Um Algoritmo Eficiente de Detecção de Trajetórias Baseado em Aceleração de Hardware

Fernando D. M. Silva, Pedro Cruz e Luís Henrique M. K. Costa

[1]GTA - PEE/COPPE/DEL/Poli
Universidade Federal do Rio de Janeiro (UFRJ)

{fernandodias,cruz,luish}@gta.ufrj.br

Resumo. *Em cidades inteligentes, aplicações de monitoramento de frotas veiculares encontram problemas de classificação de trajetórias, devido a problemas variados como limitações dos dispositivos e falhas de comunicação ou de configuração. Uma solução frequentemente usada é a classificação em nuvem dos dados de trajetória coletados, feita com o uso algoritmos de similaridade de trajetórias. Porém, algoritmos como o Dynamic Time Warping (DTW) e o Longest Common Subsequence (LCSS) podem se tornar custosos em cenários com grande volumes de dados. Assim, neste artigo é proposto o Unordered and Vectorized Common Segment Ratio Classifier (UV-CSRC), que propõe classificar segmentos de trajetórias com o uso de aceleração de hardware. Através de testes com dados reais e sintéticos demonstra-se que o algoritmo, em cenários reais, obtém uma acurácia semelhante ao do DTW e do LCSS, com tempo de processamento ao menos 28 vezes mais rápido.*

Abstract. *In a smart city environment, transportation systems monitoring can face issues when classifying their trajectories. Communication failures and faulty devices are some of the issues that affect trajectory classification. Those issues can be partly solved using similarity measure algorithms in the cloud. Such algorithms like Dynamic Time Warping (DTW) and Longest Common SubSequence (LCSS) are not fit to handle large amounts of data. In this paper, we propose the Unordered and Vectorized Common Segment Ratio Classifier (UV-CSRC), an algorithm designed to classify trajectories. This algorithm is assisted by hardware acceleration and can process data at least 28 times faster than the other algorithms. Tests were done with real and synthetic data that shows the algorithm's accuracy matching the others when applied in real-world scenarios.*

1. Introdução

A Internet das Coisas (*Internet of Things* – IoT) permite que cada vez mais objetos possuam capacidades de sensoriamento, atuação, processamento, comunicação e armazenamento. Assim, objetos são transformados em dispositivos, podendo interagir com o ambiente e entre si. Isso permite que aplicações de rastreamento de pessoas ou veículos possam se tornar mais complexas em termos de técnicas de coleta e volume de dados. A conectividade que se faz em um ambiente urbano permite que aplicações possam ser implementadas em larga escala e com baixo custo de equipamento, como por exemplo em [Farooq et al. 2010]. Porém, para manter o baixo custo dos dispositivos, perde-se a capacidade de precisão e correção própria dos aparelhos, fazendo com que erros se tornem cada vez mais evidentes com o crescimento do volume de dados.

No cenário urbano, a integração do sistema de transporte público permite que as frotas possam ser monitoradas e até mesmo multadas por descumprimento das condições de

operação[1]. Porém, os equipamentos de baixo custo podem inserir imprecisões inaceitáveis nesse contexto. Juntamente com falhas humanas, as imprecisões podem prejudicar a acurácia da classificação desses dados. Assim, análises nos dados se tornam mais difíceis de serem realizadas. Pode-se fazer o uso de algoritmos que classifiquem os dados em nuvem após coleta, com o objetivo de corrigir os dados para análises históricas mais precisas.

Esse trabalho propõe um novo algoritmo para a classificação de trajetórias, o *Unordered and Vectorized Common Segment Ratio Classifier* (UV-CSRC). O algoritmo é capaz de segmentar e classificar os trechos de uma trajetória, descobrindo as linhas que o ônibus está percorrendo. Essa classificação utiliza aceleração por hardware, diminuindo consideravelmente seu tempo de execução quando comparado a dois outros algoritmos da literatura. Junto ao teste de desempenho, são feitos testes com dados reais e sintéticos que mostram que a acurácia do UV-CSRC é próxima à dos outros em cenários reais em uma aplicação no ambiente urbano.

O trabalho também evidencia a vantagem da aplicação do algoritmo no ambiente de transporte urbano. Para fazer o teste com as trajetórias sintéticas, é feito um estudo com os dados da posição dos ônibus da cidade do Rio de Janeiro. Neles são observadas as características comuns aos dados reais e estas são replicadas na construção das trajetórias sintéticas. O teste evidencia uma vantagem do UV-CSRC em comparação aos outros algoritmos, não necessitando de etapas auxiliares para separar trajetórias e como isso impacta nos resultados. O UV-CSRC consegue ser executado ao menos 28 vezes mais rápido que os outros algoritmos testados e mantém a acurácia dos mesmos em cenários de trajetórias em ambientes urbanos, com apenas um viés de diferença.

Este artigo está organizado da seguinte forma. A Seção 2 apresenta trabalhos relacionados da literatura. A Seção 3 discute o problema de reconhecimento de trajetórias. A Seção 4 discute os algoritmos de similaridade de trajetórias usados no reconhecimento. A Seção 5 descreve em detalhes o funcionamento do algoritmo proposto. A Seção 6 discute como é feita a criação dos dados de trajetória sintéticos a serem utilizados nos testes. A Seção 7 compara o desempenho do algoritmo proposto com a literatura, enquanto a Seção 8 conclui o artigo.

2. Trabalhos Relacionados

É possível encontrar na literatura científica trabalhos que mostrem que os dados de localização de veículos podem conter erros. Os trabalhos encontrados normalmente se esforçam para detectar erros na descrição da trajetória, buscando reconstruí-la a partir de uma sequência de coordenadas [Silva et al. 2020, Cruz et al. 2020, Dias e Costa 2016, Neves et al. 2018]. No presente trabalho, busca-se, a partir da trajetória de um ônibus, descobrir as linhas que podem estar sendo percorridas pelo ônibus. Dessa forma, é possível descobrir discrepâncias entre o trajeto de um ônibus e a linha à qual o ônibus está designado.

O trabalho de Zheng mostra uma visão geral sobre o problema de análise de dados de trajetória [Zheng 2015]. Zheng elenca as diferentes etapas e possíveis processos que dados de trajetória podem passar uma vez coletados, além de elencar inúmeros processos que podem ser aplicados em cada contexto. No trabalho são elencados procedimentos de filtragem de ruído, compressão das informações, indexação de trajetórias com o uso de aprendizado de máquina, entre outros. O presente trabalho apresenta um novo algoritmo para a classificação de trajetórias e compara esse algoritmo com dois outros algoritmos na literatura, complementando a literatura estudada por Zheng.

[1]https://prefeitura.rio/transportes/prefeitura-podera-multar-consorcios-por-meio-dos-dados-de-gps-dos-onibus/.

No trabalho de Su *et al.*, é feita uma pesquisa dos mais diversos tipos de métricas utilizadas no problema de similaridade entre trajetórias [Su et al. 2020]. O artigo dá uma visão geral de diferentes algoritmos que servem para trajetórias e faz testes comparativos para mostrar o comportamento dos algoritmos quando os dados de testes apresentam tipos específicos de erros. Este trabalho mostra o panorama de algoritmos de similaridade entre trajetórias e auxilia na escolha dos algoritmos mais apropriados para o caso de classificação das linhas de ônibus, e quais seriam selecionados para comparação com o UV-CSRC.

O trabalho de Bejan *et al.* mostra a aplicação da detecção de trajetórias para a estimação do tempo de jornada de cada linha, apresentando as técnicas utilizadas para operar com dados de monitoramento esparsos [Bejan et al. 2010]. No trabalho é proposta a utilização de um algoritmo e os resultados obtidos foram validados com o uso de dispositivos de monitoramento de maior resolução.

Os trabalhos da literatura mostram a importância de sistemas de transportes inteligentes e como aplicações podem ser feitas de modo a fornecer aplicações que vão além do monitoramento da frota. O trabalho de Cruz *et al.* mostra como a rede de ônibus urbanos serve de infraestrutura para o sensoriamento de uma cidade com o custo menor devido a mobilidade de sensores individuais, e aplica isso à uma ferramenta própria [Cruz Caminha et al. 2018].

Determinar a similaridade entre trajetórias é uma tarefa importante para poder classificá-las. Os algoritmos *Dynamic Time Warping* (DTW) [Müller 2007] e o *Longest Common Subsequence* (LCSS) [Vlachos et al. 2002] são algoritmos importantes nesse sentido. Na Seção 4, ambos são discutidos em detalhes e comparados com o UV-CSRC.

Ferramentas para o monitoramento do sistema de transporte integram os elementos de uma rede de transportes inteligentes em uma plataforma para fácil manipulação das informações. Este trabalho utiliza a ferramenta FAS-Bus (Fleet Analysis System for Urban Buses) [Silva et al. 2021] para fazer a manipulação dos dados utilizados no trabalho. O FAS-Bus é uma ferramenta que executa o processo de coleta, correção e visualização de dados da frota de ônibus da cidade do Rio de Janeiro. Essa ferramenta é utilizada como base para a implementação e teste dos algoritmos avaliados neste trabalho.

3. Reconhecimento de Trajetórias de Ônibus Urbanos

O contexto do trabalho começa com a frota da cidade do Rio de Janeiro que possui um exemplo de aplicação de IoT: cada ônibus do transporte público da cidade possui um dispositivo que monitora a localização do ônibus e a envia periodicamente a um servidor da prefeitura. Esses dados podem ser obtidos diretamente pelo site do DataRio[2] ou visualizados depois de interpretados no site da secretaria municipal de transportes[3]. O sistema auxilia no monitoramento da frota, caracterização do sistema de transporte e aplicação de multas. Esses dados são coletados do servidor e armazenados desde 2018 e contém em cada entrada informações sobre o tempo, identificador do ônibus, localização em latitude e longitude, e linha percorrida. Pode-se visualizar um exemplo dos dados existentes na Tabela 1.

Para fazer o reconhecimento de trajetórias, são utilizados algoritmos que oferecem uma medida comparativa entre o grau de similaridade de dois caminhos distintos. Essas medidas, chamadas de similaridade (ou distância) entre trajetórias, recebem dois trajetos e retornam um valor que indica o quão parecidas são as duas trajetórias. Detalhes sobre esses algoritmos são descritos na Seção 4.

[2]https://www.data.rio/documents/PCRJ::gps-dos-%C3%B4nibus/about

[3]http://www.rio.rj.gov.br/web/smtr/exibenoticias?id=12350171

Essas medidas somente não bastam para fazer o reconhecimento de trajetórias no cenário de ônibus urbanos. Para fazer esse reconhecimento, faz-se necessário identificar quais trechos ao longo do dia o ônibus efetivamente percorria uma linha e separar outros momentos como o ônibus estacionado ou seu translado entre início da linha e a garagem. Nesse processo os subconjuntos da trajetória são separados para serem utilizados nos algoritmos de similaridade que comparam esses segmentos com as trajetórias das linhas percorridas pelos ônibus. Essa separação podes ser feita em função da velocidade média entre um ponto e o anterior para delimitar os segmentos em movimento do ônibus.

Portanto, a classificação de trajetórias no cenário urbano se dá nas seguintes etapas: Faz-se a divisão da trajetória em segmentos que contém apenas os trechos em que o ônibus se move e, em seguida, classifica-se cada um dos segmentos da trajetória formados. Essa classificação consiste no uso do algoritmo de similaridade entre trajetórias, comparando o segmento do trajeto com cada uma das trajetórias das linhas que se quer encontrar.

4. Algoritmos de Similaridade entre Trajetórias

Um algoritmo de similaridade ou distância entre trajetórias retorna uma grandeza que representa o quão próximo são duas trajetórias em suas características de traçado e pontos de inicio e fim. No contexto mais simples para a obtenção desse valor, consideramos duas trajetórias $A = \{a_i \mid i \in [0, n]\}$ e $B = \{b_i \mid i \in [0, n]\}$, onde n é o tamanho de ambas. Podemos obter uma métrica de distância e fizermos a média de todas as distâncias entre os pontos i de cada trajetória. Assim temos:

$$Euclidiana(A, B) = \frac{1}{n} \sum_{i=0}^{n} d(a_i, b_i). \tag{1}$$

Esta métrica é chamada de distância Euclidiana e é a mais simples e direta. Porém, ela é susceptível a variações no resultado decorrente de diferentes valores de amostragem, perda de dados, ruído na coleta, dentre outros fatores elencados em [Su et al. 2020]. Outros algoritmos são considerados para o cálculo de distância entre trajetórias de modo a compensar essas imprecisões que são comuns em dados reais. Na Equação 1, temos que a função $d(a_i, b_i)$ representa uma função de distância entre os dois pontos. Essa função é a operação primordial do algoritmo e é utilizada em todos os demais. No contexto de coordenadas terrestres, temos que essa função é a fórmula de haversine, que calcula a distância entre dois pontos em uma superfície esférica (uma aproximação da Terra).

No contexto desse trabalho, existem dois algoritmos conhecidos mais apropriados para o problema: O *Dynamic Time Warping* (DTW) [Müller 2007] e o *Longest Common SubSequence* (LCSS) [Vlachos et al. 2002]. Estes foram escolhidos por serem bem conhecidos da literatura e

Data/hora de detecção	**Identificador**	**Latitude** [°]	**Longitude** [°]	**Linha**
2019-03-18 20:18:27	A27506	-22.916090	-43.251171	433
2019-03-18 20:19:27	A27506	-22.916401	-43.252811	433
2019-03-18 20:20:27	A27506	-22.916651	-43.254009	433
2019-03-18 23:54:32	D87894	-22.913639	-43.596649	2339
2019-03-18 23:55:32	D87894	-22.914129	-43.601559	2339
2019-03-18 23:56:32	D87894	-22.914869	-43.605289	2339

Tabela 1: Exemplo de dados armazenados na ferramenta FAS-Bus.

mostrarem características de resistência a variações de velocidade e ruídos nos dados respectivamente, como visto em [Su et al. 2020]. Ambos também são destinados a trajetórias discretas e sem necessidade de informação temporal (mas sim sequencial), o mesmo contexto no qual os dados desse trabalho se apresentam.

4.1. Dynamic Time Warping

O DTW é um algoritmo que retorna um valor absoluto de distância entre duas trajetórias. Esse valor pode ser utilizado de modo comparativo para determinar qual das trajetórias é mais parecida com uma trajetória de referência. Esse algoritmo tem como propriedade ser mais resistente à variações de velocidade e tempo de amostragem.

O cálculo de seu valor é feito com o somatório das distâncias entre os pares de pontos entre as duas trajetórias comparadas. Porém, os pares de pontos que são calculadas as distâncias são escolhidos de forma a retornar os menores valores de distância por comparação. A implementação completa com programação dinâmica pode ser visualizada no algoritmo 1, com o cálculo melhor ilustrado na linha 14.

Algoritmo 1: Dynamic Time Warping (DTW)

Entrada: $TrajetóriaA$,$TrajetóriaB$
Resultado: $distância$
/* As trajetórias podem possuir tamanhos diferentes */
1 $n = tamanho(TrajetóriaA)$;
2 $m = tamanho(TrajetóriaB)$;
/* Retorna uma matriz de tamanho nxm que pode ter o elemento A_{ij} acessado por MatrizResultado[i,j] */
3 $MatrizResultado = Matriz(n, m)$;
/* Inicialização da matriz */
4 **para** *i entre 0 e* $n - 1$ **faça**
5 | $MatrizResultado[i, 0] = \infty$;
6 **fim**
7 **para** *i entre 0 e* $m - 1$ **faça**
8 | $MatrizResultado[0, i] = \infty$;
9 **fim**
10 MatrizResultado[0,0] = 0;
/* Cálculo do DTW */
11 **para** *i entre 0 e* $n - 1$ **faça**
12 | **para** *j entre 0 e* $m - 1$ **faça**
13 | | custo = $d(TrajetóriaA[i], TrajetóriaB[j])$;
14 | | $MatrizResultado[i, j] = custo + Menor(MatrizResultado[i + 1, j],$
| | $MatrizResultado[i, j + 1], MatrizResultado[i + 1, j + 1])$;
15 | **fim**
16 **fim**
17 **retorna** $MatrizResultado[n - 1, m - 1]$

O DTW ajusta seus resultados com base na variação da posição dos pontos, um reflexo da variação tanto da velocidade quanto do tempo de amostragem. Porém, esse método não é robusto à ruídos, e cada ponto que se apresenta distante da trajetória comparada tem um impacto significativo no resultado final.

4.2. Longest Common SubSequence

O LCSS é o segundo algoritmo analisado. A saída do algoritmo LCSS é um valor percentual que indica uma razão de quão parecidas são duas trajetórias. Seu conceito é o de encontrar a maior subsequência comum a ambas as trajetórias. Ao contrário dos outros métodos vistos

até aqui, este é parametrizado e tem seu resultado diretamente impactado por um parâmetro de tolerância. Esse parâmetro é uma distância máxima que os pontos comparados devem estar para serem considerados como iguais no contexto de encontrar elementos comuns entre as trajetórias.

Seu cálculo é feito comparando a distância entre todos os pares de pontos. Aqueles pontos que estiverem a uma distância menor do que a tolerância definida são considerados comuns. É contabilizado entre as duas trajetórias o tamanho das sequências contínuas de pontos marcados como comuns e o resultado da maior subsequência nos dá a razão de subsequência. Esse processo está representado no algoritmo 2, a comparação com o valor de tolerância na linha 12 e o resultado em razão de similaridade na linha 19. O resultado final é calculado de forma que, quanto maior a razão, mais parecidas são as trajetórias.

Algoritmo 2: Least Common SubSequence (LCSS)

Entrada: $TrajetóriaA$,$TrajetóriaB$,$DistânciaDeTolerância$
Resultado: $Razão$
/* As trajetórias podem possuir tamanhos diferentes */
1 $n = tamanho(TrajetóriaA)$;
2 $m = tamanho(TrajetóriaB)$;
/* Retorna uma matriz de tamanho nxm que pode ter o elemento A_{ij} acessado por MatrizResultado[i,j] */
3 $MatrizResultado = Matriz(n, m)$;
/* Inicialização da matriz */
4 **para** *i entre 0 e* $n - 1$ **faça**
5 | $MatrizResultado[i, 0] = 0$;
6 **fim**
7 **para** *i entre 0 e* $m - 1$ **faça**
8 | $MatrizResultado[0, i] = 0$;
9 **fim**
/* Cálculo do LCSS */
10 **para** *i entre 0 e* $n - 1$ **faça**
11 | **para** *j entre 0 e* $m - 1$ **faça**
12 | | **se** $Distância(A[i], B[j]) < DistânciaDeTolerância$ **então**
13 | | | $MatrizResultado[i, j] = MatrizResultado[i - 1, j - 1] + 1$;
14 | | **senão**
15 | | | $MatrizResultado[i, j] = Maior(MatrizResultado[i - 1, j]$, $MatrizResultado[i, j - 1]$;
16 | | **fim**
17 | **fim**
18 **fim**
19 **retorna** $MatrizResultado[n - 1, m - 1]/(n + m - MatrizResultado[n - 1, m - 1])$

Como o valor de distância não impacta no resultado final, essa característica de detecção em função do valor de tolerância torna o algoritmo robusto à ruído. Uma desvantagem é que quanto menor são as trajetórias avaliadas maior é a variação do erro no resultado final. Para manter a comparação válida entre diferentes linhas, faz-se necessário manter o mesmo parâmetro durante todo o processo de classificação.

5. *Unordered and Vectorized Common Segment Ratio Classifier*

A proposta do algoritmo *Unordered and Vectorized Common Segment Ratio Classifier* (UV-CSRC) é classificar subsequências de coordenadas em uma trajetória. Essa classificação consiste na identificação de trechos no qual a trajetória percorre um caminho pré-estabelecido. No contexto dos ônibus, faz-se a identificação na trajetória completa de um ônibus ao longo do dia quais linhas ele percorreu naquele dia e em que momento. Ao contrário do DTW

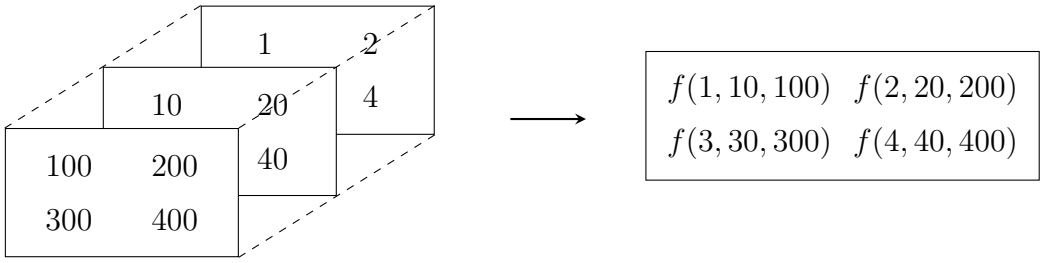


Figura 1: Exemplo da operação de redução.

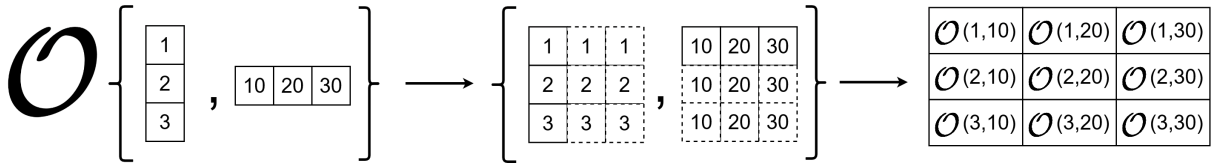


Figura 2: Exemplo da operação de broadcasting.

e do LCSS, o algoritmo não necessita de uma etapa anterior para fazer a divisão da trajetória em segmentos.

Uma das vantagens do UV-CSRC em comparação com os demais é a adoção de técnicas de programação vetorizada com a intenção de execução desse algoritmo em unidades de processamento gráfico (GPU). As etapas de maior intensidade computacional do algoritmo são executadas com o auxílio da GPU e assim milhares das operações são executadas de forma simultânea, com todos os dados disponíveis na própria memória da unidade.

Duas operações no processo de programação vetorizada são importantes no contexto desse algoritmo: redução de dimensionalidade e *broadcasting*. A redução de dimensionalidade (chamada apenas de redução daqui em diante) é o processo que seleciona todos os dados referentes a uma dimensão e aplica uma função que retorna um resultado numérico. Ambas operações são executadas no contexto da GPU. Podemos ver o exemplo com uma matriz tridimensional na Figura 1.

O *broadcasting* é uma operação que repete valores ao longo de uma dimensão para que os dois operandos possuam as mesmas dimensões e assim possa ser aplicada uma operação termo a termo. Se um vetor coluna é somado com uma matriz, o *broadcasting* permite que a operação seja realizada ao repetir o vetor em linhas até o tamanho da matriz a ser somada. Esse processo pode ser visualizado na Figura 2.

O algoritmo possui parte do seu desenvolvimento escrito de forma matricial. A operação de *broadcasting* nesse contexto auxilia operações que devem ser feitas com o produto cartesiano de dois conjuntos de dados. Assim, o algoritmo pode fazer a classificação entre todas as linhas e todos os ônibus em uma única execução, sem precisar repetir a execução do algoritmo por ônibus e por linha.

O algoritmo possui duas etapas. Primeiro é executada a etapa de detecção, que relaciona a cada ônibus as linhas que provavelmente foram percorridas por ele. Depois, a etapa de correção, que busca mitigar os erros cometidos na etapa de detecção. Nas seções seguintes, cada etapa é descrita em mais detalhes.

5.1. Etapa de detecção

Primeiramente os dados são formatados para obtenção de duas matrizes que armazenam as coordenadas de todas as trajetórias analisadas. São elas: $\mathbf{O}_{n_O \times m_O \times 2}$, onde n_O é a quantidade

de ônibus, m_O é a quantidade de coordenadas que possui a trajetória mais longa dos ônibus, e a última dimensão refere-se aos valores de latitude e longitude. Temos de forma análoga uma matriz $\mathbf{L}$ para as linhas, com dimensões variáveis n_L e m_L. No caso de linhas ou ônibus que contém uma quantidade de coordenadas menor que m_L ou m_O, usa-se o valor padrão de *Not a Number (NaN)*[4], que é desconsiderado nas operações e no cálculo dos resultados.

O próximo passo consiste em um *broadcasting* com o uso da fórmula de haversine entre essas matrizes para a criação de uma matriz de distâncias $\mathbf{D}^{[all]}_{n_O \times m_O \times n_L \times m_L}$. Vemos que, pelas dimensões da matriz, é possível localizar na matriz a distância entre qualquer coordenada de qualquer ônibus e qualquer coordenada de qualquer linha. Vale notar que essa etapa é a mais custosa em termos de memória e para isso deve ser executada em etapas para que a memória limitada das placas possua espaço o suficiente.

Queremos em seguida descobrir, para uma linha, qual a coordenada do ônibus que possui a menor distância até as coordenadas das linhas. Pra isso, faz-se uma redução na dimensão m_O em que o resultado é uma nova matriz em que seus valores correspondem a menor distância das coordenadas de um ônibus. Isso nos dá uma nova matriz $\mathbf{D}^{[min]}_{n_O \times n_L \times m_L}$ com esse resultado para cada coordenada de todas as linhas, para todos os ônibus.

Essa matriz é reduzida uma última vez na dimensão de m_L de modo similar ao que é visto no LCSS: com base em um valor de tolerância, determina-se se os valores de $\mathbf{D}^{[min]}$ são menores que esse valor e transforma-se a matriz numérica para uma matriz lógica. Conta-se a quantidade de resultados verdadeiros ao longo de m_L e divide-se esse valor pelo tamanho da linha respectiva. Vemos que esse passo tem duas diferenças cruciais ao LCSS: A razão não contabiliza o tamanho das trajetórias dos ônibus e essas operações não consideram a ordem na qual cada coordenada aparece.

O resultado da redução é a matriz de detecção $\mathbf{D}_{n_O \times n_L}$, que possui a razão entre coordenadas da linha que foi detectada a presença do ônibus e o tamanho da linha. Diversas operações podem extrair dessa matriz uma relação entre possíveis linhas que cada ônibus possa ter percorrido, tais como as 3 porcentagens mais altas ou todas que são maiores que um valor de corte. A primeira abordagem provou-se mais efetiva na detecção, acompanhada de uma razão de corte para valores muito baixos.

A desvantagem da etapa de detecção é que, graças à análise ser sem ordem, o algoritmo possui uma possibilidade maior de falsos positivos, decorrentes do acúmulo de pequenos trechos em diferentes intervalos de tempo que unidos podem compor uma terceira trajetória não percorrida. Para mitigar o efeito dessas desvantagens, cada trajetória passa por um processo de correção que visa corrigir cenários mais comuns de falsos positivos.

5.2. Correção

Para o processo de correção, é feita uma iteração para cada ônibus que conseguiu pelo menos uma identificação na matriz $\mathbf{D}$. Para cada ônibus, cria-se uma lista de listas para cada linha detectada em sua trajetória. Essas listas têm o mesmo tamanho da trajetória do ônibus atual. Para preenchê-las é feito um processo similar à etapa que calcula $D^{[min]}$ na detecção. Procura-se, por linha, a coordenada desta mais próxima de cada coordenada deste ônibus. Com isso, cada linha tem suas listas preenchidas com verdadeiro ou falso se, naquela coordenada do ônibus, pelo menos uma coordenada esteve abaixo do mesmo valor de tolerância utilizado na etapa anterior.

[4]Como definido no padrão IEEE 754.

Conflitos podem ocorrer entre linhas se houver nessa etapa de construção das linhas detecções para mais de uma linha no mesmo índice de lista. Os conflitos são resolvidos fazendo a identificação de grupos de detecção. Um grupo é uma sequência contínua de detecções feitas para uma linha. Para resolver esses conflitos, são aplicadas três heurísticas nos grupos na seguinte ordem:

1. grupos menores completamente contidos em grupos maiores são eliminados;
2. grupos de tamanho pequeno são removidos;
3. grupos separados por poucas coordenadas são unidos;
4. todas as coordenadas nas quais o ônibus está parado são eliminadas.

Se após a aplicação das heurísticas ainda houver conflitos, a preferência de correção é da linha que obteve melhor resultado na fase de detecção. Assim, o algoritmo é terminado com um vetor que classifica por entrada qual linha o ônibus percorria naquele dia.

6. Criação de Trajetórias Sintéticas

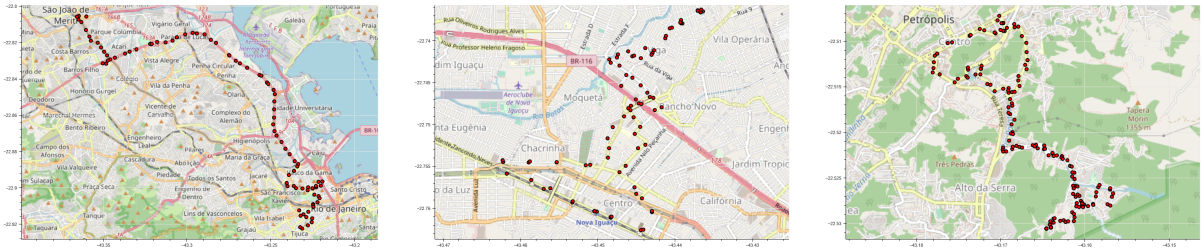
Para validar os resultados obtidos pelos algoritmos mencionados na Seção 3, faz-se o uso de dados sintéticos que simulam diferentes condições que costumam ser encontradas nos dados reais dos ônibus. Essas trajetórias foram construídas pela necessidade de se validar o desempenho dos algoritmos e pela impossibilidade de se fazer isso com os dados reais em larga escala, já que não é possível confiar nos valores de linha atribuídos

Para se criar a trajetória artificial, utiliza-se um banco de dados com as trajetórias de todas as linhas que serão testadas. Escolhe-se uma linha como teste e amostra-se dessa linha pontos separados a uma distância representada pela velocidade média e tempo de amostragem desejados. Definem-se então intervalos de falhas de comunicação de acordo com os quais alguns pontos, ou uma sequência de pontos, serão removidos. Finalmente, a todos os pontos restantes acrescenta-se um ruído Gaussiano à informação de local. As etapas desse procedimento são parametrizáveis de forma a simular diferentes condições e qualidades de dados.

O procedimento acima, sem o acréscimo de falhas, compreende um tipo de trajetória chamada nesse trabalho de simples. Esta é a menor unidade de trajetória que pode ser identificada em uma classificação de linhas. Dados reais diferem desse caso por possuírem em uma mesma trajetória mais de uma volta na mesma linha, ou percorrer linhas diferentes ao longo do dia. Pode-se montar uma sequência de trajetórias simples junto com um intervalo de pontos "parados" entre elas para definir uma trajetória complexa. Finalmente, podemos adicionar as falhas nessa trajetória complexa e ter uma trajetória complexa com falhas (chamada somente de trajetória com falhas futuramente).

Um exemplo dessa formação pode ser visualizado na Figura 3. A Figura 3 apresenta linhas diferentes que representam trajetórias simples, complexas e com falhas. Observa-se no exemplo de trajetória simples mostrada na Figura 3a, que a trajetória que o ônibus percorre possui apenas um sentido e nela o ônibus realiza apenas uma volta. Já na Figura 3b, a linha possui trajetória de ida e volta e é percorrida duas vezes pelo ônibus. Na trajetória complexa com falhas, observa-se na Figura 3c pequenos desvios no padrão de pontos com alguns aglomerados e outros trechos sem pontos.

Para entender como aplicar as falhas de comunicação e escolher os parâmetros mais apropriados para teste, fez-se uma análise nos próprios dados da prefeitura armazenados no laboratório. Foram selecionados três dias para teste: 01/03/2019, 18/03/2019 e 10/04/2019. Esses dias são, respectivamente, uma sexta-feira, uma segunda-feira e uma quarta-feira. Desses dias observou-se a velocidade média dos ônibus, a quantidade de vezes que ocorreram falhas



(a) Trajetória simples. (b) Trajetória complexa. (c) Trajetória complexa com falhas.

Figura 3: Exemplos de trajetórias sintéticas.

de comunicação por ônibus e por quanto tempo essas falhas duraram. Esses resultados podem ser observados na Figura 4. Um exemplo de uso dos resultados dessa análise é a figura 4c que evidencia a velocidade média dos ônibus na cidade com média de $19,8$ Km/h e esse valor é usado na formação dos dados sintéticos.

Em função do período de amostragem dos dados da prefeitura, definimos a ocorrência de uma falha como a remoção de um ponto na trajetória sintética. Nos dados, interpretamos que um ponto foi "removido" se há uma ausência na comunicação do ônibus por mais de 120 segundos, o dobro do tempo de amostragem padrão para aquele conjunto de dados.

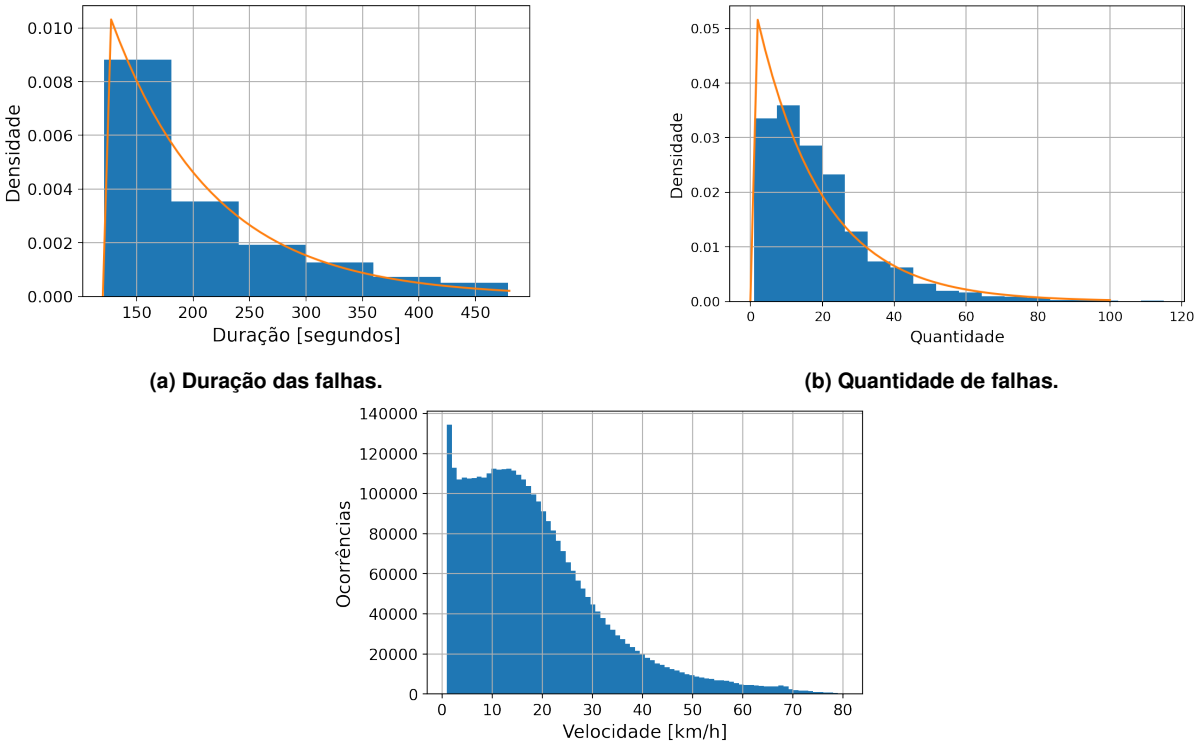
Observa-se na Figura 4a que o histograma respeita a separação dos intervalos e relaciona a duração dos intervalos em segundos diretamente com a quantidade de entradas faltantes no banco de dados que essa duração implicaria. Vemos que grande parte das falhas possui duração abaixo de 240 segundos, ou seja, é menor do que 3 entradas perdidas em sequência.

Ainda no contexto das falhas, é interessante observar a frequência de ocorrência dessas falhas na comunicação dos ônibus. Esse resultado é analisado na Figura 4b, que mostra a distribuição da quantidade de falhas. Essa distribuição mostra que, apesar da maioria das falhas resultar na perda de apenas uma ou duas entradas, essas falhas ocorrem 20 vezes em média para cada ônibus ao longo do dia, com desvio padrão de 15 falhas. Observa-se que as falhas são pequenas porém em grande quantidade e bem distribuídas.

7. Análise de Desempenho do UV-CSRC

Além dos testes executados em [Silva et al. 2020], três novos testes são executados ao longo desse trabalho: Dois testes que avaliam a eficácia de detecção do algoritmo e um para o tempo de processamento do mesmo. O primeiro consiste na validação dos resultados do UV-CSRC para dados de controle selecionados com o uso de mapas para visualização de dados, de modo a validar inicialmente a capacidade do algoritmo de reconhecer as linhas percorridas por alguns ônibus. A segunda etapa consiste na comparação dos resultados dos três algoritmos mencionados com os dados sintéticos, com um conjunto amplo de linhas em diferentes condições para avaliar como os algoritmos se comportam em cada cenário. Finalmente, um terceiro teste é realizado para avaliar o tempo de processamento de cada algoritmo para um mesmo conjunto de dados.

Nesse teste utiliza-se a ferramenta FAS-Bus [Silva et al. 2021], desenvolvida no GTA/UFRJ, para visualização do mapa para selecionar trajetórias de ônibus que percorriam pelo menos uma linha. Foram selecionados dados dos dias 06 de maio de 2019 16 de maio de 2019 (segunda e quinta-feira, respectivamente), tomando-se 24 ônibus que percorriam trajetórias das linhas 456, 457, 363 e 371. O objetivo consiste na validação da capacidade do algoritmo de classificar as linhas propriamente. Essa primeira análise mais simples não con-



(a) Duração das falhas.

(b) Quantidade de falhas.

(c) Velocidade média.

Figura 4: Histogramas da análise dos ônibus.

sidera o valor de classificação de cada entrada e sim os diferentes valores de classificação que foram obtidos ao longo de toda a trajetória.

O segundo teste consiste na criação de múltiplas trajetórias sintéticas como descritas na Seção 6 com variação dos parâmetros para a comparação dos resultados dos três algoritmos a serem avaliados. São realizados três conjuntos de testes: As primeiras trajetórias são formadas com apenas uma volta em um sentido de uma linha, as segundas são de múltiplas voltas e ambos os sentidos de uma linha e as terceiras tem a mesma formação das segundas só que com falhas adicionadas. Ao todo, são 672, 96 e 550 trajetórias testadas em cada bateria de testes, respectivamente. Denomina-se os três conjuntos de trajetórias de simples, composto sem falhas e composto com falhas daqui em diante. Para a análise do resultado, a lista de classificação de cada algoritmo é comparada com a lista de controle e desta comparação conta-se quantas entradas são diferentes e calcula-se a razão de entradas erradas.

O teste de desempenho mede o tempo em que os três algoritmos levavam para processar 100 trajetórias sintéticas que simulam um ônibus percorrendo apenas uma vez um único sentido da linha selecionada. Para esse teste é utilizada uma máquina contendo um processador Intel Core-I7 de 9ª geração, 16GB de RAM e GPU NVidia RTX 2060 com 6GB de memória de vídeo. Em função do limite de memória, o UV-CSRC faz a análise de 3 linhas e 3 ônibus por vez. Os algoritmos DTW e LCSS, executados em CPU, utilizam 5 *threads* para execução.

7.1. Resultados

Para o primeiro teste, os resultados são interpretados de dois modos: O primeiro reconhece a linha como detectada se pelo menos um de seus sentidos foi percorrido, o segundo considera que ambos os sentidos devem ser percorridos para detectar a linha. Tivemos que 15 dos 24 ônibus tiveram suas linhas detectadas com sucesso na primeira interpretação e 23 dos

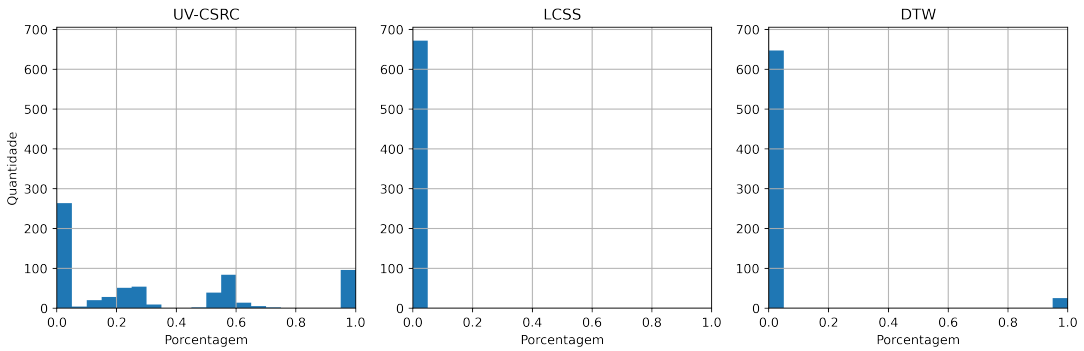


Figura 5: Resultados de trajetórias simples.

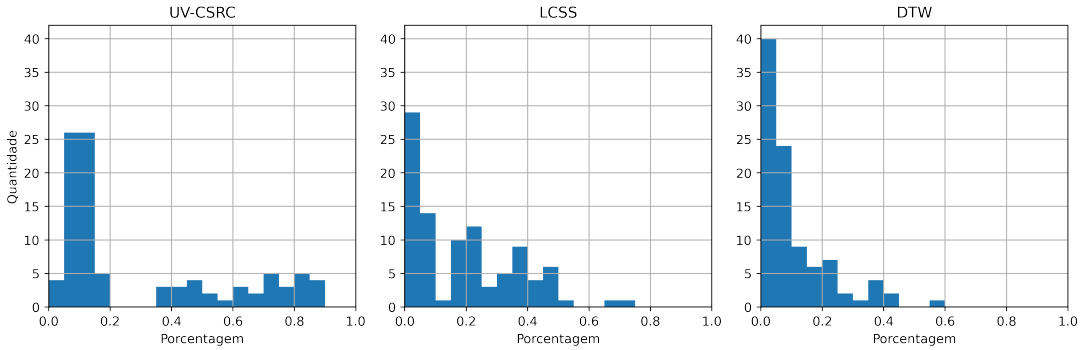


Figura 6: Resultados de trajetórias compostas sem falhas simuladas.

24 ônibus tiveram sucesso de detecção na segunda interpretação. Todas as falhas de detecção se concentraram entre a dificuldade de distinguir as linhas 456 e 457, que se diferem em apenas algumas ruas.

O segundo teste fornece histogramas para a razão de entradas inválidas nos três algoritmos. Os testes de trajetórias simples são exibidos na Figura 5, enquanto os testes de trajetórias compostas sem falhas na Figura 6 e com falhas na Figura 7. É realizada uma análise de cada conjunto em separado.

O primeiro conjunto, representado pela Figura 5, mostra um desempenho do UV-CSRC abaixo do esperado. Enquanto os resultados do UV-CSRC possuem diversas razões de porcentagem de detecção, os outros algoritmos apenas podem avaliar a trajetória como pertencente (resultado de 0% de erro no histograma) e não pertencente (resultado de 100% no histograma). Da distribuição dos resultados do UV-CSRC, que apresenta-se sde forma irregular, 31% das entradas possuem uma detecção perfeita (0% de erro), comparado à 100% do LCSS e 96.2% do DTW. Esse resultado mostra que o UV-CSRC não substitui os algoritmos da literatura para casos trivais.

O segundo conjunto, representado na Figura 6, mostra o algoritmo igualando-se em resultado com os outros dois algoritmos, que expressaram uma significativa queda de desempenho na classificação de trajetórias. Um dos motivos para essa queda se dá pelo fato de que a etapa prévia de divisão da trajetória em trechos para classificação aumenta a chance de pelo menos uma dessas divisões apresentar um resultado errado, e isso invalida o resultado do trecho como um todo. O UV-CSRC não considera a ordem dos pontos e por isso torna-se mais preciso a medida que mais pontos repetem a mesma trajetória.

O terceiro conjunto, representado na Figura 7 mostra o UV-CSRC que possui um me-

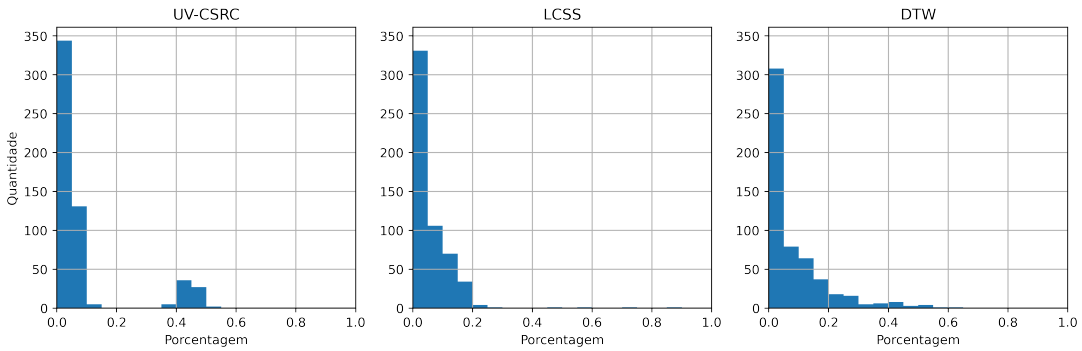


Figura 7: Resultados de trajetórias compostas com falhas simuladas.

lhor resultado em um ambiente com falhas introduzidas do que o DTW, e rendimento mais parecido com o LCSS, exceto por um viés. Observa-se que, com falhas de comunicação, introduz-se mais uma desvantagem com o uso do separador: as falhas podem prejudicar a separação das trajetórias se ocorrerem no instante que o ônibus está parado. Assim, não é possível discernir entre duas voltas distintas de uma linha e ambos os trechos recebem um resultado inválido. A razão do viés do UV-CSRC na região dos 45% requer investigação.

Finalmente, o terceiro teste mostra uma das grandes vantagens do algoritmo proposto: no teste de desempenho, o UV-CSRC mostra-se o mais rápido, finalizando em 1 minuto e 22 segundos com a janela utilizada. O DTW finaliza em 28 minutos e o LCSS finaliza em 45 minutos e 50 segundos. Isso mostra que, em cenários maior com tolerância a erros e janela de tempo mais estrita, o algoritmo proposto é uma solução para processamento e classificação de grande volumes de dados, com eficácia similar aos algoritmos da literatura porém com tempo de processamento bastante reduzido.

8. Conclusões

Neste trabalho foi apresentado o *Unordered and Vectorized Common Segment Ratio Classifier*, que propõe segmentar e classificar trechos de trajetória, para ser usado por exemplo na detecção de linhas de ônibus percorridas por um veículo da frota. Foram realizados dois testes com dados reais e sintéticos para avaliar a capacidade de detecção de linhas do UV-CSRC e comparar seu desempenho com o DTW e o LCSS, dois algoritmos encontrados na literatura.

Foi possível observar que o UV-CSRC, mesmo limitado em memória de GPU, é capaz de processar os dados de forma mais rápida do que os demais e alcançar resultados bem próximos a eles em cenários realistas. Diversos acontecimentos comuns a dados reais de trajetórias dos ônibus prejudicam a análise dos algoritmos comuns que necessitam de uma etapa intermediária de separação de trajetória, não existente no UV-CSRC .

Os próximos passos dessa pesquisa visam melhorias dos processos de criação de trajetórias sintéticas para mais testes dos algoritmos, uma observação mais aprofundada da variação dos parâmetros e o impacto que se tem nos resultados. Finalmente, planeja-se um estudo do UV-CSRC com o objetivo de aprimorar a sua acurácia, sem o comprometimento de sua eficiência.

Agradecimentos

Este trabalho foi realizado com recursos do CNPq, CAPES, FAPERJ e FAPESP (processo no. 2015/24494-8). Agradecemos a Matheus Felinto Tavares e Rodrigo de Souza Couto

pela cooperação no desenvolvimento da ferramenta FAS-Bus.

Referências

[Bejan et al. 2010] Bejan, A. I., Gibbens, R. J., Evans, D., Beresford, A. R., Bacon, J. e Friday, A. (2010). Statistical modelling and analysis of sparse bus probe data in urban areas. Em *13th International IEEE Conference on Intelligent Transportation Systems*, páginas 1256–1263. ISSN: 2153-0017.

[Cruz et al. 2020] Cruz, P., Couto, R. S., Costa, L. H. M., Fladenmuller, A. e de Amorim, M. D. (2020). A delay-aware coverage metric for bus-based sensor networks. *Computer Communications*, 156:192–200.

[Cruz Caminha et al. 2018] Cruz Caminha, P. H., Ferreira da Silva, F., Gonçalves Pacheco, R., de Souza Couto, R., Braconnot Velloso, P., Mitre Campista, M. E. e Maciel Kosmalski Costa, L. H. M. K. (2018). SensingBus: Using Bus Lines and Fog Computing for Smart Sensing the City. *IEEE Cloud Computing*, 5(5):58–69. Conference Name: IEEE Cloud Computing.

[Dias e Costa 2016] Dias, D. e Costa, L. (2016). Análise da Capacidade de Dados de uma Rede de Ônibus Urbanos. Em *Anais de XXXIV Simpósio Brasileiro de Telecomunicações*. Sociedade Brasileira de Telecomunicações.

[Farooq et al. 2010] Farooq, U., Haq, T. u., Amar, M., Asad, M. U. e Iqbal, A. (2010). GPS-GSM Integration for Enhancing Public Transportation Management Services. Em *2010 Second International Conference on Computer Engineering and Applications*, páginas 142–147, Bali Island, Indonesia. IEEE.

[Müller 2007] Müller, M. (2007). Dynamic Time Warping. Em *Information Retrieval for Music and Motion*, páginas 69–84. Springer, Berlin, Heidelberg.

[Neves et al. 2018] Neves, D. V., Dias, F. C. A. e Cordeiro, D. (2018). Uso de aprendizado supervisionado para análise de confiabilidade de dados de crowdsourcing sobre posicionamento de ônibus. Em *Anais do I Workshop Brasileiro de Cidades Inteligentes*. SBC.

[Silva et al. 2020] Silva, F., Cruz, P., Couto, R. e Costa, L. (2020). Redução de Inconsistências no Monitoramento da Frota de Ônibus Urbanos. Em *Anais de XXXVIII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*. Sociedade Brasileira de Telecomunicações.

[Silva et al. 2021] Silva, F., Tavares, M., Cruz, P., Couto, R. e Costa, L. H. (2021). FAS-Bus: Um Sistema de Análise da Frota de Ônibus Urbanos. Em *Anais Estendidos do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, páginas 1–8, Porto Alegre, RS, Brasil. SBC. ISSN: 2177-9384 event-place: Uberlândia.

[Su et al. 2020] Su, H., Liu, S., Zheng, B., Zhou, X. e Zheng, K. (2020). A survey of trajectory distance measures and performance evaluation. *The VLDB Journal*, 29(1):3–32.

[Vlachos et al. 2002] Vlachos, M., Kollios, G. e Gunopulos, D. (2002). Discovering similar multidimensional trajectories. Em *Proceedings 18th International Conference on Data Engineering*, páginas 673–684. ISSN: 1063-6382.

[Zheng 2015] Zheng, Y. (2015). Trajectory Data Mining: An Overview. *ACM Transactions on Intelligent Systems and Technology*, 6(3):1–41.