & Sons, 2009. v. 344.

KRAJZEWICZ, D.; ERDMANN, J.; BEHRISCH, M.; BIEKER, L. Recent development and applications of sumo-simulation of urban mobility. **International Journal On Advances in Systems and Measurements**, v. 5, n. 3&4, 2012.

KRAUB, S. Microscopic modeling of traffic flow: investigation of collision free vehicle dynamics. 1998.

LAURITZEN, S. M. Evaluation of dogs. **Danish Road Directorate**, 2004.

LI, X.; HAN, J.; LEE, J.-G.; GONZALEZ, H. Traffic density-based discovery of hot routes in road networks. In: SPRINGER. **International Symposium on Spatial and Temporal Databases**. [S.l.], 2007. p. 441–459.

LINDEN, R. Técnicas de agrupamento. **Revista de Sistemas de Informação da FSMA**, v. 4, p. 18–36, 2009.

MACHEMEHL, R.; SHENODA, M. **Development of a Phase-by-Phase, Arrival-Based, Delay-Optimized Adaptive Traffic Signal Control Methodology with Metaheuristic Search.** [S.l.], 2007.

MIRCHANDANI, P.; WANG, F.-Y. Rhodes to intelligent transportation systems. **IEEE Intelligent Systems**, IEEE, v. 20, n. 1, p. 10–15, 2005.

MIWA, T.; KIUCHI, D.; YAMAMOTO, T.; MORIKAWA, T. Development of map matching algorithm for low frequency probe data. **Transportation Research Part C: Emerging Technologies**, Elsevier, v. 22, p. 132–145, 2012.

NAGATANI, T. Vehicular traffic through a sequence of green-wave lights. **Physica A: Statistical Mechanics and its Applications**, Elsevier, v. 380, p. 503–511, 2007.

OPENSTREETMAP. **OpenStreetMap**. Disponível em: <http://www.openstreetmap.com/>. Acesso em: 03 jun. 2016.

PYO, J.-S.; SHIN, D.-H.; SUNG, T.-K. Development of a map matching method using the multiple hypothesis technique. In: IEEE. **Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE**. [S.l.], 2001. p. 23–27.

ROBERTSON, D. I. Transyt: a traffic network study tool. 1969.

SASAKI, M.; NAGATANI, T. Transition and saturation of traffic flow controlled by traffic lights. **Physica A: Statistical Mechanics and its Applications**, Elsevier, v. 325, n. 3, p. 531–546, 2003.

SCHUESSLER, N.; AXHAUSEN, K. W. Map-matching of gps traces on high-resolution navigation networks using the multiple hypothesis technique (mht). **Arbeitsberichte Verkehrsund Raumplanung**, v. 568, 2009.

SHIMAZAKI, H.; SHINOMOTO, S. A method for selecting the bin size of a time histogram. **Neural computation**, MIT Press, v. 19, n. 6, p. 1503–1527, 2007.

STURGES, H. A. The choice of a class interval. American Statistical Association, p. 65, 66, 1926.

TAN MICHAEL STEINBACH, V. K. P.-N. **Introduction to Data Mining**. 1th. ed. [S.l.]: Pearson, 2005.

WARBERG, A.; LARSEN, J.; JØRGESEN, R. M. **Green wave traffic optimization-a survey**. [S.l.]: Informatics and Mathematical Modeling, Technical University of Denmark, 2008.

WEBSTER, F.; COBBE, B. Traffic signals. road research technical paper no. 56. **HMSQ, London**, v. 111, 1966.

WEI, L.-Y.; ZHENG, Y.; PENG, W.-C. Constructing popular routes from uncertain trajectories. In: ACM. **Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2012. p. 195–203.

WITTEN, E. F. I. H. **Data Mining - Pratical Machine Learning Tools and Techniques**. 2th. ed. [S.l.]: Elsevier, 2005.

YANG, X. K.; ZHOU, H. G. Corsim-based simulation approach to evaluation of direct left turn versus right turn plus u-turn from driveways. **Journal of Transportation Engineering**, American Society of Civil Engineers, v. 130, n. 1, p. 68–75, 2004.

YING, J. J.-C.; SHI, B.-N.; LAN, K.-C.; TSENG, V. S. Spatial-temporal mining for urban map-matching. 2014.

ZHANG, Q.; LIN, X. Clustering moving objects for spatio-temporal selectivity estimation. In: AUSTRALIAN COMPUTER SOCIETY, INC. **Proceedings of the 15th Australasian database conference-Volume 27**. [S.l.], 2004. p. 123–130.

ZHENG, Y. Trajectory data mining: an overview. **ACM Transactions on Intelligent Systems and Technology (TIST)**, ACM, v. 6, n. 3, p. 29, 2015.

ZHENG, Y.; LOU, Y.; ZHANG, C.; XIE, X. **Map-Matching for Low-Sampling-Rate GPS Trajectories**. [S.l.]: Google Patents, 2010. US Patent App. 12/712,857.

APÊNDICE A – CÓDIGOS FONTES GPS2GR

Código-fonte 3 – Código-fonte do algoritmo TimeWindows-TW

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using NDesk.Options;
using LK.GPXUtils;
using System.Xml;

namespace LK.TimeWindows
{
    class Program
    {

        static void Main(string[] args)
        {
            string gpxPath = "";
            int samplingPeriod = 0;
            DateTime span = DateTime.Now;
            bool showHelp = false;
            bool filter = false;
            List<Double> tlist = new List<Double>();
            List<Double> rre = new List<Double>();


            OptionSet parameters = new OptionSet() {
                { "gpx=",   "path to the GPX file to process or to the directory to
                    process",               v => gpxPath = v},
                { "p|period=", "sampling period of the GPX file",
                                                                v => samplingPeriod =
                    int.Parse(v)},
                { "f|filter", "enables output post processing",
                                                                  v => filter = v !=
                    null},
                { "h|?|help",

                    v => showHelp = v != null},
            };

            try
            {
                parameters.Parse(args);
            }
            catch (OptionException e)
            {
                Console.Write("TimeWindows: ");
                Console.WriteLine(e.Message);
                Console.WriteLine("Try TimeWindows --help for more information.");
                return;
            }


            if (showHelp || string.IsNullOrEmpty(gpxPath))
            {
                ShowHelp(parameters);
                return;
            }


            // Process signle file
            if (File.Exists(gpxPath))
            {
                ProcessGPXFile(gpxPath, tlist, samplingPeriod, filter);
            }
            // Process all GPX in directory
            else if (Directory.Exists(gpxPath))
            {
                var files = Directory.GetFiles(gpxPath, "*.gpx");
                Console.WriteLine("Found {0} GPX file(s).", files.Length);

                foreach (var file in files)
                {
                    ProcessGPXFile(file, tlist, samplingPeriod, filter);
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("No GPX files found");
```

```
72              }
73
74              int DataSetSize = tlist.Count();
75              int numClusters = 2;
76              double oldWithinss = double.MaxValue;
77              double withinss;
78              double[][] means;
79              int[] count;
80              int[] clustering;
81              System.IO.StreamWriter csv;
82
83              System.IO.StreamWriter log = new System.IO.StreamWriter("TimeWindows.log")
                    ;
84
85              Console.WriteLine("Sorting DataSetSize=" + DataSetSize);
86              tlist.Sort();
87              double[][] rawData = new double[DataSetSize][];
88              for (int l = 0; l < DataSetSize; l++)
89                  rawData[l] = new double[] { tlist[l] };
90
91              csv = new System.IO.StreamWriter("TimeWindows.csv");
92              for (int n = 0; n < rawData.Length; n++)
93                  csv.WriteLine(rawData[n][0] );
94              csv.Close();
95
96
97              //Discretization Round Time
98
99              Console.Write("Testing Discretization by Downscaling (Minute)");
100
101             clustering = DiscTime.DiscretizationRoundTime(rawData, out numClusters,
                    out means, out count, out withinss, "M");
102             Console.WriteLine(",numClusters= "+ numClusters + ", withinss=" + withinss
                    );
103             rre.Add(withinss);
104             log.WriteLine(numClusters + ";" + withinss);
105
106             csv = new System.IO.StreamWriter("TimeWindows1-M.csv");
107             for (int n=0;n<numClusters;n++)
108                 csv.WriteLine(means[n][0] + ";" + count[n]);
109             csv.Close();
110
111
112             Console.Write("Testing Discretization by Downscaling (Hour)");
113
114             clustering = DiscTime.DiscretizationRoundTime(rawData, out numClusters,
                    out means, out count, out withinss, "H");
115             Console.WriteLine(",numClusters= " + numClusters + ", withinss=" +
                    withinss);
116             rre.Add(withinss);
117             log.WriteLine(numClusters + ";" + withinss);
118
119             csv = new System.IO.StreamWriter("TimeWindows1-H.csv");
120             for (int n = 0; n < numClusters; n++)
121                 csv.WriteLine(means[n][0] + ";" + count[n]);
122             csv.Close();
123
124
125             //Discretization Histogram
126
127             Console.Write("Testing Discretization by Histogram");
128             clustering = HistSample.Histogram(rawData, out numClusters, out means, out
                     count, out withinss);
129             Console.WriteLine(",numClusters= " + numClusters + ", withinss=" +
                    withinss);
130             rre.Add( withinss);
131             log.WriteLine(numClusters + ";" + withinss);
132
133             csv = new System.IO.StreamWriter("TimeWindows2.csv");
134             for (int n = 0; n < numClusters; n++)
135                 csv.WriteLine(means[n][0] + ";" + count[n]);
136             csv.Close();
137
138             Console.Write("Testing Discretization KMeans");
139
140             //Sturges formula
141             int max_cluster;
142             int min_cluster;
143
144             Utils.Sturges(rawData.Length, out min_cluster, out max_cluster);
145
146             double limWithinss = (2 * rre[0] + rre[1])/ 3;
147             rre.Add(double.MaxValue);
148             int c = min_cluster;
149
150             do
```

```
151             {
152                 Console.Write(".");
153                 clustering = KMeans.Cluster(rawData, c, out means, out count, out
                        withinss,2); // this is it
154                 //Console.WriteLine(", withinss=" + withinss);
155                 rre.Add(withinss);
156                 double tax = (rre[rre.Count-1]- rre[rre.Count - 2]) / rre[rre.Count -
                        2];
157                 log.WriteLine(c + ";" + withinss);
158                 if ((c > min_cluster && (tax > 0 || tax > -0.1))) // 10%
159                 {
160                     numClusters = c - 1;
161                     break;
162                 }
163                 if (withinss < limWithinss)
164                 {
165                     numClusters = c;
166                     break;
167                 }
168                 oldWithinss = withinss;
169
170             } while (c++ <= max_cluster);
171
172             log.Close();
173
174             clustering = KMeans.Cluster(rawData, numClusters, out means, out count,
                    out withinss, 3); // this is it
175             Console.WriteLine(",numClusters= " + numClusters + ", withinss=" +
                    withinss);
176
177             // DBScan - Variables
178             List<double> points = new List<double>();
179             int row = 0, col = 0;
180             for (row = 0; row < rawData.Length; row++)
181             {
182                 for (col = 0; col < rawData[row].Length; col++)
183                 {
184                     points.Add(rawData[row][col]);
185                 }
186             }
187
188             // Starting DBScan
189             DBScan db = new DBScan();
190             db.dbscan(rawData, points, 0.03, 100);
191
192             csv = new System.IO.StreamWriter("TimeWindows3.csv");
193             for (int n = 0; n < numClusters; n++)
194                 csv.WriteLine(means[n][0] + ";" + count[n]);
195             csv.Close();
196
197             List<Double> buckets = new List<double>();
198             buckets.Add(0);
199             int k = clustering[0];
200             for (int j=1;j< clustering.Length;j++)
201                 if (clustering[j] != k)
202                 {
203                     buckets.Add(rawData[j-1][0]);
204                     k = clustering[j];
205                 }
206             buckets.Add(0.99999);
207
208
209
210             System.Xml.XmlDocument doc = new XmlDocument();
211
212              //(1) the xml declaration is recommended, but not mandatory
213              XmlDeclaration xmlDeclaration = doc.CreateXmlDeclaration("1.0", "UTF-8",
                     null);
214              XmlElement root = doc.DocumentElement;
215              doc.InsertBefore(xmlDeclaration, root);
216
217              //(2) string.Empty makes cleaner code
218              XmlElement element1 = doc.CreateElement(string.Empty, "time-config",
                     string.Empty);
219              doc.AppendChild(element1);
220              XmlElement element2 = doc.CreateElement(string.Empty, "buckets", string.
                     Empty);
221              element1.AppendChild(element2);
222
223              char i = "A";
224
225              for (int j = 0; j < (buckets.Count - 1); j++)
226              {
227
228                  XmlElement element3 = doc.CreateElement(string.Empty, "bucket",
                         string.Empty);
```

```
229                 element2.AppendChild(element3);
230
231                 XmlAttribute attribute = doc.CreateAttribute("name");
232                 attribute.Value = i.ToString();
233                 element3.Attributes.Append(attribute);
234
235
236                 XmlAttribute attribute2 = doc.CreateAttribute("start");
237                 attribute2.Value = DateTime.FromOADate(buckets[j]).ToString("HH:mm:ss
                        ");
238                 element3.Attributes.Append(attribute2);
239
240                 XmlAttribute attribute3 = doc.CreateAttribute("end");
241                 attribute3.Value = DateTime.FromOADate(buckets[j + 1]).ToString("HH:
                        mm:ss");
242                 element3.Attributes.Append(attribute3);
243
244                 i++;
245
246             }
247
248             String fileout = Path.Combine(Path.GetDirectoryName(gpxPath), "
                    TimeWindows.xml");
249             doc.Save(fileout);
250
251             Console.WriteLine("\t\tDone.");
252             Console.WriteLine("\tSpan=" + (DateTime.Now - span));
253
254         }
255
256
257         static void ProcessGPXFile(string path, List<Double> tlist, int samplingPeriod
                , bool filterOutput)
258         {
259             LK.GPXUtils.Filters.FrequencyFilter filter = new LK.GPXUtils.Filters.
                    FrequencyFilter();
260
261             Console.Write("Loading {0} ...", Path.GetFileName(path));
262             GPXDocument gpx = new GPXDocument();
263             gpx.Load(path);
264
265             Console.WriteLine("[{0} track(s); {1} segment(s)]", gpx.Tracks.Count, gpx.
                    Tracks.Sum(track => track.Segments.Count));
266             for (int trackIndex = 0; trackIndex < gpx.Tracks.Count; trackIndex++)
267             {
268                 for (int segmentIndex = 0; segmentIndex < gpx.Tracks[trackIndex].
                        Segments.Count; segmentIndex++)
269                 {
270                     string name = string.IsNullOrEmpty(gpx.Tracks[trackIndex].Name) ?
                            "t" + trackIndex.ToString() : gpx.Tracks[trackIndex].Name.
                            Replace("\\", "-").Replace(":", "");
271                     name += "_s" + segmentIndex.ToString();
272                     //Console.Write("\t" + name + " ");
273
274                     try
275                     {
276                         GPXTrackSegment toProcess = gpx.Tracks[trackIndex].Segments[
                                segmentIndex];
277                         if (samplingPeriod > 0)
278                             toProcess = filter.Filter(new TimeSpan(0, 0,
                                    samplingPeriod), toProcess);
279
280
281                         foreach (var t in toProcess.Nodes)
282                             tlist.Add(Utils.Time2Double(t.Time));
283                         //Console.WriteLine(".");
284                     }
285                     catch (Exception e)
286                     {
287                         Console.WriteLine("Error: " + e.Message);
288                     }
289                 }
290             }
291         }
292
293         /// <summary>
294         /// Prints a help message
295         /// </summary>
296         /// <param name="p">The parameters accepted by this program</param>
297         static void ShowHelp(OptionSet p)
298         {
299             Console.WriteLine("Usage: TimeWindows [OPTIONS]+");
300             Console.WriteLine("Matches GPX track to the OSM map");
301             Console.WriteLine();
302
303             Console.WriteLine("Options:");
```

```
304 |                 p.WriteOptionDescriptions(Console.Out);                                                    |
305 |             }                                                                                              |
306 |         }                                                                                                  |
307 | }                                                                                                          |
```

Código-fonte 4 – Código-fonte do algoritmo Frequency Density Inference-FDI

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using NDesk.Options;
using Utils;
using LK.OSMUtils.OSMDatabase;
using LK.GPXUtils;
using LK.FDI.XMLUtils;
using LK.GeoUtils.Geometry;
using LK.GeoUtils;

namespace LK.FDI

{
    class Program
    {
        static void Main(string[] args)
        {
            DateTime span = DateTime.Now;

            string osmPath = "";
            string gpxPath = "";
            string xmlPath = "";
            string outputPath = ".";
            int samplingPeriod = 0;
            bool showHelp = false;
            bool filter = false;

            OptionSet parameters = new OptionSet() {
                { "osm=", "path to the routable map file",
                                                                    v => osmPath = v
                  },
                { "gpx=", "path to the GPX file to process or to the directory to
                  process",                  v => gpxPath = v},
                { "xml=", "path to the XML file with the time buckets",
                                                        v => xmlPath = v},
                { "o|output=", "path to the output directory",
                                                                v => outputPath = v},
                { "p|period=", "sampling period of the GPX file",
                                                            v => samplingPeriod =
                  int.Parse(v)},
                { "f|filter", "enables output post processing",
                                                              v => filter = v !=
                  null},
                { "h|?|help",

                  v => showHelp = v != null},
            };

            try
            {
                parameters.Parse(args);
            }
            catch (OptionException e)
            {
                Console.Write("FDI: ");
                Console.WriteLine(e.Message);
                Console.WriteLine("Try fdi --help for more information.");
                return;
            }


            if (showHelp || string.IsNullOrEmpty(osmPath) || string.IsNullOrEmpty(
                gpxPath) || string.IsNullOrEmpty(xmlPath) || string.IsNullOrEmpty(
                outputPath))
            {
                ShowHelp(parameters);
                return;
            }


            if (outputPath[outputPath.Length - 1] == "\"")
            {
                outputPath = outputPath.Substring(0, outputPath.Length - 1);
            }


            Console.Write("Loading OSM file ...");
            OSMDB map = new OSMDB();
            map.Load(osmPath);
            Console.WriteLine("\t\t\tDone.");

```

```
71              Console.Write("Building routable road graph ...");
72              RoadGraph graph = new RoadGraph();
73              graph.Build(map);
74              Console.WriteLine("\tDone.");
75
76              TMM processor = new TMM(graph) { _db = map };
77              PathReconstructer reconstructor = new PathReconstructer(graph) { _db = map
                    };
78
79              XMLDocument xml = new XMLDocument();
80              xml.Load(xmlPath);
81              var buckets = xml.Buckets;
82
83              // Process single file
84              if (File.Exists(gpxPath))
85              {
86                  List<GPXTrack> result = new List<GPXTrack>();
87                  result.AddRange(getAllGpxTrackList(gpxPath));
88
89                  ProcessGPXFile(gpxPath, processor, reconstructor, outputPath,
                        samplingPeriod, filter, buckets);
90                  GenerateOsmFiles(buckets, reconstructor, map, result);
91                  GenerateGpxFiles(buckets, gpxPath, 0);
92              }
93              // Process all GPX in directory
94              else if (Directory.Exists(gpxPath))
95              {
96                  var files = Directory.GetFiles(gpxPath, "*.gpx");
97                  List<GPXTrack> result = new List<GPXTrack>();
98
99                  Console.WriteLine("Found {0} GPX file(s).", files.Length);
100
101                 for (int i = 0; i < files.Length; i++)
102                 {
103                     ProcessGPXFile(files[i], processor, reconstructor, outputPath,
                            samplingPeriod, filter, buckets);
104                     GenerateGpxFiles(buckets, gpxPath, i);
105                     result.AddRange(getAllGpxTrackList(files[i]));
106
107                     Console.WriteLine("NEW FILE BEING PROCESSED");
108                 }
109
110                 GenerateOsmFiles(buckets, reconstructor, map, result);
111             }
112             else
113             {
114                 Console.WriteLine("No GPX files found");
115             }
116
117             Console.WriteLine("\tDone.");
118             Console.WriteLine("\tSpan=" + (DateTime.Now - span));
119
120         }
121
122         static List<GPXTrack> getAllGpxTrackList(string file)
123         {
124             GPXDocument gpx = new GPXDocument();
125             List<GPXTrack> gpxTrackList = new List<GPXTrack>();
126             List<GPXTrackSegment> gpxTrackSegList;
127
128             gpx.Load(file);
129
130             foreach (var trk in gpx.Tracks)
131             {
132                 gpxTrackSegList = new List<GPXTrackSegment>();
133
134                 // Sanatizing the data, some segments have zero nodes.
135                 foreach (var seg in trk.Segments)
136                 {
137                     if (seg.Nodes.Count != 0)
138                     {
139                         gpxTrackSegList.Add(seg);
140                     }
141                 }
142
143                 // Clearing the olds
144                 trk.Segments.Clear();
145                 // Assigning the new ones.
146                 trk.Segments.AddRange(gpxTrackSegList);
147
148                 if (trk.Segments.Count != 0)
149                 {
150                     gpxTrackList.Add(trk);
151                 }
152             }
153
```

```
154             return gpxTrackList;
155         }
156
157         static void ProcessGPXFile(string path, TMM processor, PathReconstructer
                reconstructor, string outputPath, int samplingPeriod,
158             bool filterOutput, List<Bucket> buckets)
159         {
160             GPXUtils.Filters.FrequencyFilter filter = new GPXUtils.Filters.
                    FrequencyFilter();
161
162             Console.Write("Loading {0} ...", Path.GetFileName(path));
163             GPXDocument gpx = new GPXDocument();
164             gpx.Load(path);
165
166             Console.WriteLine("[{0} track(s); {1} segment(s)]", gpx.Tracks.Count, gpx.
                    Tracks.Sum(track => track.Segments.Count));
167             for (int trackIndex = 0; trackIndex < gpx.Tracks.Count; trackIndex++)
168             {
169                 Console.WriteLine(gpx.Tracks[trackIndex].Name);
170
171                 for (int segmentIndex = 0; segmentIndex < gpx.Tracks[trackIndex].
                        Segments.Count; segmentIndex++)
172                 {
173                     string name = string.IsNullOrEmpty(gpx.Tracks[trackIndex].Name) ?
                            "t" + trackIndex.ToString() : gpx.Tracks[trackIndex].Name.
                            Replace("\\", "-").Replace(":", "");
174                     name += "_s" + segmentIndex.ToString();
175                     Console.Write("\t" + name + " ");
176
177                     try
178                     {
179                         GPXTrackSegment toProcess = gpx.Tracks[trackIndex].Segments[
                                segmentIndex];
180
181                         if (samplingPeriod > 0)
182                             toProcess = filter.Filter(new TimeSpan(0, 0,
                                    samplingPeriod), toProcess);
183
184                         if (toProcess.NodesCount > 1) {
185                             var result = processor.Match(toProcess);
186                             Console.Write(".");
187
188                             var reconstructedPath = reconstructor.Reconstruct(result);
189
190                             Console.Write(".");
191
192                             if (filterOutput)
193                             {
194                                 reconstructor.FilterUturns(reconstructedPath, 100);
195                             }
196
197                             Console.WriteLine(".");
198
199                             var trackId = gpx.Tracks[trackIndex].Name.Replace("trk_",
                                    "");
200                             buckets = GetUpdatedBuckets(toProcess, reconstructedPath,
                                    buckets, trackId);
201
202                         } else {
203                             throw new Exception(string.Format("Track segment discarded
                                     because number of nodes is less than 2."));
204                         }
205                     }
206                     catch (Exception e)
207                     {
208                         Console.WriteLine("Error: " + e.Message);
209                     }
210                 }
211             }
212         }
213
214         static List<Bucket> GetUpdatedBuckets(GPXTrackSegment toProcess, List<Polyline
                <IPointGeo>> path,
215             List<Bucket> buckets, String trackId)
216         {
217             var start = toProcess.Nodes.First().Time.TimeOfDay;
218             var end = toProcess.Nodes.Last().Time.TimeOfDay;
219
220             foreach (var b in buckets)
221             {
222                 if (start >= b.Start && end <= b.End)
223                 {
224                     b.Paths.Add(trackId, path);
225                 }
226             }
```

```
227             return buckets;
228         }
229 
230         static void GenerateOsmFiles(List<Bucket> buckets, PathReconstructer
                reconstructor, OSMDB map, List<GPXTrack> gpxTrackList)
231         {
232 
233             foreach (var b in buckets)
234             {
235                 if (b.Paths.Any())
236                 {
237                     var mapCopy = ObjectCopier.Clone<OSMDB>(map);
238                     List<Polyline<IPointGeo>> pathList = new List<Polyline<IPointGeo
                            >>();
239 
240                     OSMNode bucketInfo = new OSMNode(0, 0, 0);
241                     OSMTag start = new OSMTag("start", b.Start.ToString());
242                     OSMTag end = new OSMTag("end", b.End.ToString());
243                     bucketInfo.Tags.Add(start);
244                     bucketInfo.Tags.Add(end);
245 
246                     foreach (var p in b.Paths)
247                     {
248                         var uniquePath = p.Value.GroupBy(x => new { x.Id }).Select(x
                                => x.First());
249 
250                         foreach (var seg in uniquePath)
251                         {
252                             if (seg.Id != 0)
253                             {
254                                 var matchingWay = mapCopy.Ways[seg.Id];
255                                 var avgSpeed = getAverageSpeed(p.Key, gpxTrackList);
256 
257                                 if (avgSpeed != null)
258                                 {
259                                     if (matchingWay.Tags.ContainsTag("avgSpeed"))
260                                     {
261                                         matchingWay.Tags["avgSpeed"].Value = avgSpeed;
262                                     }
263                                     else
264                                     {
265                                         matchingWay.Tags.Add(new OSMTag("avgSpeed",
                                                avgSpeed));
266                                     }
267 
268                                 }
269 
270                                 if (matchingWay.Tags.ContainsTag("traffic"))
271                                 {
272                                     matchingWay.Tags["traffic"].Value += "," + p.Key;
273                                 }
274                                 else
275                                 {
276                                     matchingWay.Tags.Add(new OSMTag("traffic", p.Key))
                                            ;
277                                 }
278 
279                             }
280                         }
281                         pathList.AddRange(uniquePath);
282                     }
283 
284                     //OSMDB resultMap = reconstructor.SaveToOSM(pathList);
285                     //resultMap.Save("map" + b.Name + ".osm");
286 
287                     mapCopy.Nodes.Add(bucketInfo);
288                     mapCopy.Save("map" + b.Name + ".osm");
289                 }
290             }
291         }
292 
293         //
294         // It gets as parameters the traffic, and a list of gpx tracks list.
295         // Return the avarage speed of the specific traffic, if there are more
296         // than one segment, it calculate the mean.
297         //
298         static string getAverageSpeed(string traffic, List<GPXTrack> gpxTrackList)
299         {
300             double length;
301             double avgSpeed = 0;
302             TimeSpan intervalTime = new TimeSpan();
303 
304             foreach(var trk in gpxTrackList)
305             {
306                 var name = trk.Name.Replace("trk_", "");
```

```
307                 // found the correct segment
308                 if (name.ToString() == traffic)
309                 {
310                     foreach (var seg in trk.Segments)
311                     {
312                         length = 0;
313                         for (int i = 0; i < seg.Nodes.Count; i++)
314                         {
315                             if (i + 1 < seg.Nodes.Count)
316                             {
317                                 length += Calculations.GetDistance2D(seg.Nodes[i], seg
                                        .Nodes[i + 1]);
318                             }
319                         }
320 
321                         // meters to km
322                         length = length / 1000;
323                         if (seg.Nodes.Count != 0)
324                         {
325                             intervalTime = seg.Nodes.Last().Time - seg.Nodes.First().
                                    Time;
326                         }
327                         else
328                         {
329                             avgSpeed = 0;
330                             return avgSpeed.ToString();
331                         }
332 
333                         //Console.WriteLine("Length: " + length);
334                         //Console.WriteLine("Total hours: " + intervalTime.TotalHours)
                                ;
335                         avgSpeed += (length / intervalTime.TotalHours);
336                     }
337 
338                     avgSpeed = avgSpeed / trk.Segments.Count;
339                     return avgSpeed.ToString();
340                 }
341             }
342 
343             return null;
344         }
345 
346         static void GenerateGpxFiles(List<Bucket> buckets, string file, int
                positionFile)
347         {
348 
349             foreach (var b in buckets)
350             {
351                 if (b.Paths.Any())
352                 {
353                     // Saving the Map to GPX instead of OSM - START
354                     var tracks = new List<GPXTrack>();
355 
356                     foreach (var p in b.Paths)
357                     {
358                         GPXTrack track = new GPXTrack(p.Key);
359                         foreach (var t in p.Value)
360                         {
361                             List<GPXPoint> list = new List<GPXPoint>();
362 
363                             foreach (var s in t.Segments)
364                             {
365                                 GPXPoint start = new GPXPoint() { Latitude = s.
                                        StartPoint.Latitude, Longitude = s.StartPoint.
                                        Longitude };
366                                 GPXPoint end = new GPXPoint() { Latitude = s.EndPoint.
                                        Latitude, Longitude = s.EndPoint.Longitude };
367 
368                                 list.Add(start);
369                                 list.Add(end);
370                             }
371 
372                             GPXTrackSegment gpxTrack = new GPXTrackSegment(list);
373                             track.Segments.Add(gpxTrack);
374                         }
375                         tracks.Add(track);
376                     }
377 
378                     var gpx = new GPXDocument() { Tracks = tracks };
379                     gpx.Save("mapGpx" + positionFile + ".gpx");
380                     // END
381                 }
382             }
383         }
384 
```

```
385         /// <summary>
386         /// Prints a help message
387         /// </summary>
388         /// <param name="p">The parameters accepted by this program</param>
389         static void ShowHelp(OptionSet p)
390         {
391             Console.WriteLine("Usage: fdi [OPTIONS]+");
392             Console.WriteLine("Flow Density Inference by matching");
393             Console.WriteLine();
394 
395             Console.WriteLine("Options:");
396             p.WriteOptionDescriptions(Console.Out);
397         }
398     }
399 }
```

Código-fonte 5 – Código-fonte do algoritmo Hot Route by Time Window-HRT

```
using System;
using System.Collections.Generic;
using NDesk.Options;
using LK.OSMUtils.OSMDatabase;
using LK.FDI;
using LK.GPXUtils;
using System.Linq;

namespace LK.HRT
{
    class Program
    {
        static void Main(string[] args)
        {
            string osmPath = "";
            int eps = -1;
            int minTraffic = -1;
            bool showHelp = false;

            OptionSet parameters = new OptionSet() {
                { "osm=", "path to the map file",
                                                                    v => osmPath =
                    v},
                { "eps=",   "size of the eps-neighborhood to be considered (integer)",
                                    v => eps = Convert.ToInt32(v)},
                { "minTraffic=", "minimum traffic considered (integer)",
                                                v => minTraffic = Convert.ToInt32(v)},
                { "h|?|help",

                    v => showHelp = v != null},
            };

            try
            {
                parameters.Parse(args);
            }
            catch (OptionException e)
            {
                Console.Write("TRoute: ");
                Console.WriteLine(e.Message);
                Console.WriteLine("Try HRT --help for more information.");
                return;
            }

            if (showHelp || string.IsNullOrEmpty(osmPath) || eps < 0 || minTraffic <
                0)
            {
                ShowHelp(parameters);
                return;
            }

            var osmFile = new OSMDB();
            osmFile.Load(osmPath);
            var roadGraph = new RoadGraph();
            roadGraph.Build(osmFile);

            // Getting all the nodes with traffic signals
            Dictionary<double, OSMNode> tagsNodes = new Dictionary<double, OSMNode>();
            foreach (var node in osmFile.Nodes)
            {
                foreach (var tag in node.Tags)
                {
                    if (tag.Value.Equals("traffic_signals"))
                    {
                        if (!tagsNodes.Keys.Contains(node.Latitude + node.Longitude))
                        {
                            tagsNodes.Add(node.Latitude + node.Longitude, node);
                        }

                    }
                }
            }

            var hotRoutes = new FlowScan().Run(roadGraph, eps, minTraffic);

            // Saving GPX file of the Hot Route
            HashSet<GPXPoint> listPoints;
            HashSet<GPXTrackSegment> listSegments;
            GPXTrackSegment segTrack;

            List<GPXTrack> track = new List<GPXTrack>();
            GPXTrack tr;
```

```
76
77              //Console.WriteLine(hotRoutes.Count);
78              foreach (var hr in hotRoutes)
79              {
80                  //Console.WriteLine("Number segs: " + hr.Segments.Count);
81                  listSegments = new HashSet<GPXTrackSegment>();
82
83                  foreach (var seg in hr.Segments)
84                  {
85                      listPoints = new HashSet<GPXPoint>();
86
87                      foreach (var segInner in seg.Geometry.Segments)
88                      {
89                          GPXPoint start;
90                          if (tagsNodes.Keys.Contains(segInner.StartPoint.Latitude +
                                segInner.StartPoint.Longitude))
91                          {
92                              OSMNode osmNode = tagsNodes[segInner.StartPoint.Latitude +
                                    segInner.StartPoint.Longitude];
93                              start = new GPXPoint() { Id = osmNode.ID, Latitude =
                                    segInner.StartPoint.Latitude,
94                                  Longitude = segInner.StartPoint.Longitude,
                                        TrafficSignal = true };
95                          } else
96                          {
97                              OSMNode osmNode = osmFile.Nodes.ToList().First(x => x.
                                    Latitude == segInner.StartPoint.Latitude &&
98                                                                                  x.
                                                                                      Longitude
                                                                                      ==
                                                                                      segInner
                                                                                      .
                                                                                      StartPoint
                                                                                      .
                                                                                      Longitude
                                                                                      );
99                              start = new GPXPoint() { Id = osmNode.ID, Latitude =
                                    segInner.StartPoint.Latitude,
100                                 Longitude = segInner.StartPoint.Longitude,
                                        TrafficSignal = false };
101                         }
102
103                         GPXPoint end;
104                         if (tagsNodes.Keys.Contains(segInner.EndPoint.Latitude +
                                segInner.EndPoint.Longitude))
105                         {
106                             OSMNode osmNode = tagsNodes[segInner.EndPoint.Latitude +
                                    segInner.EndPoint.Longitude];
107                             end = new GPXPoint() { Id = osmNode.ID, Latitude =
                                    segInner.EndPoint.Latitude,
108                                 Longitude = segInner.EndPoint.Longitude, TrafficSignal
                                        = true };
109                         } else
110                         {
111                             OSMNode osmNode = osmFile.Nodes.ToList().First(x => x.
                                    Latitude == segInner.EndPoint.Latitude &&
112                                                                                 x.
                                                                                     Longitude
                                                                                     ==
                                                                                     segInner
                                                                                     .
                                                                                     EndPoint
                                                                                     .
                                                                                     Longitude
                                                                                     );
113                             end = new GPXPoint() { Id = osmNode.ID, Latitude =
                                    segInner.EndPoint.Latitude,
114                                 Longitude = segInner.EndPoint.Longitude, TrafficSignal
                                        = false };
115                         }
116
117                         listPoints.Add(start);
118                         listPoints.Add(end);
119                     }
120
121                     segTrack = new GPXTrackSegment(listPoints, seg.AvgSpeed, seg.Speed
                            , seg.Id);
122                     // passing the traffic
123                     segTrack.Traffic = seg.Traffic;
124                     listSegments.Add(segTrack);
125                 }
126
127                 tr = new GPXTrack();
```

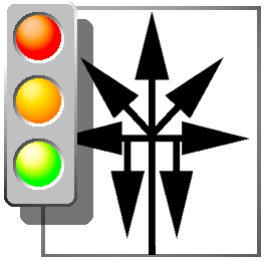
```
128                     tr.Segments.AddRange(listSegments);
129                     track.Add(tr);
130
131                 }
132
133                 // Bucket Information
134                 GPXTrack tBucket = new GPXTrack();
135                 GPXPoint pBucket = new GPXPoint(0, 0, 0, false);
136                 GPXTrackSegment sBucket = new GPXTrackSegment();
137
138                 var bucketInfo = osmFile.Nodes.ToList().Find(x => x.ID == 0);
139
140                 if (bucketInfo != null)
141                 {
142                     pBucket.StartBucket = TimeSpan.Parse(bucketInfo.Tags.First().Value);
143                     pBucket.EndBucket = TimeSpan.Parse(bucketInfo.Tags.Last().Value);
144                 }
145
146                 sBucket.Nodes.Add(pBucket);
147                 tBucket.Segments.Add(sBucket);
148
149                 var gpx = new GPXDocument() { Tracks = track };
150                 gpx.Tracks.Add(tBucket);
151                 gpx.Save("mapWithHotRoutes.gpx");
152
153             }
154
155             /// <summary>
156             /// Prints a help message
157             /// </summary>
158             /// <param name="p">The parameters accepted by this program</param>
159             static void ShowHelp(OptionSet p)
160             {
161                 Console.WriteLine("Usage: TRoute [OPTIONS]+");
162                 Console.WriteLine("Outputs hot routes found");
163                 Console.WriteLine();
164
165                 Console.WriteLine("Options:");
166                 p.WriteOptionDescriptions(Console.Out);
167             }
168         }
169 }
```

Código-fonte 6 – Código-fonte do algoritmo GreenRoute-GR

```
using NDesk.Options;
using System;
using Google.OrTools.ConstraintSolver;
using System.Linq;
using System.Collections.Generic;
using LK.GPXUtils;

namespace LK.GreenRoute
{
    class Program
    {
        static void Main(string[] args)
        {

            string gpxPath = "";
            bool showHelp = false;


            OptionSet parameters = new OptionSet() {
                { "gpx=", "path to the map file",
                                                                    v => gpxPath =
                    v},
                { "h|?|help",
                    v => showHelp = v != null},
            };

            try
            {
                parameters.Parse(args);
            }
            catch (OptionException e)
            {
                Console.Write("TRoute: ");
                Console.WriteLine(e.Message);
                Console.WriteLine("Try GreeRoute --help for more information.");
                return;
            }

            if (showHelp || string.IsNullOrEmpty(gpxPath))
            {
                ShowHelp(parameters);
                return;
            }

            HRDocument hr = new HRDocument();
            hr.Load(gpxPath);
            hr.Webster();

            GPXPoint bucketInfo = null;
            foreach (var track in hr.Tracks)
            {
                foreach (var seg in track.Segments)
                {
                    bucketInfo = seg.Nodes.ToList().Find(x => x.Id == 0);
                }
            }

            Solve(hr.getListProcessor(), hr.getJobTime(), hr.getJobId(), bucketInfo,
                hr.cycleTime);
        }

        /// <summary>
        /// Prints a help message
        /// </summary>
        /// <param name="p">The parameters accepted by this program</param>
        static void ShowHelp(OptionSet p)
        {
            Console.WriteLine("Usage: TRoute [OPTIONS]+");
            Console.WriteLine("Outputs hot routes found");
            Console.WriteLine();

            Console.WriteLine("Options:");
            p.WriteOptionDescriptions(Console.Out);
        }

        static void Solve(List<List<int>> machines, List<List<int>> processingTimes,
            List<List<int>> jobIds, GPXPoint bucketInfo, int cycleTime)
        {
            Console.WriteLine("\n---- Job shop Scheduling Program ----");
            JobShop jobShop = new JobShop(machines, processingTimes, jobIds,
                bucketInfo, cycleTime);
```

```
76                  jobShop.RunJobShopScheduling("Jobshop");
77              }
78          }
79
80
81  }
```

ANEXO A – SIMULATION OF URBAN MOBILITY



SUMO

Simulation of Urban MObility

Sumo at a Glance

generated on 2018-02-22 00:24:57.623159 from **the wiki page for Sumo_at_a_Glance** for SUMO git

Contents

About

"**S**imulation of **U**rban **MO**bility", or "SUMO" for short, is an open source, microscopic, multi-modal traffic simulation. It allows to simulate how a given traffic demand which consists of single vehicles moves through a given road network. The simulation allows to address a large set of traffic management topics. It is purely microscopic: each vehicle is modelled explicitly, has an own route, and moves individually through the network. Simulations are deterministic by default but there are various options for **introducing randomness**.

If you download the SUMO package, you will note that it contains further applications besides SUMO. These applications are used to import/prepare road networks and demand data for being used in SUMO, see **#Included Applications** for a more verbose list.

Features

- Includes all applications needed to prepare and perform a traffic simulation (network and routes import, DUA, simulation)
- Simulation
  - Space-continuous and time-discrete vehicle movement
  - Different vehicle types
  - Multi-lane streets with lane changing
  - Different right-of-way rules, traffic lights
  - A fast openGL graphical user interface
  - Manages networks with several 10.000 edges (streets)
  - Fast execution speed (up to 100.000 vehicle updates/s on a 1GHz machine)
  - Interoperability with other application at run-time
  - Network-wide, edge-based, vehicle-based, and detector-based outputs
  - Supports person-based inter-modal trips
- Network Import
  - Imports VISUM, Vissim, Shapefiles, OSM, RoboCup, MATsim, OpenDRIVE, and XML-Descriptions
  - Missing values are determined via heuristics

- Routing
  - Microscopic routes - each vehicle has an own one
  - Different Dynamic User Assignment algorithms
- High portability
  - Only standard C++ and portable libraries are used
  - Packages for Windows main Linux distributions exist
- High interoperability through usage of XML-data only
- Open source (**EPL**)

Usage Examples

Since 2001, the SUMO package has been used in the context of several national and international research **projects**. The applications included:

- traffic lights evaluation
- route choice and re-routing
- evaluation of traffic surveillance methods
- **simulation of vehicular communications**
- traffic forecast

Included Applications

The package includes:

Application Name	Short Description
SUMO	The microscopic simulation with no visualization; command line application
SUMO-GUI	The microscopic simulation with a graphical user interface
NETCONVERT	Network importer and generator; reads road networks from different formats and converts them into the SUMO-format
NETEDIT	A graphical network editor.
NETGENERATE	Generates abstract networks for the SUMO-simulation
DUAROUTER	Computes fastest routes through the network, importing different types of demand description. Performs the DUA
JTRROUTER	Computes routes using junction turning percentages
DFROUTER	Computes routes from induction loop measurements
MAROUTER	Performs macroscopic assignment
OD2TRIPS	Decomposes O/D-matrices into single vehicle trips
POLYCONVERT	Imports points of interest and polygons from different formats and translates them into a description that may be visualized by **SUMO-GUI**
ACTIVITYGEN	Generates a demand based on mobility wishes

	of a modeled population
EMISSIONSMAP	Generates an emission map
EMISSIONSDRIVINGCYCLE	Calculates emission values based on a given driving cycle
Additional Tools	There are some tasks for which writing a large application is not necessary. Several solutions for different problems may be covered by these tools.

Several parties have extended the SUMO package during their work and submitted **their code**. If not being a part of the release, these contributions are usually not tested frequently and may be outdated. The following contributions are included, please note that not all of them are longer maintained.

Application Name	Short Description
TraCI4J	A Java interface for connecting and extending information via **TraCI** by Enrico Gueli
TraCI4Matlab	A Matlab interface for connecting and extending information via **TraCI** Andres Acosta
TraaS	A SOAP(webservice) interface for connecting and extending information via **TraCIby** Mario Krumnow
~~**Contributed/SUMO Traffic Modeler**~~	~~A Java application for generating a demand based on mobility wishes by Leontios Papaleontiou~~ (obsolete)

History

The development of SUMO started in the year 2000. The major reason for the development of an open source, microscopic road traffic simulation was to support the traffic research community with a tool with the ability to implement and evaluate own algorithms. The tool has no need for regarding all the needed things for obtaining a complete traffic simulation such as implementing and/or setting up methods for dealing with road networks, demand, and traffic controls. By supplying such a tool, the DLR wanted to i) make the implemented algorithms more comparable by using a common architecture and model base, and ii) gain additional help from other contributors.

Software design criteria

Two major design goals are approached: the software shall be fast and it shall be portable. Due to this, the very first versions were developed to be run from the command line only - no graphical interface was supplied at first and all parameter had to be inserted by hand. This should increase the execution speed by leaving off slow visualisation. Also, due to these goals, the software was split into several parts. Each of them has a certain purpose and must be run individually. This is something that makes SUMO different to other simulation packages where, for instance, the dynamical user assignment is made within the simulation itself, not via an external application like here. This split allows an easier extension of each of the applications within the package because each is smaller than a monolithic application that does everything. Also, it allows the usage of faster data structures, each adjusted to the current purpose, instead of using complicated and ballast-loaded ones. Still, this makes the usage of SUMO a little bit uncomfortable in comparison to other simulation packages. As there are still other things to do, we are not thinking of a redesign towards an integrated approach by now.

Contributors and Participants

Org.	Name	Topics / Contribution
zaik	Christian Rössel	Initial microsimulation core; initial detectors implementation
DLR	Peter Wagner	Models, organisation, spiritual lead
	Daniel Krajzewicz	Everything
	Julia Ringel	Traffic Light & WAUT Algorithms
	Eric Nicolay	Everything
	Michael Behrisch	Everything
	Yun-Pang Wang	User Assignment
	Danilot Teta Boyom	Vehicular Communication Model (removed from the source)
	Sascha Krieg	
	Lena Kalleske	
	Laura Bieker	Tests, Python scripts
	Jakob Erdmann	network import, **NETEDIT**
	Andreas Gaubatz	
	Maik Drozdzynski	
Uni Lübeck	Axel Wegener	TraCI initiator
	Thimor Bohn	TraCI
	Friedemann Wesner	TraCI
	Felix Brack	
	Tino Morenz	
Friedrich-Alexander-Universität Erlangen-Nürnberg	Christoph Sommer	TraCI merge with **Veins**, Subscription Interface, Misc.
universität innsbruck	David Eckhoff	TraCI, deterministic simulation behavior
Carnegie Mellon University	Falko Dressler	TraCI
UCLA	Tobias Mayer	Traffic model abstraction, IDM model port
HU Berlin	Matthias Heppner	Unittests
Technische Universität München	Piotr Woznica	**ACTIVITYGEN**
	Walter	Development of

	Bamberger	**ACTIVITYGEN** as a base for the evaluation of trust scenarios in VANETs. The work is part of the project **Fidens: Trust between Cooperative Systems** featuring trusted probabilistic knowledge processing in vehicular networks.
	Matthew Fullerton	
IIT Bombay, India	Ashutosh Bajpai	randomDepart.py, a python script to generate the real traffic pattern by exponential Distribution.
POLITECNICO DI TORINO	Enrico Gueli	**TraCI4J**
	Leontios Papaleontiou	**Contributed/SUMO Traffic Modeler**
Uniwersytet Wrocławski	Karol Stosiek	Documentation, network building

This page was last modified on 2 October 2017, at 22:32.