

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ
PRÓ-REITORIA DE PESQUISA, PÓS-GRADUAÇÃO E INOVAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Edgar dos Santos Oliveira

**Mineração de dados de trajetória: avanço do conhecimento e
contribuições para o desenvolvimento de bibliotecas e algoritmos**

Fortaleza – CE

21 de março de 2024

Dados Internacionais de Catalogação na Publicação
Instituto Federal do Ceará - IFCE
Sistema de Bibliotecas - SIBI
Ficha catalográfica elaborada pelo SIBI/IFCE, com os dados fornecidos pelo(a) autor(a)

D722m dos Santos Oliveira, Edgar.

Mineração de dados de trajetória : avanço do conhecimento e contribuições para o desenvolvimento de bibliotecas e algoritmos / Edgar dos Santos Oliveira. - 2024.
127 f. : il. color.

Dissertação (Mestrado) - Instituto Federal do Ceará, Mestrado em Ciência da Computação, Campus Fortaleza, 2024.

Orientação: Prof. Dr. Reinaldo Bezerra Braga.

Coorientação: Prof. Me. Clayson S. F. de S. Celes.

1. Trajectory data analysis.. 2. Python libraries.. 3. Stop point detection.. I. Título.

CDD 004

Edgar dos Santos Oliveira

**Mineração de dados de trajetória: avanço do
conhecimento e contribuições para o desenvolvimento de
bibliotecas e algoritmos**

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Linha de Pesquisa: Computação aplicada

Orientador: Dr. Reinaldo Bezerra Braga

Coorientador: MSc. Clayson S. F. de S. Celes

Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação (PRPI)

Programa de Pós-Graduação em Ciência da Computação (PPGCC)

Fortaleza – CE

21 de março de 2024



Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)
Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação (PRPI)
Programa de Pós-Graduação em Ciência da Computação (PPGCC)

Edgar dos Santos Oliveira

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, sendo aprovada pela Coordenação do Programa de Pós-Graduação em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará e pela banca examinadora:

Orientador: Dr. Reinaldo Bezerra Braga

Instituto Federal de Educação, Ciência e Tecnologia do
Ceará (IFCE)

Coorientador: MSc. Clayson S. F. de S. Celes

Universidade Federal de Minas Gerais (UFMG) &
University of Ottawa

Dra. Francisca Raquel de Vasconcelos Silveira

Instituto Federal de Educação, Ciência e Tecnologia do
Ceará (IFCE)

Dr. Tales Paiva Nogueira

Universidade da Integração Internacional da Lusofonia
Afro-Brasileira (UNILAB)

Fortaleza – CE
21 de março de 2024

*Este trabalho é dedicado a todos aqueles que me ensinaram a rir da vida,
principalmente das partes difíceis.*

Agradecimentos

Agradeço primeiramente à minha família, principalmente à minha mãe, meu pai e meu irmão. Eles me apoiaram não apenas neste mestrado, mas durante toda a minha vida. Afinal, sem eles eu não existiria e não seria quem sou.

Agradeço também aos meus professores e educadores durante a minha formação básica, alguns dos quais não recordo os nomes, mas que tiveram grande impacto na minha vida enquanto estudante.

Agradeço aos meus professores durante a minha graduação, que enriqueceram meus horizontes e me ajudaram a entender melhor que futuro desejo para mim. Entre eles, agradeço especialmente a Ricardo Lenz, meu orientador na disciplina de Trabalho de Conclusão de Curso, que teve grande impacto na minha visão do fazer científico e da filosofia da ciência.

Entre meus professores, não posso deixar de agradecer ao grande Mauro Oliveira. Professor responsável pelo que considero a melhor aula que já vi em minha vida, pois ao final desta, todo o curso de Bacharelado em Ciências da Computação fez sentido para mim. Igualmente, não posso deixar de mencionar seus ensinamentos profundos transmitidos através de "parábolas", que, sem sombra de dúvidas, não deixarão nossas mentes com facilidade, como ele mesmo disse: "...Eu mudei as areias do Egito."

Agradeço especialmente à minha professora Carina Teixeira, que me orientou na minha primeira bolsa e no meu primeiro contato com a escrita de artigos científicos. Agradeço especialmente por seus ensinamentos e conselhos, entre os quais um que carregarei pelo resto da minha vida: "o atributo mais importante para um estudante é a determinação".

Além disso, acredito que as orientações do meu coorientador Clayson Celes, vulgo "Seu Crayson", ajudaram a clarear e direcionar este mestrado de forma insubstituível. Agradeço a ele pelo tempo e paciência durante todo o período do mestrado, seus conselhos e meticulosidade ao revisar artigos, me inspiraram, me mostraram o quanto posso aprimorar minhas habilidades enquanto pesquisador.

Por fim, mas não menos importante, agradeço ao meu orientador Reinaldo Braga, sem ele este mestrado não seria possível. Sua imensa paciência e comprometimento enquanto orientador foram o que manteve esse mestrado vivo até o fim. Agradeço a ele pelos seus conselhos, dos quais sempre segui e nunca me arrependi, entre eles o conselho de iniciar este mestrado.

*“Não me siga porque eu também estou perdido, façam ou descubram o próprio caminho.
Façam como eu, inventem. Ou não façam como eu, inventem!”*
(Belchior)

Resumo

Esta dissertação explora o campo da mineração de dados de trajetória, com foco em bibliotecas e métodos disponíveis, suas capacidades e limitações. O objetivo central é contribuir para o avanço da área através da proposição de soluções e ferramentas mais completas. O primeiro capítulo apresenta uma análise comparativa de diversas bibliotecas Python para mineração de dados de trajetória. A pesquisa identifica que, embora existam várias opções, nenhuma oferece suporte completo a todas as tarefas de mineração. O capítulo destaca as vantagens e desvantagens de cada biblioteca, fornecendo um guia valioso para a seleção da ferramenta mais adequada para cada caso. O segundo capítulo concentra-se na comparação de métodos para detecção de pontos de parada em trajetórias. A pesquisa avalia a precisão, *recall* e F1 de cada método, destacando a variedade de opções existentes e suas características. Adicionalmente, o capítulo propõe um método para a geração de *ground-truth*, utilizando dados do OSM e o software SUMO. Essa contribuição resulta em um *dataset* público de grande relevância para pesquisas futuras. Com base nas conclusões dos capítulos anteriores, o terceiro capítulo apresenta o desenvolvimento de uma nova biblioteca para mineração de dados de trajetória. A biblioteca oferece suporte completo a todas as tarefas de mineração, incluindo *map-matching* aprimorado e funcionalidades não encontradas em outras soluções. Essa ferramenta tem o potencial de impulsionar o desenvolvimento de pesquisas na área. Os resultados da pesquisa apresentada nesta dissertação contribuem significativamente para o avanço da mineração de dados de trajetória. A análise comparativa de bibliotecas e métodos, a criação de um *dataset* público e o desenvolvimento de uma nova biblioteca oferecem recursos valiosos para a comunidade de pesquisa. As sugestões para pesquisas futuras abrem caminho para novas investigações que aprofundarão o conhecimento e as aplicações da mineração de dados de trajetória em diversos campos.

Palavras-chave: Análise de dados de trajetória. Bibliotecas Python. Detecção de pontos de parada.

Abstract

This dissertation explores the field of trajectory data mining, focusing on available libraries and methods, their capabilities, and limitations. The main objective is to contribute to the advancement of the area by proposing more complete solutions and tools. The first chapter presents a comparative analysis of several Python libraries for trajectory data mining. The research identifies that, although there are several options, but none of those offers complete support for all mining tasks. The chapter highlights the advantages and disadvantages of each library, providing a valuable guide for selecting the most suitable tool for each case. The second chapter focuses on comparing methods for detecting stop points in trajectories. The research evaluates the precision, recall, and F1 of each method, highlighting the variety of existing options and their characteristics. Additionally, the chapter proposes a method for generating ground-truth using OSM data and SUMO software. This contribution results in a public dataset of great relevance for future research. Based on the conclusions of the previous chapters, the third chapter presents the development of a new library for trajectory data mining. The library offers complete support for all mining tasks, including improved map-matching and functionalities not found in other solutions. This tool has the potential to boost research development in the area. The results of the research presented in this dissertation significantly contribute to the advancement of trajectory data mining. The comparative analysis of libraries and methods, the creation of a public dataset, and the development of a new library offer valuable resources for the research community. Suggestions for future research pave the way for new investigations that will deepen the knowledge and applications of trajectory data mining in various fields.

Keywords: Trajectory data analysis. Python libraries. Stop point detection.

Lista de ilustrações

Figura 1 – Paradigma da mineração de dados de trajetória.	30
Figura 2 – Ilustração de uma trajetória.	37
Figura 3 – Ilustração de um fluxo.	37
Figura 4 – Arcabouço para tratamento e análise de dados de mobilidade.	40
Figura 5 – filtragem de ruídos	67
Figura 6 – Detecção de pontos de parada	67
Figura 7 – Exemplo de compressão de trajetória	67
Figura 8 – Exemplo de segmentação de trajetória	67
Figura 9 – Número de pontos por dia.	69
Figura 10 – Número de viagens em cada dia da semana.	69
Figura 11 – Número de viagens em cada mês.	69
Figura 12 – Impacto da extração de características, na classificação de trajetórias.	70
Figura 13 – Número de pontos de densidade e fluxos.	70
Figura 14 – Predição de trajetória.	70
Figura 15 – Estrutura geral da metodologia implementada.	76
Figura 16 – Amostra dos dados gerados.	77
Figura 17 – Comparação de Métricas por Método.	81
Figura 18 – MoveMiner - Estrutura de Pastas.	85
Figura 19 – Trajetória Original	87
Figura 20 – Trajetória Segmentada	87
Figura 21 – Pymove - Trajetória Original	87
Figura 22 – Pymove - Trajetória Segmentadas	87
Figura 23 – MovingPandas - Trajetória Original	88
Figura 24 – MovingPandas - Trajetória Segmentadas	88
Figura 25 – Trajetória Original	88
Figura 26 – Trajetória Comprimida	88
Figura 27 – Pymove - Trajetória Original	89
Figura 28 – Pymove - Trajetória Comprimida	89
Figura 29 – MovingPandas - Trajetória Original	89
Figura 30 – MovingPandas - Trajetória Comprimida	89
Figura 31 – Scikit-Mobility - Trajetória Original	90
Figura 32 – Scikit-Mobility - Trajetória Comprimida	90
Figura 33 – Trajetória com <i>blur</i>	90
Figura 34 – Trajetória após <i>map-matching</i>	90
Figura 35 – Trajetória Original	91
Figura 36 – Trajetória após detecção de pontos de parada	91

Figura 37 – Pymove - Trajetória Original	92
Figura 38 – Pymove - Detecção de pontos de parada	92
Figura 39 – MovingPandas - Trajetória Original	92
Figura 40 – MovingPandas - Detecção de pontos de parada	92
Figura 41 – Scikit-Mobility - Trajetória Original	93
Figura 42 – Scikit-Mobility - Detecção de pontos de parada	93
Figura 43 – Trajetória Original	94
Figura 44 – Trajetória após filtragem espacial	94
Figura 45 – Pymove - Trajetória Original	94
Figura 46 – Pymove - Trajetória após filtragem espacial	94
Figura 47 – MovingPandas - Trajetória Original	95
Figura 48 – MovingPandas - Trajetória após filtragem espacial	95
Figura 49 – Trajetória Original	95
Figura 50 – Trajetória após filtragem temporal	95
Figura 51 – Pymove - Trajetória Original	96
Figura 52 – Pymove - Trajetória após filtragem temporal	96
Figura 53 – MovingPandas - Trajetória Original	96
Figura 54 – MovingPandas - Trajetória após filtragem temporal	96
Figura 55 – Trajetória Original	97
Figura 56 – Resultado de <i>query KNN</i>	97
Figura 57 – Pymove - Trajetórias Originais	98
Figura 58 – Pymove - Resultado de <i>query KNN</i>	98
Figura 59 – Grupo de Trajetórias	99
Figura 60 – truck0	99
Figura 61 – Trajetória Original	100
Figura 62 – Trajetória Suavizada	100
Figura 63 – MovingPandas - Trajetória original	101
Figura 64 – MovingPandas - Trajetória Suavizada	101
Figura 65 – <i>Check-ins</i>	102
Figura 66 –	102
Figura 67 –	102
Figura 68 – Trajetória Sintética	103
Figura 69 – Scikit-Mobility - Trajetória Sintética	104
Figura 70 – Trajetórias Originais	105
Figura 71 – Trajetória após DBSCAN	105
Figura 72 – Trajetórias Originais	105
Figura 73 – Trajetória após K-means	105
Figura 74 – Pymove - Trajetórias Originais	106
Figura 75 – Pymove - Trajetória após DBSCAN	106

Figura 76 – MovingPandas - Trajetórias Originais	107
Figura 77 – MovingPandas - Trajetória após Agregação	107
Figura 78 – Scikit-Mobility - Trajetórias Originais	107
Figura 79 – Scikit-Mobility - Trajetória após Agregação	107
Figura 80 – Trajetórias Originais	108
Figura 81 – Encontro de Trajetórias	108
Figura 82 – Conjunto de trajetórias após K-means	109
Figura 83 – Conjunto de trajetórias após K-means	110
Figura 84 – Padrões Periódicos, ao longo de 24 horas	110
Figura 85 – Conjunto de dados	111
Figura 86 – Divisão da trajetória	114
Figura 87 – Predição de velocidade	114
Figura 88 – Trajetórias atípicas	115
Figura 89 – Eventos Atípicos	116
Figura 90 – Pymove - Trajetória Original	116
Figura 91 – Pymove - Detecção de Eventos Atípicos	116

Lista de tabelas

Tabela 1 – Bibliotecas para tratamento e análise de dados de mobilidade.	31
Tabela 3 – Tipo de entidades móveis.	60
Tabela 5 – Técnicas de pré-processamento.	61
Tabela 6 – Recursos para gerenciamento de dados.	61
Tabela 7 – Métodos para lidar com incerteza.	62
Tabela 8 – Métodos para gerar Trajetórias.	62
Tabela 9 – Técnicas de minerações de padrões.	63
Tabela 10 – Técnicas para predição ou classificação de trajetórias	63
Tabela 11 – Técnicas para detecção de anomalias.	64
Tabela 12 – Tipos de visualização disponível.	64
Tabela 13 – Resultados	81
Tabela 14 – Número de bibliotecas com suporte para cada tarefa.	84
Tabela 15 – Número de tarefas suportadas por cada biblioteca.	84
Tabela 16 – Suporte a tarefas pela MoveMiner.	117

Lista de Algoritmos

1	Exemplo de Segmentação com MoveMiner	86
2	Exemplo de Segmentação com Pymove	87
3	Exemplo de Segmentação com MovingPandas	87
4	Exemplo de compressão de trajetória com MoveMiner	88
5	Exemplo de compressão com Pymove	89
6	Exemplo de compressão com MovingPandas	89
7	Exemplo de compressão com Scikit-Mobility	90
8	Exemplo de <i>map-matching</i> de trajetória com MoveMiner	90
9	Exemplo de Detecção de pontos de parada com MoveMiner	91
10	Exemplo de Detecção de pontos de parada com Pymove	91
11	Exemplo de Detecção de pontos de parada com MovingPandas	92
12	Exemplo de Detecção de pontos de parada com Scikit-Mobility	93
13	Exemplo de <i>spacial query</i> em trajetória com MoveMiner	93
14	Exemplo de <i>spacial query</i> em trajetória com Pymove	94
15	Exemplo de <i>spacial query</i> em trajetória com MovingPandas	94
16	Exemplo de <i>time query</i> em trajetória com MoveMiner	95
17	Exemplo de <i>time query</i> em trajetória com Pymove	96
18	Exemplo de <i>time query</i> em trajetória com MovingPandas	96
19	Exemplo de <i>KNN query</i> em trajetória com MoveMiner	97
20	Exemplo de <i>KNN query</i> em trajetória com Pymove	98
21	Exemplo de busca por similaridade em trajetória com MoveMiner	99
22	Exemplo de redução de incerteza em trajetória com MoveMiner	100
23	Exemplo de redução de incerteza em trajetória com MovingPandas	100
24	Exemplo de <i>LocationAttack</i> em trajetória com MoveMiner	101
25	Exemplo de <i>HomeWorkAttack</i> em trajetória com MoveMiner	101
26	Exemplo de geração de trajetória com MoveMiner	103
27	Exemplo de geração de trajetória com Scikit-Mobility	104
28	Exemplo de agrupamento de trajetória com MoveMiner (DBSCAN)	104
29	Exemplo de agrupamento de trajetória com MoveMiner (K-means)	105
30	Exemplo de agrupamento de trajetória com Pymove (K-means)	106
31	Exemplo de agrupamento de trajetória com MovingPandas	106
32	Exemplo de agrupamento de trajetória com Scikit-Mobility	107
33	Exemplo de mineração de padrões de trajetória com MoveMiner (Encounter)	108
34	Exemplo de mineração de padrões de trajetória com MoveMiner (PrefixSpan)	109
35	Exemplo de mineração de padrões periódicos de trajetória com MoveMiner	110
36	Exemplo de classificação de trajetória com MoveMiner	112

37 Exemplo de predição de trajetória com MoveMiner 113

38 Exemplo de detecção de trajetórias atípicas com MoveMiner 115

39 Exemplo de detecção de eventos atípicos com MoveMiner 115

40 Exemplo de detecção de eventos atípicos com Pymove 116

Sumário

1	INTRODUÇÃO	29
1.1	Motivação para a Dissertação	30
1.2	Descrição do problema	31
1.3	Objetivos	32
1.4	Produção científica	33
1.5	Estrutura da Dissertação	33
2	FUNDAMENTAÇÃO TEÓRICA	35
2.1	Tipos de fontes geradoras de dados de mobilidade	35
2.2	Incerteza de GNSS	36
2.3	Definição de Trajetória e Fluxo	36
3	COMPARAÇÃO QUALITATIVA DAS BIBLIOTECAS	39
3.1	Etapas para tratamento e análise de dados de mobilidade	39
3.1.1	Fontes geradoras de trajetórias	39
3.1.2	Pré-processamento de dados	41
3.1.2.1	Filtragem de ruídos	41
3.1.2.2	Detecção de pontos de parada	41
3.1.2.3	Compressão de trajetórias	42
3.1.2.4	Segmentação de trajetórias	42
3.1.2.5	Map matching	43
3.1.3	Gerenciamento de Dados	43
3.1.3.1	Consultas em Intervalo	43
3.1.3.2	Consultas KNN	44
3.1.3.3	Distância ou Similaridade entre Trajetórias	44
3.1.4	Incerteza	45
3.1.4.1	Gerenciamento da incerteza em dados de trajetória.	45
3.1.4.2	Proteção da privacidade em dados de trajetória.	45
3.1.5	Mineração de padrões em trajetórias	46
3.1.5.1	Movimento conjunto de objetos.	46
3.1.5.2	Agrupamento de trajetórias	47
3.1.5.3	Mineração de Padrões Sequenciais de Trajetórias	47
3.1.5.4	Mineração de Padrões Periódicos de Trajetórias	48
3.1.6	Classificação de trajetórias	48
3.1.7	Detecção de anomalias em trajetórias	49
3.1.7.1	Detecção de eventos atípicos em trajetórias	49

3.1.7.2	Detecção de anomalias em trajetórias	50
3.1.8	Visualização de dados de trajetórias	50
3.2	Bibliotecas Python para Mineração de Dados de Trajetória	51
3.2.1	Trackintel (MARTIN et al., 2023)	51
3.2.2	Yupi (REYES et al., 2023)	51
3.2.3	Mobilipy (PLESKOWICZ, 2022)	52
3.2.4	MobVis (SILVA et al., 2022)	52
3.2.5	PTRAIL (HAIDRI et al., 2022)	53
3.2.6	Mobilkit (UBALDI et al., 2021)	53
3.2.7	Traja (SHENK et al., 2021)	54
3.2.8	MovingPandas (GRASER; DRAGASCHNIG, 2020)	54
3.2.9	Mobipy (MAIA; CAMPELO; GRANDE-PB-BRAZIL, 2020)	55
3.2.10	PyMove (BRÁZ, 2020)	56
3.2.11	Scikit-Mobility (PAPPALARDO et al., 2022)	56
3.2.12	Tracktable (RINTOUL, 2020)	57
3.2.13	SMAFramework (RODRIGUES; VILLAS, 2019)	58
3.2.14	MOCHA (SOUZA et al., 2018)	58
3.2.15	Teetool (EERLAND et al., 2017)	59
3.2.16	Comparação Qualitativa das Bibliotecas de Mobilidade	60
3.2.16.1	Quais bibliotecas escolher com base nos dados?	60
3.2.16.2	Bibliotecas para tarefas de pré-processamento	60
3.2.16.3	Bibliotecas para gerenciar mobilidade	61
3.2.16.4	Bibliotecas para lidar com incerteza e análise de privacidade	61
3.2.16.5	Bibliotecas para gerar trajetórias	62
3.2.16.6	Bibliotecas para mineração de padrões de trajetórias	62
3.2.16.7	Bibliotecas para tarefa de classificação ou predição de trajetória	63
3.2.16.8	Bibliotecas para detecção de anomalias	63
3.2.16.9	Bibliotecas para visualização de mobilidade	64
3.3	Abordagem Prática para Tratamento e Análise de Mobilidade	65
3.3.1	Conjunto de dados reais	65
3.3.2	Pré-processamento dos dados	66
3.3.3	Análise exploratória	68
3.3.4	Extração de características	68
3.3.5	Predição de trajetória	70
3.4	Resultados da Comparação	70
4	COMPARAÇÃO DE MÉTODOS PARA DETECÇÃO DE PONTOS DE PARADA	73
4.1	Trabalhos Relacionados	73
4.1.1	Infostop	73

4.1.2	Detecção de pontos de parada para mineração de similaridade entre usuários	74
4.1.3	Movement-Stop-Noise (NOGUEIRA et al., 2018)	74
4.1.4	<i>Stop Go Classifier</i> (SPANG et al., 2022a)	75
4.1.5	Outlier Detection and Cleaning in Trajectories: A Benchmark of Existing Tools	75
4.2	Metodologia	76
4.2.1	Aquisição dos dados ou conjunto de dados	76
4.2.2	Detecção de Parada	78
4.2.2.1	Movement-Stop-Noise	78
4.2.2.2	Hampel Filter	78
4.2.2.3	Baseado em distância, tempo, velocidade e aceleração	78
4.2.3	<i>Maximum Physically Consistent Trajectories</i>	79
4.2.4	<i>Deep Feed Forward</i>	79
4.2.5	Métricas de comparação	79
4.2.5.1	Métricas de avaliação	80
4.3	Resultados e Discussões	80
4.4	Resultados	81
5	MOVEMINER: UMA BIBLIOTECA PARA MINERAÇÃO DE TRA- JETÓRIA	83
5.1	Tarefas e Bibliotecas	83
5.2	MoveMiner	83
5.2.1	Representação de Trajetória	85
5.2.2	Pré-processamento	86
5.2.2.1	Filtragem de ruídos	86
5.2.2.2	Segmentação	86
5.2.2.2.1	MoveMiner	86
5.2.2.2.2	Equivalente com Pymove	87
5.2.2.2.3	Equivalente com MovingPandas	87
5.2.2.2.4	Equivalente com Scikit-Mobility	88
5.2.2.3	Compressão	88
5.2.2.3.1	MoveMiner	88
5.2.2.3.2	Equivalente com Pymove	89
5.2.2.3.3	Equivalente com MovingPandas	89
5.2.2.3.4	Equivalente com Scikit-Mobility	90
5.2.2.4	<i>Map-Matching</i>	90
5.2.2.4.1	MoveMiner	90
5.2.2.4.2	Equivalente com Pymove	90
5.2.2.4.3	Equivalente com MovingPandas	91
5.2.2.4.4	Equivalente com Scikit-Mobility	91
5.2.2.5	Detecção de pontos de parada	91

5.2.2.5.1	MoveMiner	91
5.2.2.5.2	Equivalente com Pymove	91
5.2.2.5.3	Equivalente com MovingPandas	92
5.2.2.5.4	Equivalente em Scikit-Mobility	92
5.2.3	Gerenciamento de Dados	93
5.2.3.1	Busca por critério espacial	93
5.2.3.1.1	MoviMiner	93
5.2.3.1.2	Equivalente em Pymove	94
5.2.3.1.3	Equivalente em MovingPandas	94
5.2.3.1.4	Equivalente em Scikit-Mobility	95
5.2.3.2	Busca por critério temporal	95
5.2.3.2.1	MoviMiner	95
5.2.3.2.2	Equivalente em Pymove	96
5.2.3.2.3	Equivalente em MovingPandas	96
5.2.3.2.4	Equivalente em Scikit-Mobility	97
5.2.3.3	Busca por critério KNN	97
5.2.3.3.1	MoveMiner	97
5.2.3.3.2	Equivalente em Pymove	97
5.2.3.3.3	Equivalente em MovingPandas	98
5.2.3.3.4	Equivalente em Scikit-Mobility	98
5.2.3.4	Busca por critério de Similaridade	99
5.2.3.4.1	MoveMiner	99
5.2.3.4.2	Equivalente em Pymove	99
5.2.3.4.3	Equivalente em MovingPandas	99
5.2.3.4.4	Equivalente em Scikit-Mobility	100
5.2.4	Incerteza	100
5.2.4.0.1	MoveMiner	100
5.2.4.0.2	Equivalente em Pymove	100
5.2.4.0.3	Equivalente em MovingPandas	100
5.2.4.0.4	Equivalente em Scikit-Mobility	101
5.2.5	Privacidade	101
5.2.5.0.1	MoveMiner	101
5.2.5.0.2	Equivalente em Pymove	102
5.2.5.0.3	Equivalente em MovingPandas	102
5.2.5.0.4	Equivalente em Scikit-Mobility	102
5.2.6	Gerar Mobilidade	103
5.2.6.0.1	MoveMiner	103
5.2.6.0.2	Equivalente em Pymove	103
5.2.6.0.3	Equivalente em MovingPandas	103

5.2.6.0.4	Equivalente em Scikit-Mobility	104
5.2.7	Mineração de Padrões	104
5.2.7.1	Agrupamento	104
5.2.7.1.1	MoveMiner	104
5.2.7.1.2	Equivalente em Pymove	106
5.2.7.1.3	Equivalente em MovingPandas	106
5.2.7.1.4	Equivalente em Scikit-Mobility	107
5.2.7.2	Moving Together	108
5.2.7.2.1	MoveMiner	108
5.2.7.2.2	Equivalente em Pymove	108
5.2.7.2.3	Equivalente em MovingPandas	108
5.2.7.2.4	Equivalente em Scikit-Mobility	108
5.2.7.3	Padrões Sequenciais	109
5.2.7.3.1	MoveMiner	109
5.2.7.3.2	Equivalente em Pymove	110
5.2.7.3.3	Equivalente em MovingPandas	110
5.2.7.3.4	Equivalente em Scikit-Mobility	110
5.2.7.4	Padrões Periódicos	110
5.2.7.4.1	MoveMiner	110
5.2.7.4.2	Equivalente em Pymove	111
5.2.7.4.3	Equivalente em MovingPandas	111
5.2.7.4.4	Equivalente em Scikit-Mobility	111
5.2.8	Modelagem	111
5.2.8.1	Classificação	111
5.2.8.1.1	MoveMiner	111
5.2.8.1.2	Equivalente em Pymove	111
5.2.8.1.3	Equivalente em MovingPandas	112
5.2.8.1.4	Equivalente em Scikit-Mobility	112
5.2.8.2	Predição	112
5.2.8.2.1	MoveMiner	112
5.2.8.2.2	Equivalente em Pymove	114
5.2.8.2.3	Equivalente em MovingPandas	114
5.2.8.2.4	Equivalente em Scikit-Mobility	114
5.2.9	Detecção de Anomalias	114
5.2.9.1	Detectando Trajetórias Atípicas	114
5.2.9.1.1	MoveMiner	114
5.2.9.1.2	Equivalente em Pymove	115
5.2.9.1.3	Equivalente em MovingPandas	115
5.2.9.1.4	Equivalente em Scikit-Mobility	115

5.2.9.2	Detecção de Eventos Atípicos	115
5.2.9.2.1	MoveMiner	115
5.2.9.2.2	Equivalente em Pymove	116
5.2.9.2.3	Equivalente em MovingPandas	116
5.2.9.2.4	Equivalente em Scikit-Mobility	117
5.2.10	Visualização	117
5.3	Resultados	117
6	CONCLUSÃO	119
6.1	Trabalhos futuros	119
	REFERÊNCIAS	121

1 Introdução

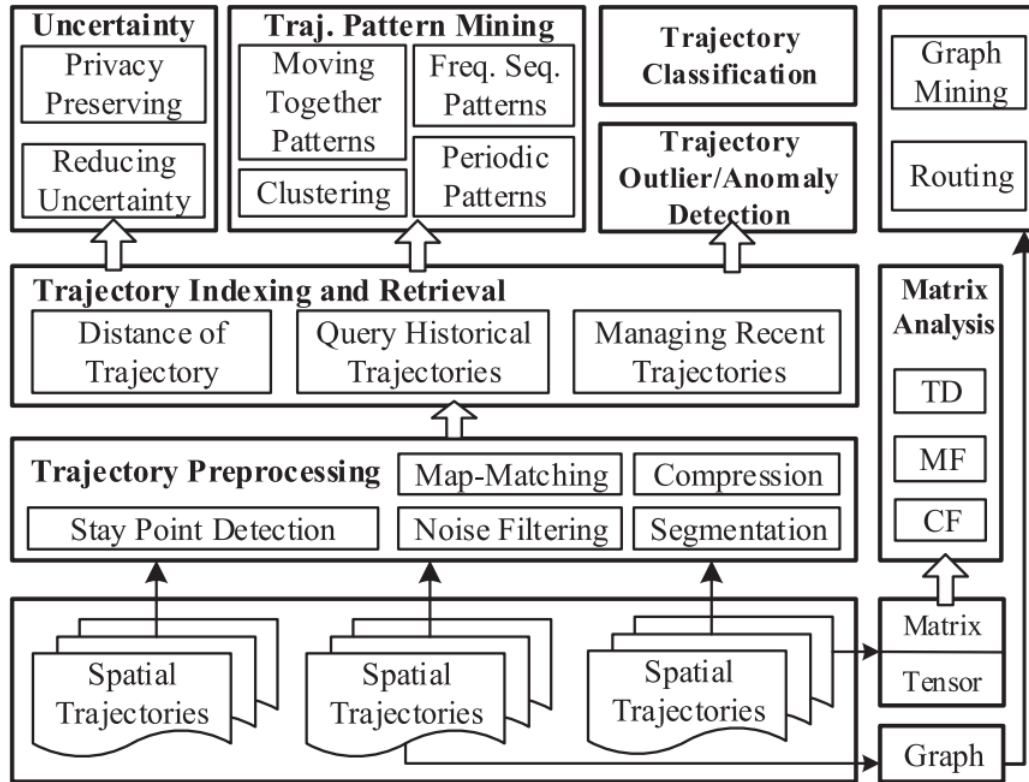
A área de Ciência de Dados tem ganhado significativa importância em diversos segmentos da sociedade, principalmente em virtude da crescente disponibilidade de dados, nos mais variados sistemas computacionais. Essa vasta quantidade de dados, quando analisada e relacionada, pode resultar em informações valiosas para diferentes domínios. Por exemplo, dados de trajetórias de pessoas e veículos podem oferecer uma oportunidade única para obter um entendimento da dinâmica da mobilidade urbana. Dessa forma, isso proporciona uma compreensão mais profunda dos padrões de deslocamento em ambientes geoespaciais (DENG et al., 2021; NAZIA et al., 2022).

Nesse contexto de mobilidade urbana, diversas iniciativas têm evidenciado como o conhecimento derivado da análise de dados pode ser altamente benéfico (DENG et al., 2021; YUAN; LI, 2021; MORENO; OLIVEIRA, 2022). No entanto, a extração desse conhecimento nem sempre se revela uma tarefa simples e, frequentemente, demanda um conhecimento multidisciplinar. Essa complexidade é ampliada quando se trata de dados de mobilidade (ZHENG, 2015), que, na maioria das vezes, são dados não estruturados, contendo ruídos, além de serem suscetíveis a imprecisões e lacunas.

Nesse sentido, a comunidade científica tem dedicado consideráveis esforços no desenvolvimento de algoritmos, ferramentas e bibliotecas destinadas a superar os desafios inerentes ao processo de mineração de dados.

1.1 Motivação para a Dissertação

Figura 1 – Paradigma da mineração de dados de trajetória.



Fonte: (ZHENG, 2015)

Neste contexto, um arcabouço teórico foi apresentado em (ZHENG, 2015), onde as tarefas inerentes da mineração de dados de trajetórias são divididas e exemplificadas, visível na Figura 1. Esse arcabouço tem se tornado cada vez mais estabelecido e utilizado em diversos trabalhos como base para análise da cobertura de ferramentas, assim como para revelar *insights* sobre as etapas essenciais para descoberta de conhecimento.

Desta forma, tendo como fundamento esse arcabouço teórico, é desejável que se investigue qual o atual estado da arte de bibliotecas Python (SAABITH; FAREEZ; VINOTHRAJ, 2019) utilizadas para mineração de dados de trajetória. Nesse sentido, essa investigação deve realizar uma comparação qualitativa que revele as funcionalidades de cada biblioteca disponível, identificando as implementações práticas discutidas no arcabouço teórico.

Essa investigação fornece base para novos trabalhos que analisem de forma mais aprofundada cada etapa do arcabouço teórico. Por exemplo, uma análise aprofundada de métodos para detecção de pontos de parada, ou seja, a detecção de momentos onde os objetos não estão se movendo. Desta maneira, demonstrando uma comparação entre

métodos descritos na literatura, mostrando o desempenho de cada método em métricas relevantes.

Além disso, uma análise aprofundada do ecossistema Python para mineração de dados de trajetória também possibilita a visualização de lacunas de métodos e soluções implementadas para solucionar a tarefa. Desse modo, diante de tal conhecimento, é desejável que se proponham novas ferramentas para suprir as soluções ausentes ou escassas nas bibliotecas.

1.2 Descrição do problema

O ecossistema Python oferece diversas bibliotecas para tratamento e análise de dados de mobilidade urbana, conforme detalhado na Tabela 1. Apesar de alguns estudos realizarem comparações entre algumas dessas bibliotecas disponíveis, a literatura carece de uma análise quantitativa abrangente do estado da arte das bibliotecas Python de código aberto para análise de mobilidade urbana. Uma outra deficiência observada na literatura é a falta de um estudo que apresente uma metodologia e a aplique em um cenário de uso prático, fazendo uso das ferramentas disponíveis no ecossistema de bibliotecas Python. Nesse sentido, o presente estudo abrange desde a teoria até a prática.

Tabela 1 – Bibliotecas para tratamento e análise de dados de mobilidade.

Trackintel (MARTIN et al., 2023)	Yupi (REYES et al., 2023)
Mobilipy (PLESKOWICZ, 2022)	Mobvis (SILVA et al., 2022)
PTRAIL (HAIDRI et al., 2022)	Mobilkit (UBALDI et al., 2021)
Traja (SHENK et al., 2021)	MovingPandas (GRASER; DRAGASCHNIG, 2020)
Mobipy (MAIA; CAMPELO; GRANDE-PB-BRAZIL, 2020)	PyMove (BRÁZ, 2020)
Scikit-Mobility (PAPPALARDO et al., 2022)	Tracktable (RINTOUL, 2020)
SMAFramework (RODRIGUES; VILLAS, 2019)	MOCHA (SOUZA et al., 2018)
Teetool (EERLAND et al., 2017)	

Além disso, a literatura apresenta inúmeros métodos propostos para detecção de pontos de parada (LI et al., 2008; NOGUEIRA et al., 2018; ASLAK; ALESSANDRETTI, 2020; CUSTERS et al., 2021; DUARTE; SAKR, 2023). No entanto, poucos trabalhos concentram-se na comparação abrangente dessas abordagens. Essa lacuna na pesquisa representa uma oportunidade para aprimorar a análise e a compreensão dos métodos, identificando suas fraquezas e pontos fortes, bem como suas características distintas que

podem favorecer um método em detrimento de outro, em diferentes cenários e tipos de dados.

Além das lacunas mencionadas anteriormente, é importante ressaltar que nenhuma das bibliotecas disponíveis disponibiliza todas as tarefas necessárias para mineração de trajetória. Isso pode dificultar o trabalho de pesquisadores de mobilidade e desenvolvedores de *software*, que precisam combinar bibliotecas diferentes para realizar suas análises.

Por exemplo, uma biblioteca é especializada no pré-processamento, enquanto outra pode ser especializada na visualização de trajetórias. Para realizar uma análise completa, o pesquisador ou desenvolvedor precisaria aprender a usar ambas as bibliotecas, o que pode ser uma tarefa complexa e demorada.

Essa situação também pode levar a problemas de integração e compatibilidade entre as bibliotecas. Isso pode dificultar o reuso do código e a reprodução dos resultados de pesquisa.

A título de exemplo, a forma como as bibliotecas representam e indexam os dados varia bastante, além de usarem dependências externas para realizar tarefas similares, gerando uma carga extra de memória e tempo de execução, para conversão de formato de trajetória.

Para resolver esse problema, é necessário o desenvolvimento de bibliotecas mais abrangentes, que disponibilizem um conjunto completo de tarefas para mineração de trajetória. Essas bibliotecas devem ser fáceis de usar e integrar com outras ferramentas.

A criação de bibliotecas desse tipo seria um importante avanço para a pesquisa e o desenvolvimento de software em mobilidade urbana.

1.3 Objetivos

Diante do contexto e problema apresentados, esta dissertação tem como objetivos:

- Desenvolver uma comparação qualitativa das bibliotecas Python de código aberto para análise de mobilidade urbana (OLIVEIRA et al., 2023a);
- Criar, validar e disponibilizar um *dataset* para *ground-truth* de pontos de parada de mobilidade (OLIVEIRA et al., 2023b)
- Comparar os métodos existentes para detecção de pontos de parada, considerando a acurácia, precisão, *recall* e F1 (OLIVEIRA et al., 2023b);
- Arquitetar, implementar e validar uma biblioteca Python que ofereça ferramentas para todas as etapas de mineração de trajetórias de acordo com o arcabouço teórico.

1.4 Produção científica

Durante este projeto de mestrado, os seguintes trabalhos científicos foram aceitos e publicados, a saber:

- **Oliveira, E., 2023. Tratamento e análise de dados de mobilidade urbana: Uma metodologia teórica e prática.** ERCEMAPI 2023 - Capítulo de livro de Minicursos;
- **Oliveira, E., 2023. Uma análise comparativa de técnicas de detecção de pontos de parada em ambientes urbanos.** ERCEMAPI 2023 - Artigos Completos.

1.5 Estrutura da Dissertação

Este trabalho está organizado da seguinte forma:

Capítulo 2: Conceitos e termos usados no restante da dissertação são estabelecidos, como trajetória, fluxo e tipos de fontes geradoras de dados.

Capítulo 3: O ecossistema de bibliotecas Python para mineração de dados de trajetória é explorado, com descrição de cada biblioteca, seus pontos fortes e limitações. Uma comparação é apresentada, verificando o suporte de métodos para cada tarefa ligada à análise de trajetórias. A metodologia usada na comparação é implementada e aplicada em dados reais.

Capítulo 4: Uma comparação abrangente de métodos para detecção de pontos de parada, uma das etapas da mineração de trajetórias, é apresentada. Um método para geração de *ground-truth* é usado para criar um *dataset*, posteriormente disponibilizado.

Capítulo 5: Com base nas descobertas dos capítulos anteriores, uma nova biblioteca, nomeada MoveMiner, é desenvolvida, com suporte para todas as tarefas estabelecidas anteriormente. A nova biblioteca é validada por meio de exemplos de código e resultados gerados para cada tarefa.

Capítulo 6: O presente capítulo apresenta e discute os resultados obtidos com os experimentos realizados ao longo da pesquisa. Além disso, são elencadas as considerações finais sobre o tema, incluindo perspectivas para trabalhos futuros.

2 Fundamentação teórica

2.1 Tipos de fontes geradoras de dados de mobilidade

De acordo com o trabalho de referência (CELES; BOUKERCHE; LOUREIRO, 2020), os principais tipos de fontes de dados de mobilidade de veículos podem ser classificados em seis categorias principais, conforme listado a seguir:

- ***Survey***: Estes dados são coletados por meio de questionários respondidos por indivíduos específicos, abrangendo informações como origem e destino da viagem, tempo de viagem, frequência e rotas. Geralmente, são realizados em intervalos de tempo espaçados.
- ***Schedule***: Estes dados são obtidos a partir de agendas pré-determinadas de veículos, geralmente abrangendo um único tipo de transporte, como ônibus.
- ***Road Sensors***: Estes dados são adquiridos por sensores instalados nas margens ou ao longo das vias. Podem fornecer informações como contagem de veículos, fluxo de tráfego e velocidade.
- ***Closed-Circuit Television***: Estes dados consistem em imagens capturadas por câmeras em veículos ou nas vias, que podem ser usadas para extrair informações por meio de técnicas de visão computacional. Assim como os *Road Sensors*, eles têm a desvantagem de exigir um alto custo de infraestrutura para instalação e o monitoramento é restrito a estradas e áreas específicas.
- ***Global Navigation Satellite System (GNSS)***: Dispositivos equipados com GNSS, como Sistema de Posicionamento Global (*Global Positioning System* - GPS), podem registrar a posição geográfica durante o movimento. Portanto, essa fonte de dados fornece informações de posicionamento refinadas e movimento quase em tempo real de um determinado veículo. Geralmente, dados de veículos particulares não são públicos, sendo mais comuns em táxis e ônibus.
- ***Sensing-as-a-Service (SaaS)***: Estas são plataformas que fornecem informações gerais sobre dados de tráfego, geralmente usando técnicas de fusão de dados para classificar a intensidade e fornecer informações de tráfego em ruas específicas, como apresentado no Google Maps ou Waze.

Além dos tipos de fontes geradoras, o trabalho citado também mostra que as fontes GNSS possuem mais disponibilidade que as demais. Por exemplo, dados de mobilidade de objetos representados por pontos de Global Positioning System (GPS).

Portanto, doravante, ao se referir a mobilidade ou trajetórias, tem-se como objetivo referir-se a dados gerados por GNSS. Desta forma, representado por um conjunto de pontos

com coordenadas geoespaciais, que será especificado a seguir.

2.2 Incerteza de GNSS

A incerteza, inerente a qualquer sistema de medição, incluindo o GNSS, refere-se à faixa de valores possíveis dentro da qual o valor "verdadeiro" provavelmente se encontra. Embora o GNSS seja conhecido por sua impressionante acurácia, diversos fatores contribuem para algum nível de incerteza em seus dados de posicionamento. Aqui estão algumas fontes principais de incerteza no GNSS:

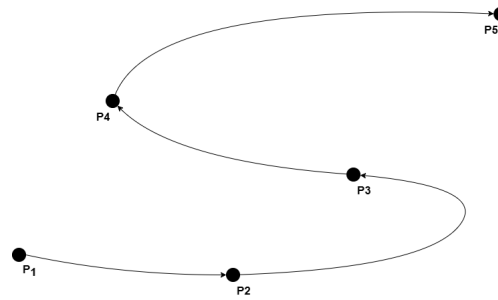
- Erros de relógio dos satélites: Mesmo os relógios atômicos altamente precisos a bordo dos satélites podem ter pequenas variações, impactando o tempo de viagem do sinal calculado e, conseqüentemente, a posição.
- Erros nas efemérides: As informações sobre a órbita do satélite (efemérides) transmitidas aos receptores podem conter pequenas imprecisões, levando a leves desvios na posição calculada.
- Atrasos ionosféricos e troposféricos: À medida que o sinal GNSS viaja pela atmosfera da Terra, ele pode ser atrasado pela ionosfera e troposfera, afetando a distância calculada a partir do satélite. Este efeito é mais significativo sob certas condições atmosféricas.
- Erros de multi-caminho: Os sinais GNSS podem refletir em edifícios, terrenos e outros objetos, chegando ao receptor por vários caminhos. Isso pode levar a confusão e erros no cálculo da posição.
- Ruído do receptor: O ruído eletrônico dentro do próprio receptor GNSS pode introduzir pequenos erros no sinal recebido, impactando a posição calculada.

2.3 Definição de Trajetória e Fluxo

Trajetórias podem ser representadas de várias maneiras, incluindo pontos geoespaciais, fluxos, imagens, vídeos e texto. Este trabalho lida exclusivamente com trajetórias de GPS e fluxos, conforme definido a seguir.

A representação de trajetória utilizada é a de uma sequência de pontos, onde cada elemento P_i pertencente a P consiste em uma tupla com latitude, longitude e *timestamp*. Desta forma, a trajetória é representada por uma sequência de pontos geoespaciais ordenados pelo tempo em que foram coletados, como ilustrado na Figura 2.

Figura 2 – Ilustração de uma trajetória.

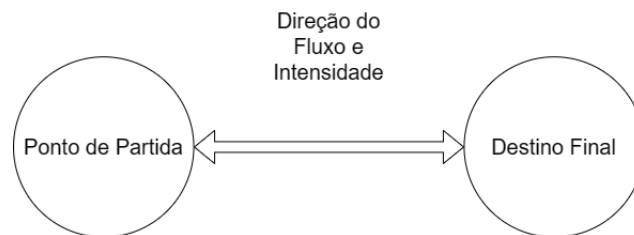


Fonte: Autores.

Outra forma de representar mobilidade muito comum são os fluxos, formados por duas coordenadas geoespaciais, um valor de intensidade e, opcionalmente, direção do fluxos.

Conforme ilustrado na Figura 3, um fluxo é representado por uma linha que conecta duas coordenadas geoespaciais, e o valor de intensidade indica a quantidade de movimento nessa linha.

Figura 3 – Ilustração de um fluxo.



Fonte: Autores.

Dados em formato de fluxos são comumente usados para representar o movimento de um número grande de objetos de um ponto a outro. Esse tipo de representação de mobilidade pode ser extraído de trajetórias, como será mostrado posteriormente.

Um exemplo de como transformar um conjunto de trajetórias em fluxo é o seguinte: Suponha que temos um conjunto de trajetórias de pessoas que se deslocaram de suas casas para o trabalho. Cada trajetória é representada por uma sequência de pontos geoespaciais, onde cada ponto representa a localização da pessoa em um determinado momento. Para transformar esse conjunto de trajetórias em fluxo, podemos calcular o fluxo entre cada par de coordenadas geoespaciais. O fluxo entre duas coordenadas é calculado como a quantidade de pessoas que passaram por essa linha em um determinado período de tempo.

Por exemplo, se temos uma trajetória que começa em (10, 10) e termina em (20, 20), podemos calcular o fluxo entre (10, 10) e (20, 20) como a quantidade de vezes que a pessoa que gerou essa trajetória passou pela linha que conecta essas duas coordenadas.

3 Comparação qualitativa das bibliotecas

No artigo (ZHENG, 2015), os autores descrevem e exemplificam um conjunto de etapas necessárias para a mineração de trajetórias, conforme apresentado na Figura 1. O paradigma proposto é bem consolidado, mas é desejável que novas tarefas sejam englobadas, como a predição de trajetórias, que extrai informações para prever o destino de um objeto.

Neste trabalho, é utilizado um arcabouço adaptado para análise de dados de mobilidade, baseado no paradigma proposto por (ZHENG, 2015), com alguns ajustes, como a inclusão do formato de dados de fluxos, como mostrado na Figura 4.

3.1 Etapas para tratamento e análise de dados de mobilidade

Com base na análise das subdivisões de processos para analisar trajetórias, são extraídas perguntas que mantenham as mesmas tarefas e facilitem a busca de recursos por parte do leitor. Tais tarefas, descritas em (OLIVEIRA et al., 2023a), são descritas a seguir. Em seguida, as perguntas serão apresentadas.

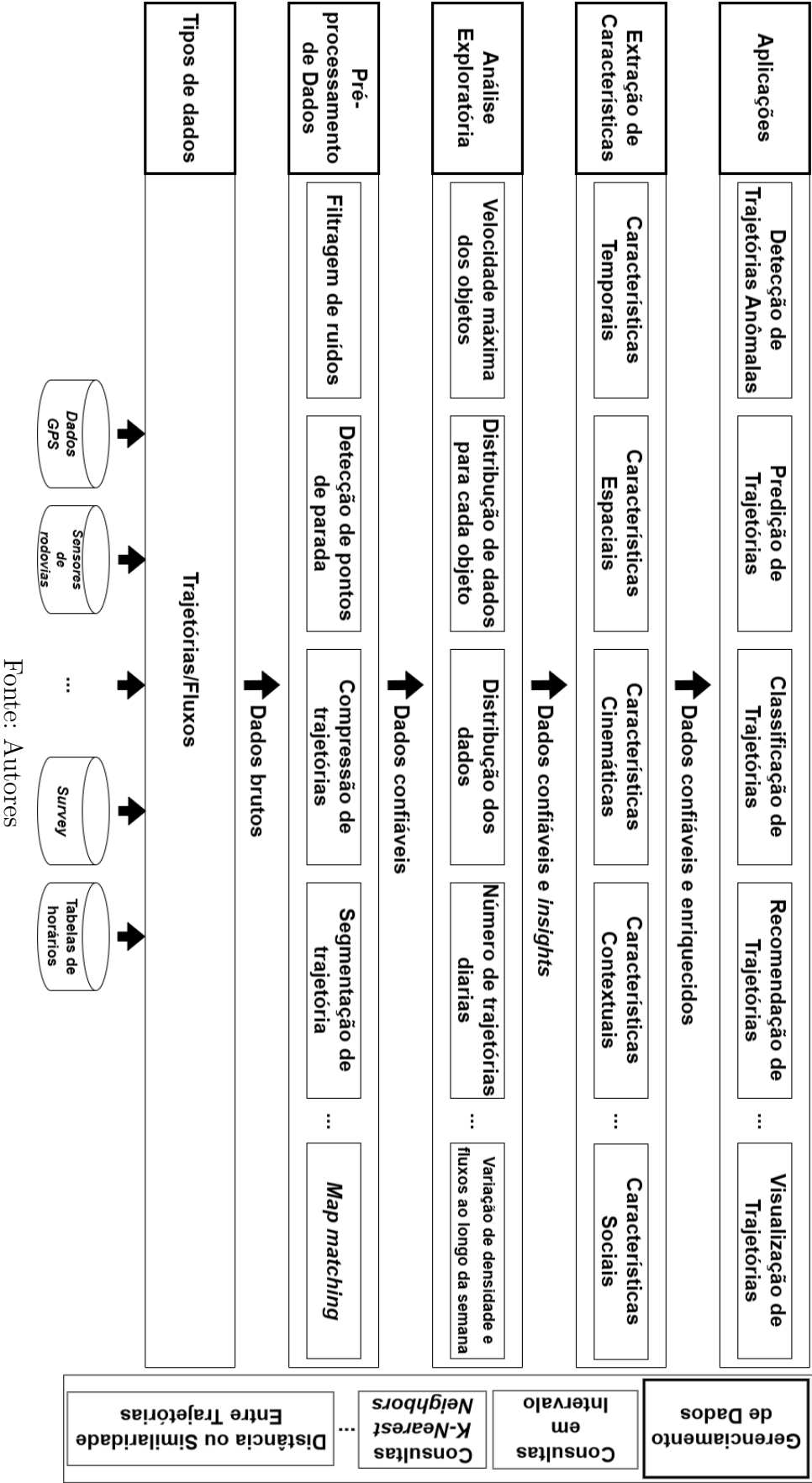
Com base no arcabouço apresentado na Figura 4, nesta seção, apresenta-se uma descrição de cada etapa, e suas sub-tarefas, necessárias para a mineração de dados de trajetórias. Além disso, para cada etapa, são extraídas Questões de Pesquisa (QP) pertinentes relacionadas às bibliotecas *open-source* do ecossistema Python para mineração de trajetórias. Essas perguntas indagam o suporte das bibliotecas às tarefas descritas por cada etapa.

3.1.1 Fontes geradoras de trajetórias

Durante esta fase da análise de trajetória, é necessária a classificação das fontes geradoras de dados em quatro grandes grupos: mobilidade de pessoas, mobilidade de veículos, mobilidade de animais e mobilidade de eventos naturais. Entretanto, atualmente, não existe nenhuma biblioteca disponível que ofereça funcionalidades para essa tarefa, o que torna impossível a extração de perguntas relacionadas. É importante ressaltar que a classificação das fontes geradoras de dados geralmente é realizada pela entidade responsável pela coleta dos dados e é disponibilizada como metadados.

Adicionalmente, é fundamental levar em consideração que a coleta de dados é uma tarefa complexa e a presença de incertezas é inevitável. Portanto, muitas bibliotecas apresentam soluções para mitigar esse problema. É importante, assim, considerar a capacidade da biblioteca de representar diferentes tipos de mobilidade e a maneira como os dados são gerenciados.

Figura 4 – Arcabouço para tratamento e análise de dados de mobilidade.



Diante desta etapa podemos extrair as seguintes Questões de Pesquisa (QP):

QP1) Quais tipo(s) de dados de mobilidade (por exemplo, ponto(s) GPS, segmento(s)) estão(ão) disponível(is) como representação(ões) para análise nas bibliotecas?

QP2) Explora qual(is) tipo(s) de representação de mobilidade?

3.1.2 Pré-processamento de dados

Este estágio da análise de trajetória envolve o tratamento dos dados coletados na etapa anterior. As tarefas aqui se dividem em cinco categorias, descritas a seguir:

3.1.2.1 Filtragem de ruídos

Os dados de trajetória frequentemente apresentam erros devido a diversos fatores, tais como a qualidade dos sensores ou falhas de comunicação. Esses erros podem prejudicar a análise dos dados, por isso é importante removê-los.

A filtragem de ruídos é uma tarefa que visa identificar e remover os erros dos dados de trajetória. Existem diversos algoritmos de filtragem de ruídos disponíveis, que podem ser divididos em três grandes categorias:

- Filtros de médias: Esses filtros calculam uma média dos valores vizinhos de um ponto de dados e o substituem por esse valor;
- Filtros Kalman: Esses filtros são baseados no modelo de Kalman, que é um modelo probabilístico que pode ser usado para estimar o estado de um sistema em tempo real;
- Detecção de *outliers* baseada em heurísticas: Esses métodos usam heurísticas para identificar pontos de dados que são considerados *outliers*.

Em alguns casos, os erros dos dados de trajetória são tão grandes que não podem ser corrigidos por algoritmos de filtragem.

3.1.2.2 Detecção de pontos de parada

Em uma trajetória, alguns pontos podem revelar informações importantes, como uma parada para abastecer o carro. Esses pontos são chamados de pontos de parada.

Em alguns casos, é necessário extrair esses pontos para melhorar a acurácia da análise. Por exemplo, se o objetivo é identificar o padrão de deslocamento de uma pessoa, é importante considerar os pontos de parada. Em outros casos, como para estimar o tempo de viagem, é preciso remover os pontos de parada e computar apenas os deslocamentos.

A detecção de pontos de parada é uma tarefa fundamental para a análise de mobilidade, um sub-campo da análise de dados que extrai conhecimento de informações de

deslocamento. Isso ocorre porque pontos de parada podem revelar informações importantes, como o início ou o fim de um evento, ou o local de uma atividade.

3.1.2.3 Compressão de trajetórias

Para tornar o envio, armazenamento e computação de trajetórias mais eficientes, é necessário reduzir o número de pontos sem perder muito de sua precisão. A literatura apresenta diversos algoritmos para realizar esta tarefa.

Os algoritmos de compressão de trajetórias podem ser divididos em quatro categorias principais:

- Distância métrica: Esses algoritmos usam a distância entre os pontos para identificar os pontos que podem ser removidos sem perder muita informação;
- Compressão offline: Esses algoritmos comprimem a trajetória inteira de uma vez, quando ela já foi totalmente coletada;
- Redução de dados online: Esses algoritmos comprimem a trajetória ponto a ponto, conforme ela é recebida;
- Compressão com significado semântico: Esses algoritmos comprimem a trajetória levando em consideração o significado dos pontos.

Uma forma de compressão com significado semântico é a redução de uma trajetória aos seus pontos de parada, tornando-a uma sequência de viagens diretas entre pontos de repouso, ignorando os detalhes entre esses pontos.

3.1.2.4 Segmentação de trajetórias

A segmentação de trajetórias é uma tarefa que divide uma trajetória em partes menores, chamadas de segmentos. Essa tarefa pode ser realizada por diversos motivos, como reduzir o poder computacional necessário para a análise, extrair padrões de sub-trajetórias ou enriquecer o conhecimento.

A segmentação de trajetórias pode ser feita de diversas formas, como:

- Com base em espaços temporais: Os segmentos são definidos com base em intervalos de tempo. Por exemplo, uma trajetória pode ser dividida em segmentos de 15 minutos;
- Com base em mudanças de ângulo: Os segmentos são definidos com base em mudanças na direção de movimento. Por exemplo, um segmento pode ser definido como um trecho de trajetória em que a direção de movimento não varia mais do que 10 graus;
- Outra forma de segmentar uma trajetória é por meio de pontos de parada, detectando-os e dividindo a trajetória com eles. Desta forma, é possível analisar apenas os segmentos de movimento entre eles.

3.1.2.5 Map matching

O *map matching* é uma tarefa que consiste em reposicionar pontos de uma trajetória em uma estrada ou rodovia. Essa tarefa é importante para corrigir erros causados por imprecisão na coleta dos dados.

Os algoritmos de *map matching* podem ser classificados em duas categorias principais:

- *Map matching offline*: Esses algoritmos usam um mapa pré-carregado para reposicionar os pontos da trajetória;
- *Map matching online*: Esses algoritmos reposicionam os pontos da trajetória conforme eles são recebidos.

Os algoritmos de *map matching* podem levar em conta diversas informações, como:

- Informações do sensor: Essas informações podem incluir a localização, a velocidade e a direção de movimento do objeto que gerou a trajetória;
- Informações do mapa: Essas informações podem incluir o traçado das estradas e rodovias;
- Informações adicionais: Essas informações podem incluir dados meteorológicos ou dados de tráfego.

Diante desta etapa podemos extrair a seguinte Questão de Pesquisa (QP):

QP3) Qual(is) técnica(s) de pré-processamento está(ão) implementada(s)?

3.1.3 Gerenciamento de Dados

O gerenciamento de dados é uma etapa importante no processo de análise de trajetórias, especialmente quando a quantidade de dados é grande. Isso ocorre porque o acesso a várias partes de trajetórias em um grande volume de dados pode ser oneroso. Portanto, é desejável que as ferramentas utilizadas para análise possuam um gerenciamento eficiente de dados e ofereçam técnicas de busca eficazes.

3.1.3.1 Consultas em Intervalo

Existem diversas abordagens para o gerenciamento eficiente de trajetórias. Uma delas é organizar as trajetórias em uma estrutura de árvore como *3D-Rtree*, que permite realizar buscas espaço-temporais representadas por uma caixa 3D. O resultado da busca são os nós da árvore que estiverem dentro da caixa.

Outra abordagem possível é dividir um período em vários intervalos de tempo, construindo um índice espacial individual como *R-tree* para as trajetórias geradas em cada intervalo. O resultado da busca são as trajetórias que estiverem dentro do intervalo de tempo especificado.

Por fim, é possível particionar um espaço geográfico em grades e construir um índice para as trajetórias que caem em cada grade. O resultado da busca são as trajetórias que estiverem dentro da grade especificada.

3.1.3.2 Consultas KNN

A abordagem de encontrar k trajetórias com a menor distância agregada em relação a alguns pontos podem ser útil em diversas situações, como na identificação de todos os carros que passaram por determinada estrada ou rodovia. Essa técnica é especialmente importante em análises de dados de mobilidade urbana, permitindo uma compreensão mais precisa do tráfego em uma determinada região e fornecendo informações valiosas para o planejamento urbano.

A abordagem K-Nearest Neighbors (KNN) consiste em encontrar as k trajetórias mais próximas de um determinado ponto ou conjunto de pontos. Essa abordagem pode ser útil em diversas situações, como:

- Identificação de trajetórias que passaram por uma determinada região: Essa aplicação pode ser usada, por exemplo, para identificar todos os carros que passaram por uma estrada ou rodovia específica;
- Identificação de trajetórias com comportamento similar: Essa aplicação pode ser usada, por exemplo, para identificar carros que estão viajando a uma velocidade semelhante ou que estão seguindo um caminho semelhante.

3.1.3.3 Distância ou Similaridade entre Trajetórias

A similaridade entre duas trajetórias é uma medida de quão parecidas elas são. Essa medida pode ser usada para uma variedade de aplicações, como identificação de grupos de trajetórias, análise de padrões de mobilidade e previsão de comportamento.

Uma forma comum de calcular a similaridade entre duas trajetórias é por meio da distância agregada entre os pontos das trajetórias. Essa distância pode ser calculada usando diferentes algoritmos, como:

- Distância de par mais próximo (Closest-Pair Distance): Essa distância é calculada encontrando o par de pontos mais próximos de cada trajetória;
- Distância da soma dos pares (Sum-of-Pairs Distance): Essa distância é calculada somando as distâncias entre todos os pares de pontos das trajetórias;
- Deformação dinâmica de tempo (Dynamic Time Wrapping): Essa distância é calculada ajustando as trajetórias no tempo para que coincidam com a duração uma da outra;
- Subsequência comum mais longa (Longest Common Subsequence - LCSS): Essa distância é calculada encontrando a subsequência de pontos mais longa que

aparece em ambas as trajetórias;

- Distância baseada em LCSS: Essa distância é calculada combinando a distância LCSS com outras medidas, como a distância da soma dos pares.

QP4) Qual(is) recurso(s) para gerenciamento de dados esta(ão) disponível(is)?

3.1.4 Incerteza

O movimento de um objeto no espaço é contínuo, enquanto seu registro é discreto. Portanto, as informações sobre uma trajetória sempre terão lacunas e, portanto, incertezas.

3.1.4.1 Gerenciamento da incerteza em dados de trajetória.

Existem duas principais abordagens para lidar com a incerteza em dados de trajetória. A primeira abordagem envolve a recuperação de trajetórias existentes a partir de uma base de dados, usando diferentes tipos de busca, como por meio de objetos geométricos ou funções de densidade e probabilidade independentes. A segunda abordagem envolve a inferência de caminhos a partir de dados incompletos, interpolando os pontos faltantes (CELES et al., 2017). Em alguns casos, é possível melhorar as inferências usando algoritmos adicionais, como o *map-matching*.

Para reduzir essa incerteza, existem diversas técnicas, como:

- Interpolation: Essa técnica consiste em interpolar os valores faltantes entre os pontos registrados;
- Smoothing: Essa técnica consiste em suavizar os dados registrados para reduzir a variação aleatória;
- Data fusion: Essa técnica combina dados de diferentes fontes para obter uma representação mais completa da trajetória.

3.1.4.2 Proteção da privacidade em dados de trajetória.

Serviços baseados em localização em tempo real podem expor a localização de um usuário, o que pode revelar informações sensíveis. Para proteger a privacidade, existem diversas abordagens, como o uso de técnicas de ocultação de localização, como *spatial cloaking* (CHOW; MOKBEL; AREF, 2009) e *mix-zones* (BERESFORD; STAJANO, 2003).

No caso de trajetórias históricas, a publicação de múltiplas trajetórias de um mesmo indivíduo pode permitir a inferência de informações sensíveis, como o seu endereço residencial ou local de trabalho. Algoritmos de privacidade, como *clustering-based* (ABUL; BONCHI; NANNI, 2008), *generalization-based* (NERGIZ; ATZORI; SAYGIN, 2008), *suppression-based* (TERROVITIS; MAMOULIS, 2008), *grid-based* (GIDOFALVI; HUANG; PEDERSEN, 2007), podem ser usados para proteger a privacidade em tais cenários.

Também existem métodos para aumentar a incerteza e proteger a privacidade do usuário, como:

- Noise addition: Essa técnica adiciona ruído aleatório aos dados registrados;
- Randomization: Essa técnica altera a ordem dos pontos registrados;
- Data obfuscation: Essa técnica obscurece os dados registrados para dificultar sua identificação.

Por fim, também é possível contornar esse problema através da geração de trajetórias sintéticas, porém dependendo de como elas são geradas é possível que as trajetórias geradas revelem padrões dos usuários

A partir desta etapa, podemos extrair as seguintes perguntas:

QP5) Qual(is) método(s) para lidar com a incerteza esta(ão) implementado(s)?

QP6) É possível gerar dados de mobilidade?

3.1.5 Mineração de padrões em trajetórias

A mineração de padrões em trajetórias pode ser dividida em quatro grandes categorias, que são descritas a seguir.

3.1.5.1 Movimento conjunto de objetos.

Para demonstrar que um grupo de objetos se movem juntos durante um intervalo de tempo, pode-se usar algoritmos como:

- Flock: Este algoritmo, baseado no comportamento de bandos de animais, como pássaros ou peixes, pode ser encontrado em (GUDMUNDSSON; KREVELD, 2006; GUDMUNDSSON; KREVELD; SPECKMANN, 2004);
- Convoy: Este algoritmo, projetado para rastrear um grupo de objetos que se movem em uma formação específica, pode ser encontrado em (JEUNG; SHEN; ZHOU, 2008; JEUNG et al., 2010);
- Swarm: Este algoritmo, baseado na ideia de que um grupo de objetos pode se comportar como um único organismo, pode ser encontrado em (LI et al., 2010);
- Travelling companion: Este algoritmo, projetado para rastrear um grupo de objetos que se movem juntos, mas não necessariamente em uma formação específica, pode ser encontrado em (TANGLU'AN et al., 2012; TANG et al., 2014);
- Gathering: Este algoritmo, projetado para reunir um grupo de objetos em um único local, pode ser encontrado em (ZHENG et al., 2013a; ZHENG et al., 2013b).

3.1.5.2 Agrupamento de trajetórias

O agrupamento de trajetórias é uma técnica que consiste em agrupar um conjunto de trajetórias em *clusters*, de forma que os pontos de cada *cluster* sejam semelhantes entre si. Essa técnica é útil para encontrar padrões comuns em um conjunto de trajetórias, como tendências de deslocamento ou padrões de uso de um espaço.

No cenário de redes rodoviárias, o agrupamento de trajetórias pode ser usado para encontrar:

- Caminhos representativos, ou seja, trajetórias que são seguidas por muitas pessoas;
- Tendências de deslocamento, como a direção mais comum de deslocamento em um determinado horário do dia;
- Padrões de uso de um espaço, como os locais mais populares para visitar em uma cidade;
- Existem vários estudos sobre o agrupamento de trajetórias em redes rodoviárias (KHARRAT et al., 2008).

Map-matching: Essa técnica consiste em mapear os pontos de uma trajetória para uma rede rodoviária. Isso permite que os algoritmos de agrupamento considerem a estrutura da rede rodoviária ao agrupar as trajetórias.

Algoritmos de agrupamento de grafos: Esses algoritmos são projetados para agrupar vértices em um grafo. Eles podem ser usados para agrupar os pontos de uma trajetória, que podem ser considerados como vértices de um grafo.

3.1.5.3 Mineração de Padrões Sequenciais de Trajetórias

Na análise de trajetórias, um padrão sequencial é identificado quando um determinado número de objetos em movimento percorrem uma sequência comum de locais em um intervalo de tempo semelhante. Para que uma sequência seja considerada um padrão, geralmente é necessário que ela seja encontrada mais vezes do que um valor de limiar previamente estabelecido. Alguns exemplos de algoritmos que podem ser usado para minerar padrões sequências, estão descritos a seguir.

O PrefixSpan é um algoritmo de mineração de padrões sequenciais que é baseado no conceito de prefixos. Um prefixo é uma sub-sequência de uma sequência mais longa. O PrefixSpan começa com um prefixo vazio e, em seguida, expande-o adicionando elementos um a um. Cada vez que um prefixo é expandido, o número de aparições nos dados é verificado; caso o número de vezes que parece seja superior a um *threshold*, ele é considerado um padrão.

O CloSpan é um algoritmo de mineração de padrões sequenciais que é baseado no conceito de fechamentos. Um fechamento é uma sub-sequência de uma sequência que não

pode ser estendida adicionando mais elementos. O CloSpan começa com um conjunto de padrões candidatos e, em seguida, usa um algoritmo de fechamento para encontrar novos padrões que são extensões dos padrões candidatos.

3.1.5.4 Mineração de Padrões Periódicos de Trajetórias

Dados de trajetória geralmente apresentam padrões periódicos, pois os locais que um objeto visita tendem a se repetir ao longo do tempo. A mineração de padrões periódicos em trajetórias pode ser útil para:

Previsão de movimentos futuros: Os padrões periódicos podem ser usados para prever onde um objeto estará em um determinado momento no futuro. Isso pode ser útil para aplicações como navegação ou vigilância. Compressão de dados: Os padrões periódicos podem ser usados para comprimir dados de trajetória, reduzindo o espaço necessário para armazená-los. Isso pode ser útil para aplicações como armazenamento de dados ou análise. Diversos algoritmos têm sido desenvolvidos para a detecção de padrões periódicos em trajetórias. Dois exemplos são:

Transformada de Fourier (BERESFORD; STAJANO, 2003): Esta técnica converte uma trajetória em uma série de componentes de frequência. Os componentes de frequência que são significativamente maiores que os outros indicam a presença de um padrão periódico. Algoritmos baseados em *clustering* (MORRIS; TRIVEDI, 2009): Esses algoritmos agrupam os pontos de uma trajetória em *clusters*. Os *clusters* que são significativamente maiores que os outros indicam a presença de um padrão periódico.

Diante desta etapa podemos extrair a seguinte Questão de Pesquisa (QP):

QP7) Qual(is) técnica(s) de mineração de padrões de trajetórias esta(ão) implementada(s)?

3.1.6 Classificação de trajetórias

A classificação de trajetórias é uma técnica que permite identificar diferenças entre trajetórias ou segmentos de trajetórias. Essa técnica pode ser usada para identificar diferentes tipos de movimentos, modos de transporte ou atividades humanas.

O processo de classificação de trajetórias geralmente é dividido em três etapas:

1. Segmentação: as trajetórias são divididas em segmentos, que são partes menores da trajetória que são mais facilmente classificáveis. A segmentação pode ser baseada em vários critérios, como a velocidade, a direção ou a mudança de direção da trajetória;
2. Extração de características: características são extraídas de cada segmento. Essas características podem descrever o movimento do segmento, como a distância percorrida, a aceleração ou a direção;

3. Classificação: um modelo de classificação é criado para classificar cada segmento. Várias técnicas de aprendizado de máquina podem ser usadas para criar modelos de classificação, como KNN, Support Vector Machine (SVM) e redes neurais.

Diante desta etapa podemos extrair a seguinte Questão de Pesquisa (QP):

QP8) Qual(is) técnica(s) para predição ou classificação de trajetórias esta(ão) implementada(s)?

3.1.7 Detecção de anomalias em trajetórias

A detecção de anomalias em trajetórias é uma técnica que permite identificar pontos ou segmentos de trajetórias que se destacam dos demais. Essa técnica pode ser usada em diversas aplicações, como monitoramento de veículos, detecção de comportamentos suspeitos e análise de mobilidade.

As anomalias podem ser identificadas de acordo com diferentes critérios, como:

Distância espacial percorrida: Um ponto ou segmento que se destaca dos demais em termos de distância espacial percorrida pode ser considerado uma anomalia. Por exemplo, um veículo que se desloca a uma velocidade muito alta ou muito baixa pode ser considerado uma anomalia.

Quebra de um padrão encontrado nas demais trajetórias: Um ponto ou segmento que foge de um padrão encontrado nas demais trajetórias pode ser considerado uma anomalia. Por exemplo, um veículo que faz uma curva muito acentuada ou que muda de direção repentinamente pode ser considerado uma anomalia.

A detecção de anomalias pode ser realizada tanto em tempo real quanto em trajetórias históricas. A detecção em tempo real permite identificar eventos incomuns assim que eles ocorrem, o que pode ser útil para a tomada de decisões em tempo real. A detecção em trajetórias históricas permite identificar tendências e padrões no longo prazo.

3.1.7.1 Detecção de eventos atípicos em trajetórias

A detecção de anomalias em trajetórias individuais pode ser usada para identificar comportamentos suspeitos, como pessoas que estão se movendo de forma anormal ou que estão entrando em áreas restritas. No entanto, essa abordagem pode não ser suficiente para detectar eventos anormais de grande escala, como acidentes ou protestos.

Uma abordagem alternativa para a detecção de eventos atípicos é usar um grande número de trajetórias. Essa abordagem pode ser usada para identificar eventos anormais no trânsito, por exemplo.

Uma maneira de identificar eventos anormais no trânsito é representar o evento como um subgrafo das redes de rodovias da cidade. Um evento anormal no trânsito pode ser identificado como um subgrafo onde os motoristas exibem comportamentos diferentes de seus padrões históricos.

Por exemplo, um evento anormal no trânsito pode ser representado como um subgrafo onde há um aumento repentino no número de veículos em uma determinada área. Uma vez que o evento anormal é identificado, é possível descrever o evento por meio da mineração de termos significativos postados em redes sociais no local da ocorrência.

Por exemplo, se um evento anormal no trânsito é identificado como um subgrafo onde há um aumento repentino no número de veículos em uma determinada área, a mineração de dados pode ser usada para identificar termos significativos que são mencionados em postagens nas redes sociais nessa área. Esses termos podem ajudar a descrever o evento anormal, como por exemplo, "acidente", "protesto" ou "manifestação".

3.1.7.2 Detecção de anomalias em trajetórias

Trajetórias atípicas, também conhecidas como anomalias em trajetórias, são aquelas que se diferenciam das demais em termos de forma, tempo de viagem ou outros critérios. Essas anomalias podem ser detectadas por meio de diferentes técnicas, como:

Clustering baseado em densidade: Essa técnica divide as trajetórias em grupos, chamados de *clusters*. Uma trajetória ou segmento é considerado uma anomalia se não se encaixa em nenhum *cluster*.

Mineração de padrões de frequência: Essa técnica identifica padrões de ocorrência frequentes nas trajetórias. Uma trajetória ou segmento é considerado uma anomalia se não é frequente o suficiente.

Diante desta etapa podemos extrair as seguintes Questões de Pesquisa (QP):

QP9) Quais técnicas para detecção de anomalias estão implementadas?

3.1.8 Visualização de dados de trajetórias

A visualização de dados de trajetórias é uma ferramenta importante para a análise e compreensão de padrões de movimento. No entanto, a exibição de dados brutos, como em uma tabela, pode não ser suficiente para fornecer uma visão abrangente dos dados, como oferecido pela biblioteca (REBACK et al., 2020).

Para melhorar a compreensão dos dados, é importante enriquecer as informações geoespaciais por meio de imagens, gráficos e mapas interativos. Esses recursos podem ajudar a destacar padrões, tendências e anomalias nos dados de trajetórias.

Diante desta etapa podemos extrair a seguinte Questão de Pesquisa (QP):

QP10) Existe suporte para visualização?

3.2 Bibliotecas Python para Mineração de Dados de Trajetória

Nesta seção, são apresentadas as bibliotecas listadas na Tabela 1. Além disto, métricas comparativas com base nas informações disponíveis no GitHub podem ser encontradas juntamente com o código-fonte do capítulo¹.

3.2.1 Trackintel (MARTIN et al., 2023)

O Trackintel é uma biblioteca *open-source* para análise de dados de movimento, com foco em *human mobility*.

Pontos fortes: tem como objetivo preencher a lacuna de ferramentas voltadas para a etapa de pré-processamento, fornecendo recursos para padronizar essas tarefas e aumentar a reprodutibilidade e comparabilidade de estudos científicos. Para a etapa de pré-processamento, a biblioteca oferece recursos para limpeza, transformação e enriquecimento de dados.

Limitações: Embora o Trackintel seja uma biblioteca abrangente para o pré-processamento de dados de movimento, ele apresenta algumas limitações em relação a outras bibliotecas. Uma limitação é que o Trackintel não é otimizado para grandes conjuntos de dados.

Outra limitação é que o Trackintel não oferece suporte a todas as etapas de mineração de dados de movimento.

3.2.2 Yupi (REYES et al., 2023)

O Yupi é uma biblioteca Python para mineração de trajetórias que busca adicionar novos recursos enquanto aproveita recursos já existentes.

Pontos fortes:

- Aproveitamento de recursos existentes: O Yupi faz o uso de outras bibliotecas para análise de trajetórias, o que aumenta sua flexibilidade e abrangência;
- Recursos próprios: O Yupi também oferece recursos próprios, como a extração de trajetórias de vídeos, que podem ser úteis em aplicações específicas;
- Personalização: A biblioteca é altamente configurável e extensível, permitindo que os usuários personalizem o processo de análise de acordo com suas necessidades.

Limitações:

- Desempenho: O Yupi não é otimizado para grandes conjuntos de dados;

¹ <https://github.com/edgarsoliveira1/MINICURSO-ERCEMAPI2023>

- Suporte a tarefas de mineração: O Yupi não oferece suporte a todas as tarefas de mineração de dados de mobilidade.

3.2.3 Mobilipy (PLESKOWICZ, 2022)

Mobilipy é um pacote de análise de mobilidade que permite identificar informações completas de viagens a partir de dados GPS brutos.

Pontos fortes:

- Permite a extração de informações como segmentação em viagens e atividades;
- Identificação de meios de transporte utilizados;
- Detecção da Localização de casa e trabalho;
- Trabalha com dados de GTFS;

Limitações:

- Suporte a tarefas de mineração: O Mobilipy não oferece suporte a todas as tarefas de mineração de dados de mobilidade, listas na subseção anterior.

3.2.4 MobVis (SILVA et al., 2022)

O MobVis é uma biblioteca Python para visualização de dados de mobilidade, que fornece recursos para pré-processamento, extração de métricas e visualização de dados de movimento.

Recursos:

- Pré-processamento: limpeza, transformação e enriquecimento de dados;
- Extração de métricas: cálculo de métricas de mobilidade, como distância percorrida, tempo de viagem e velocidade média;
- Visualização: criação de mapas e gráficos de dados de movimento.

Pontos fortes:

- Foco na visualização: O MobVis é uma biblioteca especializada em visualização de dados de mobilidade;
- Facilidade de uso: O MobVis é uma biblioteca relativamente simples de usar, mesmo para usuários com pouca experiência em mineração de dados de mobilidade.

Limitações:

- Documentação: O MobVis é uma biblioteca nova e não possui documentação oficial. No entanto, é possível usar *docstrings* para entender melhor a ferramenta;
- Suporte a tarefas de mineração: O MobVis não oferece suporte a todas as tarefas de mineração de dados de mobilidade, listadas na subseção anterior.

3.2.5 PTRAIL (HAIDRI et al., 2022)

O PTRAIL é uma biblioteca Python para pré-processamento de dados de mobilidade, que oferece recursos para filtragem de ruídos, remoção de outliers, interpolação e extração de features.

Recursos:

- Filtragem de ruídos: remoção de dados inválidos ou imprecisos;
- Remoção de *outliers*: identificação e remoção de dados discrepantes;
- Interpolação: preenchimento de lacunas nos dados;
- Extração de *features*: cálculo de métricas de mobilidade, como distância percorrida, tempo de viagem e velocidade média.

Pontos fortes:

- Foco no pré-processamento: O PTRAIL é uma biblioteca especializada em pré-processamento de dados de mobilidade;
- Eficiência: O PTRAIL executa tarefas de pré-processamento de forma paralela e vetorizada, o que aumenta a eficiência do processamento;
- Integração com outras bibliotecas: O PTRAIL apresenta uma integração bem definida com outras bibliotecas, como GeoPandas.

Limitações:

- Suporte a tarefas de mineração: O PTRAIL não oferece suporte a todas as tarefas de mineração de dados de mobilidade, listadas na subseção anterior.

3.2.6 Mobilkit (UBALDI et al., 2021)

O Mobilkit é uma biblioteca Python para análise de mobilidade, que possui um enfoque especial na análise de mobilidade pós-desastres naturais. A biblioteca é construída sobre o framework Dask, que tem como objetivo facilitar o processamento de grandes volumes de dados.

Recursos:

- Identificação de áreas de risco;
- Avaliação do impacto de desastres naturais no deslocamento de pessoas;
- Apoio à tomada de decisão em situações de emergência.

Pontos fortes:

- Foco em uma área específica: O Mobilkit é uma biblioteca especializada em uma área específica;
- Eficiência: O Mobilkit é construído sobre o framework Dask, que facilita o processamento de grandes volumes de dados.

Limitações:

- Suporte a tarefas de mineração: O Mobilkit não oferece suporte a todas as tarefas de mineração de dados de mobilidade, listadas na subseção anterior.

3.2.7 Traja (SHENK et al., 2021)

O Traja é uma biblioteca Python para análise de trajetórias de animais. A biblioteca oferece um conjunto de ferramentas para a extração, análise e visualização de trajetórias de animais a partir de dados de vídeo.

Recursos:

- Extração de trajetórias de vídeos: O Traja oferece um conjunto de algoritmos para extrair trajetórias de animais a partir de dados de vídeo;
- Análise de trajetórias: O Traja oferece um conjunto de ferramentas para analisar trajetórias de animais, incluindo identificação de padrões, análise de tendências e detecção de anomalias;
- Visualização de trajetórias: O Traja oferece um conjunto de ferramentas para visualizar trajetórias de animais.

Pontos fortes:

- Foco em uma área específica: O Traja é uma biblioteca especializada em uma área específica;
- Funcionalidade abrangente: O Traja oferece um conjunto abrangente de recursos para a análise de trajetórias de animais.

Limitações:

- Suporte a tarefas de mineração: O Traja não oferece suporte a todas as tarefas de mineração de dados de mobilidade, listadas na subseção anterior.

3.2.8 MovingPandas (GRASER; DRAGASCHNIG, 2020)

O MovingPandas é uma biblioteca Python para exploração e análise de dados de mobilidade. A biblioteca é uma extensão do Pandas, Geopandas (JORDAHL et al., 2019) e HoloViz (STEVENS; RUDIGER; BEDNAR, 2015), e fornece estruturas de dados e funções para:

Cálculo de velocidade e direção:

- Cálculo da velocidade instantânea e média;
- Cálculo da direção instantânea e média.

Divisão de trajetórias em sub-viagens:

- Divisão por distância;

- Divisão por tempo;
- Divisão por evento.

Agregação:

- Agregação por tempo;
- Agregação por espaço.

Generalização:

- Redução da resolução espacial;
- Redução da resolução temporal.

Visualização:

- Visualização estática;
- Visualização dinâmica.

Pontos fortes:

- Popularidade: O MovingPandas é uma biblioteca popular, com uma grande comunidade de usuários e desenvolvedores;
- Abordagem abrangente: O MovingPandas oferece uma ampla gama de recursos para a exploração e análise de dados de mobilidade;
- Integração com outras bibliotecas: O MovingPandas é integrado com outras bibliotecas populares, como o Pandas, Geopandas e HoloViz, o que disponibiliza diversos recursos.

Limitações:

- Suporte a tarefas de mineração: O MovingPandas não oferece suporte a todas as tarefas de mineração de dados de mobilidade.

3.2.9 Mobipy (MAIA; CAMPELO; GRANDE-PB-BRAZIL, 2020)

O Mobipy é uma biblioteca Python para cálculo de métricas e padrões de mobilidade de dados geo-temporais. A biblioteca é construída sobre o Pandas, uma biblioteca popular para manipulação de dados.

Recursos:

- Cálculo de métricas básicas de mobilidade: velocidade, direção, distância, tempo de viagem, etc.;
- Cálculo de métricas avançadas de mobilidade: densidade de tráfego, fluxo de pessoas, etc.;
- Detecção de padrões de mobilidade: agrupamento, classificação, etc.

Pontos fortes:

- Base sólida em Mineração de Dados: O Mobipy oferece recursos para o cálculo de métricas de mobilidade;
- Desempenho competitivo: O Mobipy é uma biblioteca eficiente, que pode competir com outras bibliotecas populares em termos de desempenho.

Limitações:

- Limitação de métricas: O Mobipy oferece uma gama limitada de métricas, que pode não ser suficiente para aplicações específicas;
- Suporte a tarefas de mineração: O Mobipy não oferece suporte a todas as tarefas de mineração de dados de mobilidade.

3.2.10 PyMove (BRÁZ, 2020)

O PyMove é uma biblioteca Python para o processamento e visualização de trajetórias. A biblioteca é construída com foco na flexibilidade e extensibilidade do código, o que permite que os usuários adicionem funcionalidades conforme necessário.

Recursos:

- Processamento de trajetórias: O PyMove oferece uma ampla gama de recursos para o processamento de trajetórias, incluindo: leitura e escrita de dados de trajetória; cálculo de métricas de mobilidade; visualização de trajetórias;
- Extensibilidade: O PyMove é uma biblioteca extensível, o que permite que os usuários adicionem funcionalidades conforme necessário.

Pontos fortes:

- Flexibilidade: O PyMove é uma biblioteca flexível, que pode ser adaptada às necessidades específicas dos usuários;

Limitações:

- Complexidade: O PyMove é uma biblioteca complexa, o que pode dificultar seu uso por usuários inexperientes, especialmente quando há necessidade de adição de novos recursos;
- Suporte a tarefas de mineração: O PyMove não oferece suporte a todas as tarefas de mineração de dados de mobilidade.

3.2.11 Scikit-Mobility (PAPPALARDO et al., 2022)

O Scikit-Mobility é uma biblioteca Python de análise de mobilidade baseada em estatística. A biblioteca é bastante popular na área de mineração de mobilidade, oferecendo uma ampla gama de funcionalidades, incluindo:

Recursos:

- Análise de trajetórias: O Scikit-Mobility oferece uma ampla gama de recursos para análise de trajetória;
- Análise de fluxos: O Scikit-Mobility oferece recursos para análise de fluxos, incluindo;
- Análise de privacidade: O Scikit-Mobility oferece recursos para análise de riscos à privacidade, incluindo estimação de risco de identificação de usuários.

Pontos fortes:

- Popularidade: O Scikit-Mobility é uma biblioteca popular, com uma grande comunidade de usuários e desenvolvedores;
- Funcionalidade: O Scikit-Mobility oferece uma ampla gama de funcionalidades, incluindo análise de trajetórias, análise de fluxos e análise de privacidade;
- Escalabilidade: O Scikit-Mobility é capaz de trabalhar com dados de mobilidade em grande escala.

Limitações:

- Complexidade: O Scikit-Mobility é uma biblioteca complexa, o que pode dificultar seu uso por usuários inexperientes;
- Suporte a tarefas de mineração: O Scikit-Mobility não oferece suporte a todas as tarefas de mineração de dados de mobilidade.

3.2.12 Tracktable (RINTOUL, 2020)

O Tracktable é uma biblioteca Python para análise de mobilidade baseada em C++. A biblioteca se destaca pela sua eficiência no processamento de grandes volumes de dados de trajetória, graças à implementação de suas principais estruturas de dados e algoritmos em C++.

Recursos:

- Processamento de trajetórias: O Tracktable oferece uma ampla gama de recursos para processamento de trajetórias, incluindo: leitura e escrita de dados de trajetória; cálculo de métricas de mobilidade; visualização de trajetórias;
- Análise de mobilidade: O Tracktable oferece recursos para análise de mobilidade, incluindo: identificação de padrões de mobilidade; detecção de anomalias; predição de mobilidade.

Pontos fortes:

- Eficiência: O Tracktable é uma biblioteca eficiente, capaz de processar grandes volumes de dados de trajetória;
- Funcionalidade: O Tracktable oferece uma ampla gama de funcionalidades, incluindo processamento de trajetórias, análise de mobilidade.

Limitações:

- Disponibilidade de Recursos: O Tracktable disponibiliza apenas uma pequena parte dos recursos disponíveis em C++ para Python;
- Suporte a tarefas de mineração: O tracktable não oferece suporte a todas as tarefas de mineração de dados de mobilidade.

3.2.13 SMAFramework (RODRIGUES; VILLAS, 2019)

O SMAFramework é uma biblioteca de análise de mobilidade urbana que foca na integração de múltiplos conjuntos de dados heterogêneos. A biblioteca oferece ferramentas para gerenciamento e análise de dados, mas não possui documentação, o que torna seu uso mais desafiador.

Recursos:

- Integração de dados: O SMAFramework oferece ferramentas para integração de múltiplos conjuntos de dados heterogêneos, incluindo dados de GPS, dados de transporte público e dados de sensoriamento urbano;
- Gerenciamento de dados: O SMAFramework oferece ferramentas para gerenciamento de dados, incluindo armazenamento, limpeza e preparação;
- Análise de dados: O SMAFramework oferece ferramentas para análise de dados, incluindo identificação de padrões, detecção de anomalias e predição.

Pontos fortes:

- Foco na integração de dados: O SMAFramework é uma biblioteca única que oferece um conjunto de ferramentas abrangente para integração de dados de mobilidade urbana.

Limitações:

- Falta de documentação: A falta de documentação torna o uso do SMAFramework mais desafiador em comparação com as demais bibliotecas;
- Suporte a tarefas de mineração: O SMAFramework não oferece suporte a todas as tarefas de mineração de dados de mobilidade.

3.2.14 MOCHA (SOUZA et al., 2018)

O MOCHA é uma ferramenta de pré-processamento de trajetórias que extrai métricas temporais, espaciais e sociais, além de classificar as trajetórias com base em suas distribuições de probabilidade, mostrando qual distribuição mais se assemelha a cada trajetória. A ferramenta também produz guias visuais para comparação de traços e modelos de mobilidade, o que é útil para avaliar a qualidade de trajetórias reais ou sintéticas.

Recursos:

- Extração de métricas: O MOCHA extrai uma ampla gama de métricas temporais, espaciais e sociais de trajetórias, incluindo: velocidade; direção; tempo de viagem; duração da estadia; tempo de parada; interação com outros usuários; atividade social;
- Visualização: O MOCHA produz guias visuais para comparação de traços e modelos de mobilidade.

Pontos fortes:

- Foco: O MOCHA oferece uma ampla gama de recursos para extração de métricas de trajetórias;
- Visualização: A geração de guias visuais é uma ferramenta útil para avaliar as trajetórias.

Limitações:

- Suporte a tarefas de mineração: O MOCHA não oferece suporte a todas as tarefas de mineração de dados de mobilidade.

3.2.15 Teetool (EERLAND et al., 2017)

O Teetool é uma biblioteca Python para análise de trajetórias em duas ou três dimensões. A biblioteca oferece ferramentas para mineração de dados e geração de predições, além de métricas específicas para análise de trajetórias de objetos aéreos.

Recursos:

- Métricas específicas: O Teetool oferece métricas específicas para análise de trajetórias de objetos aéreos, incluindo: * Região de confiança; * Altura média.

Pontos fortes:

- Métricas específicas: O Teetool oferece métricas específicas para análise de trajetórias de objetos aéreos.

Limitações:

- Puramente espacial: A biblioteca ignora fatores temporais;
- Obsolescência: A biblioteca não recebe atualizações desde 24 de maio de 2017;
- Falta de recursos para outras etapas da mineração: O Teetool não oferece recursos para todas as etapas da mineração de trajetórias, como limpeza e visualização.

Apesar de todas as bibliotecas apresentadas apresentarem pontos fortes e limitações, uma limitação é prevalente em todos os casos. A limitação compartilhada por todas as bibliotecas, é falta de recurso para uma cobertura total de todas as tarefas descrita no processo de mineração de dados de trajetórias.

3.2.16 Comparação Qualitativa das Bibliotecas de Mobilidade

Com base nas descrições apresentadas anteriormente, é notável a ausência de suporte para algumas tarefas. Portanto, buscando reduzir ainda mais a incerteza sobre o suporte de cada biblioteca para as tarefas de tratamento e análise de dados de mobilidade, esta seção realizará uma comparação qualitativa para verificar abrangência e cobertura da bibliotecas. Por meio dessa comparação, torna-se possível apresentar uma classificação para cada biblioteca, definindo qual desempenha melhor por cenário e quais tarefas não possuem suporte em nenhuma biblioteca.

3.2.16.1 Quais bibliotecas escolher com base nos dados?

Todos os tipos de representação de mobilidade abordados neste estudo, GNSS, são suportados por todas as ferramentas apresentadas. No entanto, ao considerar o tipo de entidade conforme a Tabela 3, é perceptível que, ao lidar com fluxos, a escolha se restringe a cinco bibliotecas. Nesse aspecto, é importante ressaltar que *MovingPandas*, *Scikit-Mobility* e *PyMove* oferecem um suporte mais abrangente para outras tarefas. Nesse sentido, com base na Tabela 3, as seguintes questões de pesquisas são respondidas: **QP1) Quais tipo(s) de dados de mobilidade (por exemplo, ponto(s) GPS, segmento(s)) estão(ão) disponível(is) como representação(ões) para análise nas bibliotecas?** e **QP2) Explora qual(is) tipo(s) de representação de mobilidade?**.

Tabela 3 – Tipo de entidades móveis.

Bibliotecas	<i>Trackintel</i>	<i>Yupi</i>	<i>Mobilipy</i>	<i>MobVis</i>	<i>PTRAIL</i>	<i>Mobilkit</i>	<i>Traja</i>	<i>MovingPandas</i>	<i>Mobipy</i>	<i>PyMove</i>	<i>Scikit-Mobility</i>	<i>Tracktable</i>	<i>SMAFramework</i>	<i>MOCHA</i>	<i>Teetool</i>
Trajetoória	+	+	+	+	+	+	+	+	+	+	+	+	-	+	+
Fluxo	-	-	+	-	-	-	-	+	-	+	+	-	+	-	-

3.2.16.2 Bibliotecas para tarefas de pré-processamento

Ao analisar os dados apresentados na Tabela 5, torna-se evidente a única biblioteca que abrange as outras cinco tarefas mencionadas é a *PyMove*. No entanto, considerando a possibilidade da utilização de várias bibliotecas, como *MovingPandas* e *Scikit-Mobility*, nas quais os dados são armazenados de maneira compatível, é factível que um desempenho semelhante seja alcançado. Portanto, é viável que a combinação dessas bibliotecas seja explorada para a obtenção de suporte quase completo para as atividades de pré-processamento.

Nesse sentido, com base na Tabela 5, a seguinte questão de pesquisa é respondida: **QP3) Qual(is) técnica(s) de pré-processamento esta(ão) implementada(s)?**.

Tabela 5 – Técnicas de pré-processamento.

Bibliotecas	Trackintel	Yupi	Mobilipy	MobVis	PTRAIL	Mobilkit	Traja	MovingPandas	Mobipy	PyMove	Scikit-Mobility	Tracktable	SMAFramework	MOCHA	Teetool
Filtragem de ruídos	-	+	-	-	+	-	-	+	-	+	+	+	-	-	-
Detecção de pontos de parada	+	-	+	+	-	+	-	+	-	+	+	-	-	-	-
Compressão de trajetórias	-	-	+	-	-	+	+	+	-	+	+	-	-	-	-
Segmentação de trajetória	-	-	+	-	+	-	+	+	-	+	-	-	-	-	-
Map-matching	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

A biblioteca PyMove, possui um *plugin* chamado *pymove-osmnx*. Este *plugin*, em tese, permitia a realização da tarefa de *map-matching*. No entanto, o *plugin pymove-osmnx* encontra-se descontinuado e não está mais funcionando.

3.2.16.3 Bibliotecas para gerenciar mobilidade

Uma conclusão que podemos derivar da Tabela 6 é que, ao lidar com um grande volume de dados, é viável empregar a ferramenta *PyMove* para a filtragem das tabelas específicas desejadas. Posteriormente, caso seja necessário, pode-se recorrer a outras bibliotecas para realizar tarefas adicionais. Essa abordagem permite uma manipulação de dados de mobilidade eficaz e modular. Nesse sentido, com base na Tabela 6, a seguinte questão de pesquisa é respondida: **QP4) Qual(is) recurso(s) para gerenciamento de dados esta(ão) disponível(is)?**.

Tabela 6 – Recursos para gerenciamento de dados.

Bibliotecas	Trackintel	Yupi	Mobilipy	MobVis	PTRAIL	Mobilkit	Traja	MovingPandas	Mobipy	PyMove	Scikit-Mobility	Tracktable	SMAFramework	MOCHA	Teetool
Consultas em Intervalo	-	-	-	-	+	+	+	+	-	+	-	+	-	-	-
Consultas KNN	-	-	+	-	-	-	-	-	-	+	-	-	-	-	-
Similaridade de Trajetórias	-	-	-	-	-	-	-	-	+	+	-	-	-	-	-

3.2.16.4 Bibliotecas para lidar com incerteza e análise de privacidade

O exame da Tabela 7 revela que análise de riscos para a privacidade é apoiada apenas pela biblioteca *Scikit-Mobility*, o que evidencia uma oportunidade para a implementação

de novas abordagens. Nesse sentido, com base na Tabela 7, a seguinte questão de pesquisa é respondida: **QP5) Qual(is) método(s) para lidar com a incerteza esta(ão) implementado(s)?**.

Tabela 7 – Métodos para lidar com incerteza.

Bibliotecas	Trackintel	Yupi	Mobilipy	MobVis	PTRAIL	Mobilkit	Traja	MovingPandas	Mobipy	PyMove	Scikit-Mobility	Tracktable	SMAFramework	MOCHA	Teetool
Reduzindo a Incerteza	-	-	-	-	+	-	+	+	-	-	-	-	-	-	-
Análise de Privacidade	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-

3.2.16.5 Bibliotecas para gerar trajetórias

Na tarefa de geração de mobilidade sintética, conforme demonstrado na Tabela 8, observa-se que nenhum dos recursos disponíveis nas bibliotecas se equipara em abrangência aos oferecidos pela *Scikit-Mobility*. No entanto, em outras bibliotecas, estão disponíveis recursos de geração de trajetórias por meio de métodos mais simples, como o *Random Walk*. A ausência de métodos mais avançados revela uma carência de abordagens robustas para a geração de mobilidade sintética. Este é um ponto que merece atenção e pode ser explorado como uma oportunidade para o desenvolvimento de novas técnicas e métodos nessa área. Nesse sentido, com base na Tabela 8, a seguinte questão de pesquisa é respondida: **QP6) É possível gerar dados de mobilidade?**.

Tabela 8 – Métodos para gerar Trajetórias.

Bibliotecas	Trackintel	Yupi	Mobilipy	MobVis	PTRAIL	Mobilkit	Traja	MovingPandas	Mobipy	PyMove	Scikit-Mobility	Tracktable	SMAFramework	MOCHA	Teetool
Gerar Trajetória	-	+	-	-	-	-	+	-	-	-	+	+	-	-	+

3.2.16.6 Bibliotecas para mineração de padrões de trajetórias

É possível que várias ferramentas sejam utilizadas em conjunto para obter uma gama mais completa de opções, como ilustrado na Tabela 9, combinando o uso do *Scikit-Mobility* com o *MovingPandas* ou *Traja*, que são compatíveis entre si. No entanto, não foi encontrada uma biblioteca, dentre as listadas, que disponha de recursos específicos para a mineração de padrões de movimentação conjunta (*Moving Together*), além da escassez de suporte para a mineração de padrões sequenciais e periódicos. Esses são aspectos que indicam possíveis

áreas de desenvolvimento e aprimoramento nas bibliotecas existentes. Nesse sentido, com base na Tabela 9, a seguinte questão de pesquisa é respondida: **QP7) Qual(is) técnica(s) de mineração de padrões de trajetórias esta(ão) implementada(s)?**.

Tabela 9 – Técnicas de minerações de padrões.

Bibliotecas	Trackintel	Yupi	Mobilipy	MobVis	PTRAIL	Mobilkit	Traja	MovingPandas	Mobipy	PyMove	Scikit-Mobility	Tracktable	SMAFramework	MOCHA	Teetool
<i>Moving Together</i>	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Agrupamento de Trajetórias	-	-	+	-	-	+	+	+	+	+	+	+	+	-	-
Padrões Sequenciais	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
Padrões Periódicos	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-

3.2.16.7 Bibliotecas para tarefa de classificação ou predição de trajetória

Conforme observado na Tabela 10, nenhuma das ferramentas listadas apresenta uma solução específica para a classificação. Isso indica que há uma lacuna nas bibliotecas existentes em relação a esse aspecto particular da mineração de trajetórias. Seria interessante explorar o desenvolvimento de métodos ou a integração de outras bibliotecas para abordar essa necessidade.

Não obstante, apenas a biblioteca *Traja* oferece uma solução para o problema de predição de trajetória. Isso destaca a importância de explorar e implementar abordagens específicas para essa tarefa, considerando sua relevância na análise de dados de mobilidade. Nesse sentido, com base na Tabela 10, a seguinte questão de pesquisa é respondida: **QP8) Qual(is) técnica(s) para predição ou classificação de trajetórias esta(ão) implementada(s)?**.

Tabela 10 – Técnicas para predição ou classificação de trajetórias

Bibliotecas	Trackintel	Yupi	Mobilipy	MobVis	PTRAIL	Mobilkit	Traja	MovingPandas	Mobipy	PyMove	Scikit-Mobility	Tracktable	SMAFramework	MOCHA	Teetool
Predição	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-
Classificação	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

3.2.16.8 Bibliotecas para detecção de anomalias

Conforme pode ser observado na Tabela 11, outra lacuna é evidenciada no estado da arte das bibliotecas Python para mineração de dados de mobilidade. Apenas a biblioteca

Tracktable oferece uma solução para a detecção de trajetórias anômalas. Isso realça a necessidade de ferramentas adicionais e o desenvolvimento de métodos específicos para a detecção de anomalias em dados de mobilidade, com o propósito de explorar todo o potencial dessas informações e identificar padrões incomuns ou comportamentos fora do esperado. Nesse sentido, com base na Tabela 11, a seguinte questão de pesquisa é respondida:

QP9) Quais técnicas para detecção de anomalias estão implementadas?.

Tabela 11 – Técnicas para detecção de anomalias.

Bibliotecas	Trackintel	Yupi	Mobilipy	MobVis	PTRAIL	Mobilkit	Traja	MovingPandas	Mobipy	PyMove	Scikit-Mobility	Tracktable	SMAFramework	MOCHA	Teetool
Detectando Trajetórias Atípicas	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-
Detecção de Eventos Atípicos	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-

3.2.16.9 Bibliotecas para visualização de mobilidade

Ao analisar os dados apresentados na Tabela 12, a ampla variedade de métodos disponíveis para a visualização de dados de mobilidade é evidente. É importante ressaltar que a biblioteca *MobVis* tem como foco exclusivo essa tarefa específica da mineração de dados de mobilidade. Portanto, a ausência de recursos para outras tarefas não diminui a efetividade de seu uso em conjunto com outras bibliotecas que abordam diferentes aspectos da análise de dados de mobilidade. A combinação dessas ferramentas pode oferecer uma abordagem completa e abrangente na visualização e exploração dos dados, proporcionando *insights* valiosos para a compreensão da mobilidade e a tomada de decisões informadas. Nesse sentido, com base na Tabela 12, a seguinte questão de pesquisa é respondida: **QP10) Existe suporte para visualização?.**

Tabela 12 – Tipos de visualização disponível.

Bibliotecas	Trackintel	Yupi	Mobilipy	MobVis	PTRAIL	Mobilkit	Traja	MovingPandas	Mobipy	PyMove	Scikit-Mobility	Tracktable	SMAFramework	MOCHA	Teetool
Imagens	+	+	-	+	+	+	+	+	-	+	+	+	-	+	+
Gráficos	+	+	-	+	+	+	+	+	-	+	+	+	-	+	+
Mapas Interativos	-	-	+	+	+	-	-	+	-	+	+	-	+	-	-

Ao final desta seção é possível entender melhor o suporte que cada biblioteca oferece às tarefas estabelecidas. Contudo, seria desejável fornecer um exemplo prático da aplicação

das bibliotecas em dados reais.

3.3 Abordagem Prática para Tratamento e Análise de Mobilidade

Na abordagem prática, é apresentado um conjunto de dados reais, demonstrando a aplicação dos recursos já presentes nas bibliotecas comparadas. Essa aplicação deve abranger a execução de todas as tarefas relacionadas à necessidade de uso de cada biblioteca.

A metodologia proposta será aplicada da seguinte forma:

1. **Conjunto de dados:** A apresentação do conjunto de dados que será analisado revela que o *dataset* utilizado foi coletado em uma cidade urbana e contém informações de trajetórias registradas por dispositivos GPS em veículos de transporte público, táxis.
2. **Pré-processamento dos dados:** A realização do pré-processamento dos dados de trajetória será conduzida utilizando as bibliotecas *SciKit-Mobility* e *MovingPandas*. Isso incluirá a limpeza dos dados, a filtragem de ruídos, a detecção de pontos de parada, a segmentação de trajetória e a compressão.
3. **Análise exploratória:** A biblioteca *MovingPandas* será empregada para a realização de uma análise exploratória dos dados. Isso envolverá a visualização das trajetórias, a identificação de padrões e a obtenção de *insights* iniciais sobre a mobilidade.
4. **Extração de características:** A biblioteca *PTRAIL* será utilizada para a extração de características relevantes dos dados de trajetória. Isso pode incluir informações como velocidade, distância percorrida, duração da viagem, entre outras.
5. **Predição de trajetória:** A modelagem dos dados de mobilidade será realizada por meio da biblioteca *Traja*. Isso pode envolver a aplicação de algoritmos de aprendizado de máquina ou técnicas de mineração de dados para a identificação de padrões e a predição dos dados de mobilidade.

Ao seguir essa metodologia, uma análise completa dos dados de mobilidade pode ser realizada, abrangendo desde o pré-processamento até a obtenção de conhecimentos e *insights* relevantes. Essa instância da metodologia é aplicada a um conjunto de dados real, possibilitando uma demonstração prática e aplicada dos conceitos discutidos nas seções anteriores.

3.3.1 Conjunto de dados reais

O uso do conjunto de dados *Taxi Service Trajectory*² é realizado para conduzir a análise. Este conjunto de dados descreve um ano completo das trajetórias de todos os

² <https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i>

442 táxis que operam na cidade do Porto, em Portugal. Esses táxis operam por meio de uma central de despacho de táxis, utilizando terminais de dados móveis instalados nos veículos. As viagens são categorizadas em três categorias: (A) baseada na central de táxis, (B) baseada em ponto de táxi ou (C) não baseada na central de táxis. Para a primeira categoria, foi fornecido um ID anonimizado quando essa informação estava disponível por meio da ligação telefônica. As duas últimas categorias referem-se a serviços que foram solicitados diretamente aos motoristas de táxi em um (B) ponto de táxi ou em uma (C) rua aleatória. Este conjunto de dados oferece uma oportunidade valiosa para aplicar a metodologia discutida e obter *insights* significativos sobre a mobilidade na cidade do Porto.

3.3.2 Pré-processamento dos dados

Limpeza dos dados

Antes de prosseguir para as etapas subsequentes, é fundamental realizar a limpeza e formatação dos dados. Para assegurar a integridade dos dados, foram excluídas todas as trajetórias em que a coluna 'MISSING_DATA' continha um valor diferente de 'False'. Além disso, as colunas que continham valores ausentes ou apresentavam apenas um valor em todo o conjunto de dados também foram removidas. Após essa filtragem, uma amostra aleatória das trajetórias remanescentes foi selecionada, correspondendo a aproximadamente 0,05% do total. Dessa amostra, apenas as trajetórias com mais de 5 pontos foram mantidas. Por fim, houve uma modificação na representação das trajetórias: anteriormente, cada linha do conjunto de dados representava uma trajetória completa, mas agora cada linha corresponde a um ponto específico, compartilhando os mesmos valores para o restante da trajetória.

Filtragem de ruídos

A seguir, é apresentado um exemplo específico de uma trajetória que ilustra claramente a presença de ruídos ou valores discrepantes, nos quais dois pontos são repentinamente distanciados consideravelmente de seus vizinhos. Para lidar com esses casos, é necessário que a filtragem dos dados seja realizada, e esse problema é abordado por várias bibliotecas disponíveis. Nesta instância, a solução fornecida pela biblioteca *Scikit-Mobility* foi utilizada. Por meio dessa biblioteca, todos os pontos que excedem a velocidade máxima de 300 km/h foram removidos. O resultado, conforme ilustrado na Figura 5, é apresentado na comparação entre a trajetória original, representada pela linha tracejada vermelha, e a nova trajetória representada pela linha azul, na qual os pontos removidos são destacados em verde.

Detecção de pontos de parada

Para ilustrar mais um exemplo, outra trajetória foi selecionada na qual um ponto de parada foi detectado, como pode ser visto na Figura 6. A detecção desse ponto de parada foi realizada por meio da funcionalidade fornecida pelo *Scikit-Mobility*. Nesse caso, os

Figura 5 – filtragem de ruídos

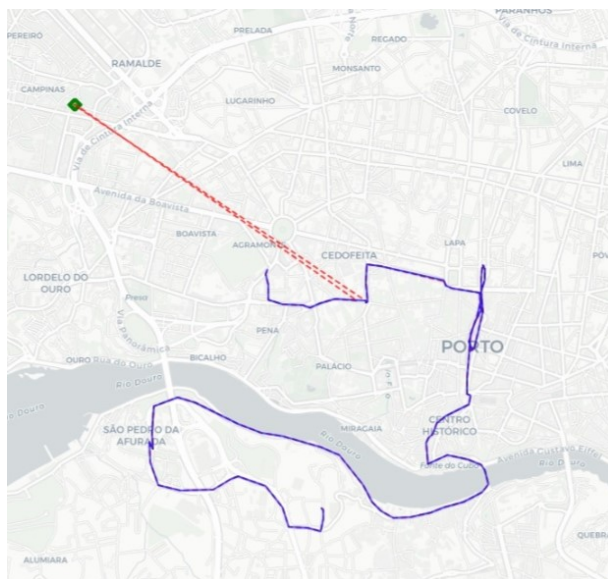
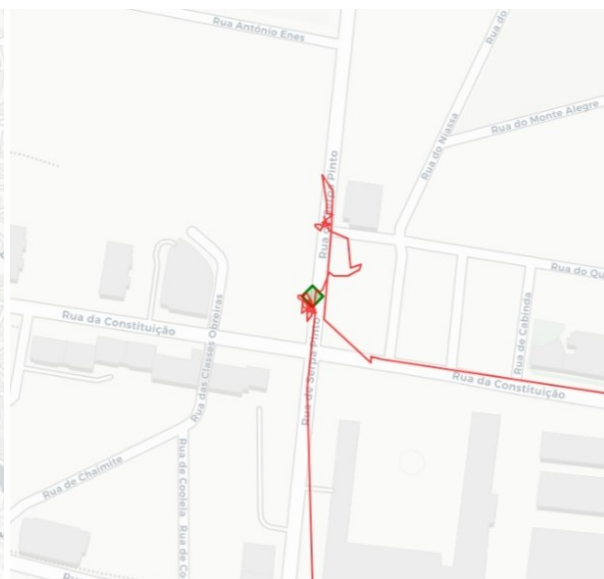


Figura 6 – Detecção de pontos de parada

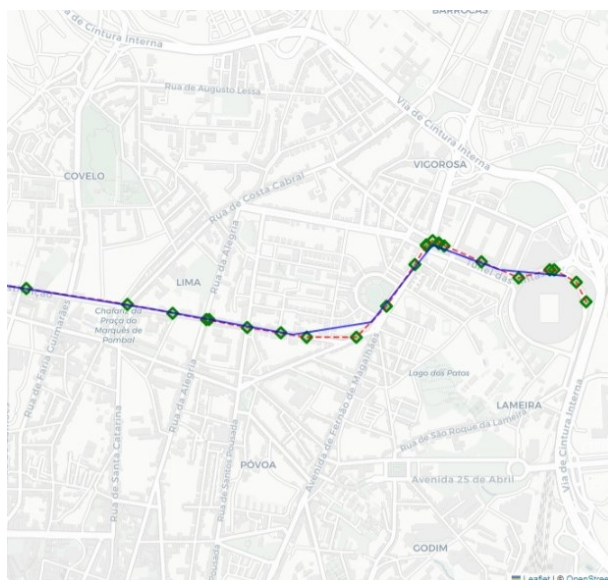


critérios utilizados para a classificação de um ponto como parada foram aqueles em que o objeto permaneceu por mais de 5 minutos a uma distância de 0.2 quilômetros. O ponto de parada detectado está destacado em verde na Figura 6.

Compressão de trajetória

Para exemplificar a compressão de uma trajetória, uma instância do *dataset* foi selecionada e o método de compressão fornecido pelo *Scikit-Mobility* foi aplicado. Esse método substitui conjuntos de pontos que estejam a uma distância de 0.2 quilômetros por um único ponto médio. A Figura 7 apresenta a trajetória original em azul, a nova trajetória comprimida em vermelho tracejado e destaca em verde os pontos que foram removidos durante o processo de compressão.

Figura 7 – Exemplo de compressão de trajetória



Segmentação de trajetória

Para realizar essa tarefa, os recursos da biblioteca *MovingPandas* são utilizados para segmentar uma trajetória, exemplificando critérios que podem ser empregados para essa divisão. Na Figura 8, a trajetória original é dividida em três novos grupos de sub-trajetórias. O primeiro grupo é criado com base em lacunas temporais, o segundo é formado pela detecção de pontos de parada, e o terceiro é resultado da análise das diferenças de velocidade.

3.3.3 Análise exploratória

Esta etapa busca aumentar o conhecimento sobre os dados disponíveis. Para exemplificá-la, serão respondidas algumas perguntas, listadas a seguir.

1. ***Qual foi a maior velocidade registrada?*** A maior velocidade registrada foi de 292.46 km/h, possivelmente originada de um *outlier* não filtrado.

2. ***Os pontos para cada 'taxi_id' estão igualmente distribuídos?***

Como é perceptível na Figura 13, onde cada barra representa o número de pontos para cada 'taxi_id', a quantidade varia consideravelmente.

3. ***Onde estão localizados os pontos mais significativos, os agrupamentos com maior densidade e qual é o fluxo entre eles?*** A Figura 13 torna visível que o centro da cidade tende a conter uma maior densidade de veículos.

4. ***Existe uma lacuna temporal nos dados de mobilidade?*** Os dados apresentam uma forma semelhante a uma função senoidal ao visualizar a quantidade de pontos para cada dia, começando em '2013-07-01 05:55:41' e terminando em '2014-06-30 09:22:04', conforme mostrado na Figura 9.

5. ***Qual a distribuição de viagem em cada mês e dia da semana?***

Nas Figuras 10 e 11, é perceptível que há uma maior quantidade de viagens nas segundas e sextas-feiras. De forma semelhante, é notável o aumento de viagens nos meses próximos a junho e outubro.

6. ***A densidade e os fluxos de mobilidade mudam ao longo dos dias da semana?*** Na Figura 9, a densidade e os fluxos entre os agrupamentos variam pouco ao longo dos dias da semana.

3.3.4 Extração de características

Nesta fase, métodos de extração de características presentes na biblioteca *PTRAIL* foram empregados para enriquecer o conjunto de dados, adicionando informações temporais,

Figura 9 – Número de pontos por dia.



Figura 10 – Número de viagens em cada dia da semana.

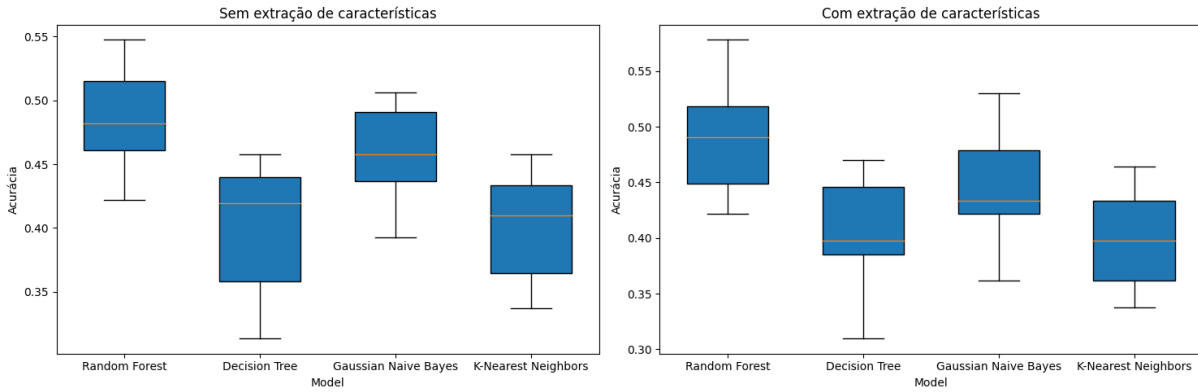
Figura 11 – Número de viagens em cada mês.

como dia da semana e hora do dia, além de informações derivadas dos movimentos, como distância, velocidade e aceleração.

Por fim, modelos disponíveis na biblioteca *scikit-learn*³ foram utilizados para exemplificar o impacto que a extração de características pode ter em tarefas relacionadas à análise de mobilidade, como a melhoria na acurácia da classificação dos tipos de chamadas (*call_type*). Essa abordagem demonstra como a utilização adequada das características extraídas pode contribuir significativamente para o aprimoramento dos resultados em aplicações de análise de mobilidade.

³ <https://scikit-learn.org/stable/index.html>

Figura 12 – Impacto da extração de características, na classificação de trajetórias.



3.3.5 Predição de trajetória

Com base na biblioteca Python *Traja*, realizou-se um experimento para prever os três próximos pontos de uma trajetória. Para isso, utilizamos o modelo *Long Short-Term Memory* (LSTM) disponibilizado pela biblioteca, o qual foi treinado e validado em subconjuntos distintos da amostra de dados. Após o processo de treinamento e validação, a predição com o menor Erro Quadrático Médio foi selecionada. Os resultados obtidos foram apresentados na Figura 14, demonstrando a acurácia da predição.

Figura 13 – Número de pontos de densidade e fluxos.

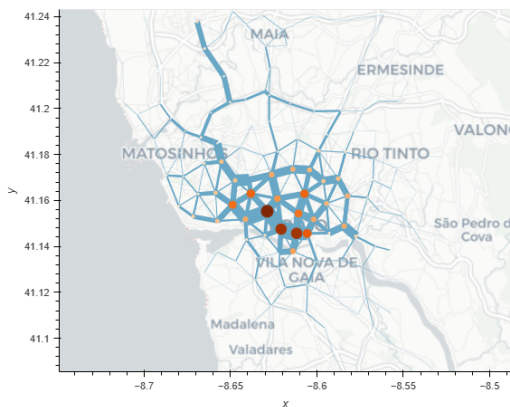
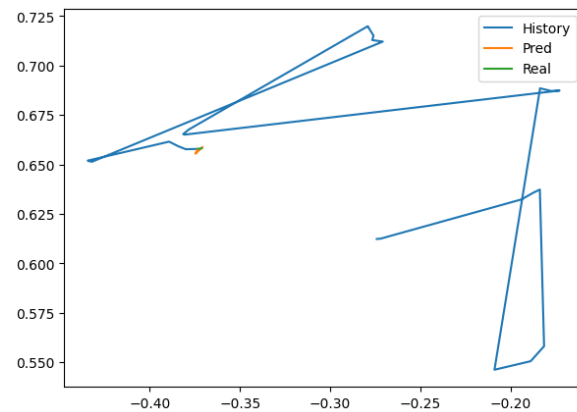


Figura 14 – Predição de trajetória.



3.4 Resultados da Comparação

O objetivo principal deste capítulo foi realizar uma comparação quantitativa entre bibliotecas Python que disponibilizam recursos para a análise de mobilidade. Posteriormente, as bibliotecas foram classificadas e associadas a fases específicas da aquisição de conhecimento de dados de trajetórias, seguindo o arcabouço apresentado na Figura 4.

A combinação dessas fases em uma metodologia ou *pipeline*, juntamente com os resultados obtidos na comparação, foi exemplificada por meio de um caso de uso. Esse caso de uso ilustrou resultados extraídos de uma base de dados real e disponibilizou o código-fonte⁴ utilizado, demonstrando a relevância desses métodos na aquisição de conhecimento em dados de mobilidade.

Os *insights* gerados pela comparação das bibliotecas revelaram várias informações importantes. Foi identificada a quase ausência de suporte para tarefas de geração de trajetória baseados em modelagem de conjuntos de dados e a inexistência de bibliotecas que permitam a mineração de movimentação conjunta (Moving Together). Além disso, observou-se a falta de suporte para a classificação de trajetórias e *map-matching*. Também foi notada uma escassez de soluções para análise de privacidade, predição e detecção de trajetórias atípicas.

Embora essas lacunas tenham sido abordadas, é importante mencionar uma limitação deste trabalho, que consiste na não exploração das diferenças entre os métodos disponíveis nas bibliotecas ou na busca por discrepâncias entre os métodos implementados e os descritos na literatura. Portanto, trabalhos futuros que abordem essas lacunas de conhecimento são desejáveis.

Em resumo, a comparação, a metodologia e o caso de uso apresentados neste capítulo proporcionam auxílio na tomada de decisões relacionadas ao uso dos recursos disponíveis nas bibliotecas Python para análise de mobilidade. Além disso, as lacunas identificadas revelam oportunidades para o avanço da pesquisa nessa área, incentivando o desenvolvimento de métodos mais abrangentes e aprimorados.

⁴ <https://github.com/edgarsoliveira1/MINICURSO-ERCEMAPI2023>

4 Comparação de métodos para detecção de pontos de parada

No capítulo anterior foi possível visualizar a cobertura de cada biblioteca Python em relação às tarefas necessárias para mineração de trajetória segundo um paradigma similar ao estabelecido em (ZHENG, 2015). Contudo, a comparação realizada limitou-se apenas à presença ou não de soluções para cada tarefa, sem se aprofundar em cada algoritmo, implementado nas bibliotecas ou descrito na literatura.

Uma das etapas mais cruciais e fundamentais nesse processo é a classificação dos momentos em que um objeto está em movimento ou em repouso, denominada detecção de pontos de parada.

Embora a literatura apresente inúmeros métodos propostos para solucionar essa tarefa, como os descritos em (LI et al., 2008; NOGUEIRA et al., 2018; ASLAK; ALESSANDRETTI, 2020; CUSTERS et al., 2021; DUARTE; SAKR, 2023), poucos trabalhos concentram-se na comparação abrangente dessas abordagens. Essa lacuna na pesquisa representa uma oportunidade para aprimorar a análise e a compreensão dos métodos, identificando suas fraquezas e pontos fortes, bem como suas características distintas que podem favorecer um método em detrimento de outro, em diferentes contextos.

Portanto, este capítulo se propõe a realizar uma comparação profunda entre métodos descritos na literatura para detecção de pontos de parada.

4.1 Trabalhos Relacionados

Como mencionado, muitos trabalhos apresentam métodos para solucionar o problema de detecção de parada. No entanto, é comum encontrar premissas sobre o tipo de dados, seu estado e a forma como foram coletados e processados. Além disso, a literatura carece de uma análise focada na comparação profunda de métodos, o que se torna evidente nos trabalhos listados a seguir.

4.1.1 Infostop

A detecção de pontos de parada é uma tarefa crucial na mineração de mobilidade, e o algoritmo de Hariharan e Toyama (HARIHARAN; TOYAMA, 2004) destaca-se como um dos mais utilizados para esse propósito. Esse algoritmo é dividido em duas etapas distintas. Na primeira fase, ele identifica pontos de parada considerando períodos em que um indivíduo não se afasta além de uma distância máxima, r_1 , por uma duração

mínima, t_{min} , ambos definidos como parâmetros. Na segunda etapa, os pontos de parada são agrupados, sendo o DBSCAN (ESTER et al., 1996a) a versão mais empregada para essa finalidade.

No entanto, surge o Infostop (ASLAK; ALESSANDRETTI, 2020) como uma nova alternativa na segunda etapa, superando algumas limitações. Diferentemente do DBSCAN, o Infostop evita a mesclagem de pontos de parada muito próximos, o que possibilita uma análise mais precisa das trajetórias de múltiplos usuários simultaneamente. Além disso, destaca-se por executar o segundo passo do algoritmo de forma mais eficiente, resultando em tempos de processamento inferiores aos de outros métodos disponíveis.

Uma área de melhoria perceptível deste trabalho é a inclusão de uma comparação entre diversos métodos de detecção de paradas, indo além das variações do método apresentado por Hariharan e Toyama.

4.1.2 Detecção de pontos de parada para mineração de similaridade entre usuários

No artigo (LI et al., 2008), é apresentado um método para demonstrar a similaridade entre usuários por meio do histórico de localizações. Para isso, são executados dois passos distintos: no primeiro passo, o artigo introduz um algoritmo destinado à detecção de pontos de parada; no segundo passo, é proposto um método de agrupamento dos pontos de parada em camadas, permitindo, assim, a visualização de diferentes níveis de similaridade entre os usuários. No contexto deste estudo, o método de detecção de pontos de parada assemelha-se consideravelmente ao apresentado por Hariharan e Toyama, porém sem a segunda fase de agrupamento. Nesse sentido, apenas dois critérios, temporal e espacial, são empregados para determinar se um objeto permaneceu numa determinada região por um período de tempo.

4.1.3 Movement-Stop-Noise (NOGUEIRA et al., 2018)

Dentre os estudos relacionados presentes na literatura, este trabalho propõe uma abordagem baseada em métodos probabilísticos para a detecção de pontos de parada ao longo de trajetórias de objetos em movimento. A premissa fundamental desta abordagem é que os pontos que representam as trajetórias não se apresentam na mesma frequência.

O algoritmo desenvolvido é detalhadamente apresentado, destacando suas etapas e estratégias probabilísticas adotadas para a detecção de pontos de parada. Em seguida, uma seção de validação é apresentada, dividida em comparação qualitativa e quantitativa.

Na comparação qualitativa, são listados os parâmetros utilizados para classificar as trajetórias de acordo com oito algoritmos distintos. Além de parâmetros, a comparação leva em consideração o tratamento de ruído, o suporte a filtro espacial e a independência de

dados externos. Já na comparação quantitativa, o algoritmo proposto é avaliado utilizando um conjunto de dados do *dataset* Dublin Bus GPS, em conjunto com informações do OpenStreetMap. Essa abordagem de validação com dados reais proporciona uma avaliação mais precisa do desempenho do algoritmo em cenários reais. No entanto, é importante mencionar que os resultados de outros algoritmos similares não foram incluídos nessa comparação, o que pode ser considerado uma limitação do estudo.

Uma possível melhoria metodológica discutida no artigo é o emprego do GTFS (General Transit Feed Specification) como fonte adicional de informação para a validação de pontos de parada.

4.1.4 *Stop Go Classifier* (SPANG et al., 2022a)

Este trabalho apresenta um método para a detecção de pontos de parada com base em uma análise geométrica da trajetória. Para a detecção, o sistema inclui quatro métodos independentes de classificação, que comparam diferentes aspectos das propriedades geométricas de uma trajetória. Além disso, essa abordagem pode fazer uso de informações como acelerômetro, quando disponíveis. Para validar a ferramenta proposta, o conjunto de dados reunidos no STAGA (SPANG et al., 2022b) foi utilizado para realizar uma comparação de resultados com métodos implementados em duas bibliotecas, Scikit-Mobility (PAPPALARDO et al., 2022) e MovingPandas (GRASER; DRAGASCHNIG, 2020).

Contudo, é possível realizar uma comparação mais abrangente, já que as ferramentas citadas apresentam suporte somente a métodos baseados em distância, duração e velocidade. Além disso, novas fontes de *ground-truth* podem ser empregadas para validar o classificador proposto.

4.1.5 Outlier Detection and Cleaning in Trajectories: A Benchmark of Existing Tools

No trabalho (DUARTE; SAKR, 2023), é apresentado um método para gerar *ground-truth*, ou seja, um método para gerar um conjunto de *targets* ou *labels* que posteriormente podem ser usados para treinar e testar um ou mais modelos, utilizando validação cruzada. Especificamente, esse método serve para gerar *labels* de 'outliers' ou 'ruído' em quatro conjuntos de dados distintos, que serão utilizados para avaliar os métodos de detecção de outliers, e limpeza de trajetória, implementados em sete bibliotecas: (i) Movetk (CUSTERS et al., 2021); (ii) Moving Pandas (GRASER; DRAGASCHNIG, 2020); (iii) Scikit-mobility (PAPPALARDO et al., 2022); (iv) PTRAIL (HAIDRI et al., 2022); (v) Pymove (BRÁZ, 2020); (vi) Argosfilter (FREITAS, 2012); (vii) Stmove (SEIDEL; DOUGHERTY; GETZ, 2019).

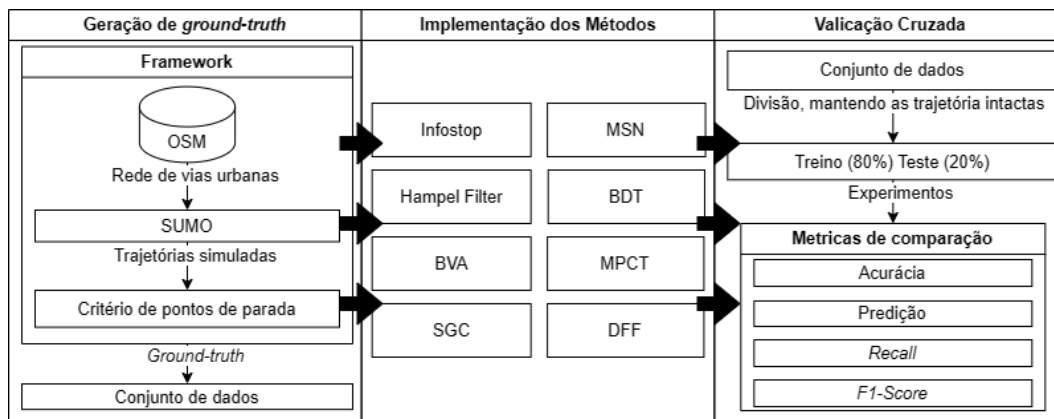
Após calcular a **acurácia**, **precisão**, **Recall** e **F-1 score**, o trabalho conclui que o MoveTk, seguido pelo Moving Pandas, apresentam os melhores resultados nos experimentos realizados. Contudo, também é mencionado que outros métodos de *ground-truth* podem ser explorados, assim como outros métodos que não estão implementados nessas bibliotecas.

Um ponto de melhoria identificado pelo *benchmark* está relacionado aos métodos de detecção de pontos de parada. Como descrito no artigo, a avaliação está centrada na detecção de *outliers* e limpeza de ruído, que são etapas de pré-processamento, diferentes da detecção de pontos de parada (ZHENG, 2015). Desta forma, o estado da arte ainda carece de uma análise focada em métodos para a detecção de pontos de parada.

4.2 Metodologia

Nesta seção são apresentados os conjuntos de dados usados nos experimentos e a técnica de *ground-truth* aplicada. Além disso, são detalhadas as técnicas que serão comparadas e, quando necessário, alguns detalhes de implementação. Por fim, estão descritas as métricas de comparação. Uma imagem da estrutura geral da metodologia implementada encontra-se na Figura 15.

Figura 15 – Estrutura geral da metodologia implementada.

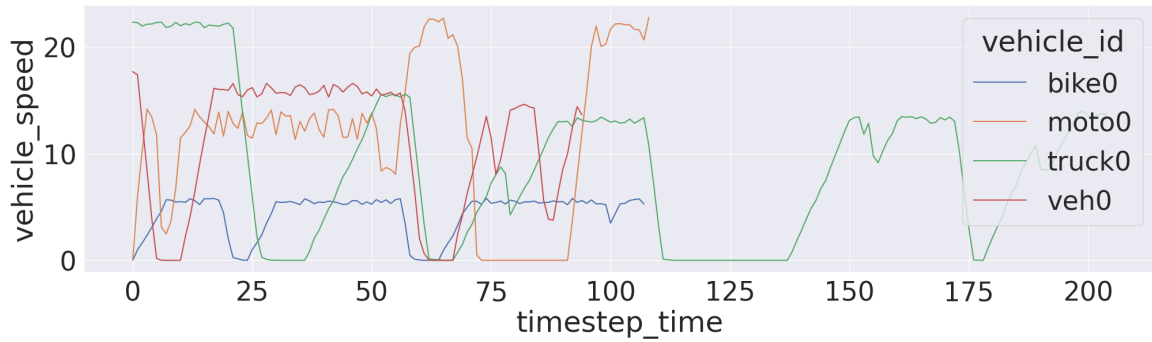


4.2.1 Aquisição dos dados ou conjunto de dados

Nesta seção são apresentados os problemas nas opções disponíveis para *ground-truth* e o método utilizado e implementado neste capítulo.

No estudo de (SPANG et al., 2022b), um conjunto de dados de trajetórias GPS com anotações de pontos de parada foi registrado por meio de um aplicativo durante 126 dias. Os usuários do aplicativo, que eram voluntários com idades entre 25 e 35 anos, trabalhavam em tempo integral de forma presencial e residiam principalmente em áreas urbanas da Europa. As trajetórias coletadas foram realizadas principalmente de bicicleta, embora

Figura 16 – Amostra dos dados gerados.



também tenham sido registradas viagens de carro e trem. O trabalho também incluiu pós-processamentos, como deslocamento temporal, espelhamento, rotação e deslocamento espacial, com o objetivo de proteger a privacidade dos voluntários. Todos esses dados foram coletados e disponibilizados com a finalidade de fornecer *ground-truth* para treinar e/ou testar modelos de detecção de pontos de parada. No entanto, é importante observar que os dados disponibilizados não contêm informações sobre o início e o fim das trajetórias, nem a identificação de objetos, o que pode dificultar sua usabilidade em algumas aplicações, como validação cruzada.

Para uma comparação justa entre os métodos de detecção de paradas, é necessário um conjunto de dados confiáveis. A ferramenta de código aberto *Simulation of Urban MObility* (SUMO) (LOPEZ et al., 2018) é amplamente utilizada para modelar e simular o tráfego urbano e sistemas de transporte. Desenvolvido pelo Instituto Alemão de Tecnologia dos Transportes (*German Aerospace Center - DLR*), o SUMO oferece uma plataforma flexível para simular o comportamento de veículos, pedestres e outros agentes de tráfego em ambientes urbanos e rodoviários. É amplamente utilizado por pesquisadores, engenheiros de tráfego, planejadores urbanos e desenvolvedores de sistemas de transporte para analisar o desempenho de infraestruturas viárias, estudar congestionamentos, testar algoritmos de controle de tráfego e avaliar o impacto de medidas de gerenciamento de tráfego e planejamento urbano.

Aproveitando essa ferramenta, juntamente com as informações de infraestrutura de mobilidade disponíveis no OpenStreetMap (OSM) (VARGAS-MUNOZ et al., 2020), foi gerado e disponibilizado um conjunto de dados, exemplificado por uma amostra na Figura 16, para realizar os testes e comparações dos algoritmos de detecção de paradas. Por fim, também é disponibilizado o *framework* Python usado para realizar o fluxo de trabalho, incluindo a aquisição de dados do OSM, simulação usando o SUMO e a anotação dos pontos de parada com base em um critério de velocidade definido pelo usuário. Para esta comparação, o critério de parada utilizado foi o de momentos em que os veículos simulados possuem velocidade igual a zero.

4.2.2 Detecção de Parada

Nesta seção, apresenta-se os métodos e detalhes de implementação, escolhidos para esta análise. A maior parte dos métodos listados, a seguir, foram implementados na linguagem de programação Python, e serão disponibilizados¹.

4.2.2.1 Movement-Stop-Noise

O algoritmo *Movement-Stop-Noise* envolve um processo de classificação de segmentos de trajetória em categorias de movimento, parada ou ruído. Inicialmente, ele realiza cálculos para determinar métricas como distância percorrida, duração, velocidade e ângulos de curva, que representam os ângulos entre três pontos vizinhos na trajetória. Essas métricas são utilizadas para categorizar os segmentos com base em limiares lineares, que são definidos como padrão em 3.5 para cada métrica. O algoritmo permite a personalização dos limiares e a inclusão de um ângulo mínimo opcional para aprimorar a detecção de ruídos. Neste trabalho, os limiares escolhidos foram zero para todas as métricas.

4.2.2.2 Hampel Filter

O Hampel Filter (WELCH; BISHOP et al., 1995) é uma técnica de filtragem robusta amplamente utilizada em processamento de séries temporais para detecção e suavização de *outliers*, pontos de dados que se desviam significativamente do comportamento esperado da série. Quando aplicado à análise de trajetórias, o Hampel Filter pode ser usado para identificar pontos de parada ao detectar valores discrepantes nos dados de localização. Funciona da seguinte maneira: em uma trajetória, os pontos de dados correspondem às coordenadas espaciais ou temporais da posição de um objeto. O filtro calcula uma estatística de dispersão, como a média móvel ponderada, para uma janela deslizante de pontos de dados. Quando um ponto de dados está muito afastado da estatística de dispersão calculada, é considerado um *outlier* e, portanto, pode ser identificado como um ponto de parada.

4.2.2.3 Baseado em distância, tempo, velocidade e aceleração

Um algoritmo eficaz para identificar pontos de parada em uma trajetória se baseia em um simples, mas poderoso conceito de *threshold*, que pode comparar diversos parâmetros, como distância percorrida, tempo decorrido, velocidade e aceleração. Utilizando uma abordagem de análise sequencial dos dados de movimento, o algoritmo examina cada ponto ao longo da trajetória, calculando a distância entre pontos consecutivos, medindo o tempo decorrido entre eles e derivando informações de velocidade e aceleração. Em seguida, estabelece limites para esses parâmetros, permitindo a detecção de pontos de parada com base em quando a velocidade cai abaixo de um certo valor mínimo, a aceleração se

¹ <https://github.com/edgarsoliveira1/ARTIGO-ERCEMAPI2023>

aproxima de zero e a distância percorrida fica relativamente constante por um período de tempo determinado. Os algoritmos que seguem esta abordagem estão entre os mais difundidos.

4.2.3 *Maximum Physically Consistent Trajectories*

Outra forma de identificar pontos de parada é através da verificação da consistência da trajetória sobre um modelo físico (CUSTERS et al., 2021). Neste trabalho uma versão deste algoritmo foi implementada usando como modelo físico um limite mínimo de velocidade. Neste método, a quantidade de parâmetros varia de acordo com o modelo físico escolhido, por exemplo, na implementação desta comparação para identificar pontos de parada, apenas um limite inferior de velocidade, *threshold* igual a zero, foi usado. Desta forma, o algoritmo busca a maior sub-trajetória consistente com o modelo, para identificar os pontos de parada.

4.2.4 *Deep Feed Forward*

A tarefa de detecção de pontos de parada também pode ser encarada como um problema de classificação de trajetória. Desta forma, é possível aplicar uma rede neural simples, utilizando uma parte do conjunto de dados para treinar o modelo e outra parte para validação. Neste caso, o tipo de rede neural utilizado é um modelo que não faz uso dos pontos classificados anteriormente, e, portanto, pode resultar em um desempenho pior em comparação com modelos que fazem uso de resultados anteriores.

Por fim, os algoritmos avaliados nesta comparação são: *Movement-Stop-Noise* (MSN) (NOGUEIRA et al., 2018); Infostop (ASLAK; ALESSANDRETTI, 2020); Hampel Filter (HF); Baseado em distância e tempo (BDT)(LI et al., 2008); Baseado em velocidade e aceleração (BVA); *Maximum Physically Consistent Trajectories* (MPCT) (CUSTERS et al., 2021); *Stop Go Classifier* (SGC); e *Deep Feed Forward* (DFF).

4.2.5 Métricas de comparação

O conjunto de trajetórias geradas anteriormente foi dividido em duas partes para viabilizar a validação cruzada. A primeira, chamada de treino, que contém 80% das trajetórias e foi usada para treinar ou buscar os parâmetros mais adequados para a detecção. Uma vez treinados ou definidos os parâmetros, a segunda parte, chamada de teste, é usada para computar diversas métricas, que descrevemos a seguir. Por fim, é importante destacar que as trajetórias foram mantidas intactas no processo de divisão e, desta forma, não trazendo distorções para o desempenho dos algoritmos.

4.2.5.1 Métricas de avaliação

Acurácia: No contexto de detecção de pontos de parada em trajetórias, a acurácia representa a proporção de trajetórias que foram corretamente classificadas como "parada" ou "não parada" pelo modelo de detecção. Uma alta acurácia indica que o modelo está acertando a maioria dos pontos, identificando quando um veículo ou objeto está em um ponto de parada e quando não está.

Precisão: A precisão no contexto de detecção de pontos de parada refere-se à capacidade do modelo de evitar a classificação errônea de trajetórias como pontos de parada quando, na verdade, não são. Isso é crucial para garantir que não haja alarmes falsos, o que pode ser problemático em sistemas de monitoramento de tráfego ou logística, onde a identificação incorreta de paradas pode levar a decisões inadequadas.

Recall (Revocação): O *recall* é particularmente importante na detecção de pontos de parada em trajetórias, pois mede a capacidade do modelo de identificar todas as paradas reais. Em outras palavras, quantas das verdadeiras paradas foram corretamente identificadas pelo modelo. Um *recall* alto é essencial para garantir que não ocorram falsos negativos, ou seja, que nenhum ponto de parada real seja deixado de fora da detecção.

F1-Score: O F1-Score, no contexto de detecção de pontos de parada, avalia o equilíbrio entre a precisão e o *recall*. Um alto F1-Score indica que o modelo é capaz de encontrar a maioria das paradas reais enquanto mantém um baixo número de falsos positivos. Isso é crucial para sistemas de rastreamento de veículos, onde a identificação precisa de paradas é essencial para otimizar rotas e melhorar a eficiência operacional.

No contexto de pontos de parada, onde existem apenas dois estados para um ponto, parado ou não parado, e a quantidade de pontos de parada é frequentemente inferior à quantidade de pontos não parados, a métrica F1 tende a ter maior peso. Isso ocorre porque ela exige que o algoritmo classifique corretamente os pontos de parada e não classifique os pontos não parados de forma equivocada.

4.3 Resultados e Discussões

Nesta seção são apresentados os resultados obtidos a partir da realização de uma avaliação crítica das métricas coletadas através da aplicação da metodologia descrita anteriormente. Tais resultados, podem ser encontrados de forma resumida na Tabela 13 e na Figura 17.

Como pode-se observar, na Tabela 13, o algoritmo que obteve o melhor desempenho foi o Baseado em Velocidade e Aceleração, critério usado para a criação do *ground-truth* no conjunto de dados. No entanto, mesmo utilizando esse critério, não foram alcançadas métricas perfeitas, o que demonstra a não trivialidade da tarefa e a confiabilidade dos

Figura 17 – Comparação de Métricas por Método.

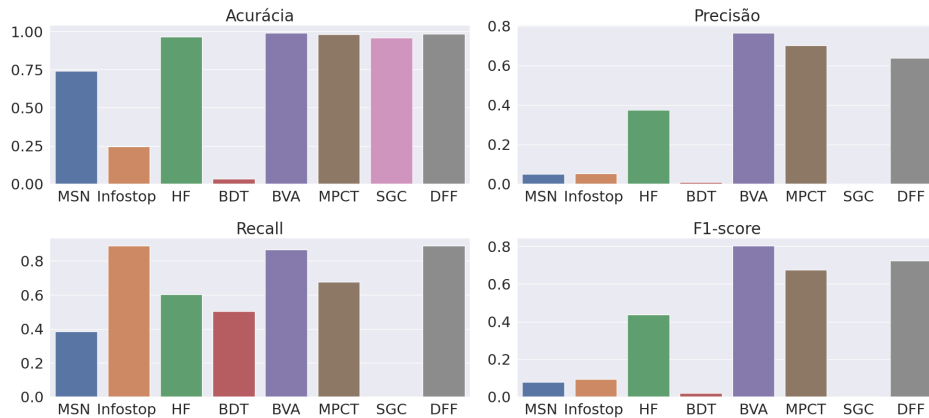


Tabela 13 – Resultados

Método	MSN	Infostop	HF	BDT	BVA	MPCT	SGC	DFF
Acurácia	0.743	0.246	0.965	0.033	0.991	0.983	0.961	0.986
Precisão	0.051	0.053	0.376	0.010	0.765	0.701	0.000	0.639
Recall	0.386	0.889	0.603	0.504	0.865	0.677	0.003	0.889
F1-scores	0.079	0.093	0.438	0.020	0.803	0.676	0.000	0.723

dados gerados pelo *framework* criado. Em relação ao BVA, os algoritmos DFF e o MPCT apresentaram resultados comparáveis. Além disso, o método DFF apresentou um *Recall* superior ao BVA, sem uma queda considerável de Precisão, indicando um resultado melhor quando há um peso maior para verdadeiros positivos.

A ferramenta Infostop apresentou um valor de *Recall* tão alto quanto DFF ao custo de uma queda considerável de Precisão. Portanto, o método considerou muitos pontos 'não parados' como 'parados' gerando um número considerável de falsos positivos.

Portanto, é desejável que trabalhos futuros aprimorem essa análise, incluindo não apenas novos métodos, mas também avaliando o impacto da combinação desses métodos com a aplicação de técnicas de pré-processamento, como o filtro de Kalman, por exemplo. Além disso, é importante realizar comparações que levem em consideração o tipo de ponto de parada, ou seja, a motivação por trás da interrupção do objeto. Essa comparação exigirá a criação de um *ground-truth* específico para esse propósito. Por fim, o desempenho razoável do uso de uma rede neural simples, DFF, torna desejável novas comparações que apliquem o uso de redes neurais, como Redes Neurais Recorrentes, e validem os modelos treinados com dados reais.

4.4 Resultados

Neste capítulo, foi apresentado e disponibilizado um *framework* capaz de criar *ground-truth*, utilizando dados do OpenStreetMap e do SUMO (Simulation of Urban

MObility), para a detecção automática de pontos de parada, assim como o código-fonte implementado². Usando um conjunto de dados gerados por meio desse *framework*, foram implementados e comparados alguns dos métodos mais utilizados descritos na literatura. Os resultados confirmaram trabalhos anteriores, como (DUARTE; SAKR, 2023), e introduziram novos algoritmos a comparação, SGC, MSN e DFF. Contudo, é importante destacar também que pontos de melhoria foram identificados, como a realização de novas análises que considerem métodos diferentes de aquisição de dados e seus impactos.

² <https://github.com/edgarsoliveira1/ARTIGO-ERCEMAPI2023>

5 MoveMiner: Uma biblioteca para mineração de trajetória

A partir das informações obtidas no Capítulo 3, é perceptível que as bibliotecas analisadas não possuem uma cobertura completa das tarefas necessárias para a mineração de dados de trajetória. Portanto, neste capítulo, é apresentada uma biblioteca capaz de preencher essa lacuna no ecossistema Python.

5.1 Tarefas e Bibliotecas

Através da comparação apresentada anteriormente, no Capítulo 3, algumas informações ficam evidentes. Algumas tarefas sequer possuem suporte de qualquer uma das bibliotecas apresentadas. Além disso, em outros casos, é necessária a integração entre diferentes bibliotecas para realizar uma etapa por completo, o que pode não ser uma tarefa trivial. Outra desvantagem que pode ser gerada pela combinação de diferentes bibliotecas é a necessidade de modificar a forma como os dados de mobilidade são representados para a integração entre as bibliotecas, o que pode gerar complexidade e redução no desempenho do código de forma geral.

Como é evidente ao somar as bibliotecas com suporte para cada tarefa e como está visível na Tabela 14. Além disso, é perceptível que algumas tarefas não possuem suporte, como *Moving Together*, Classificação e *Map-matching*.

Além disso, na Tabela 15, é possível visualizar a relação de tarefas suportadas por cada biblioteca. Observa-se que três bibliotecas, Pymove, MovingPandas e Scikit-Mobility, possuem uma maior cobertura de tarefas entre as demais e serão utilizadas para comparar os resultados obtidos com a biblioteca MoveMiner na próxima seção.

5.2 MoveMiner

A biblioteca MoveMiner tem como objetivo principal preencher a lacuna de uma biblioteca Python *open-source* para mineração de dados de trajetória. Desta forma, evita a necessidade de integrar mais de uma biblioteca para realizar todos os processos estabelecidos pelo arcabouço, inspirado pelo paradigma de (ZHENG, 2015), descrito na Figura 4.

Como é possível visualizar na Figura 18, a biblioteca apresentada possui uma subdivisão de módulos coerente com o arcabouço, e portanto apresenta ao menos uma solução para cada tarefa citada. Portanto, oferece soluções para tarefas que não são

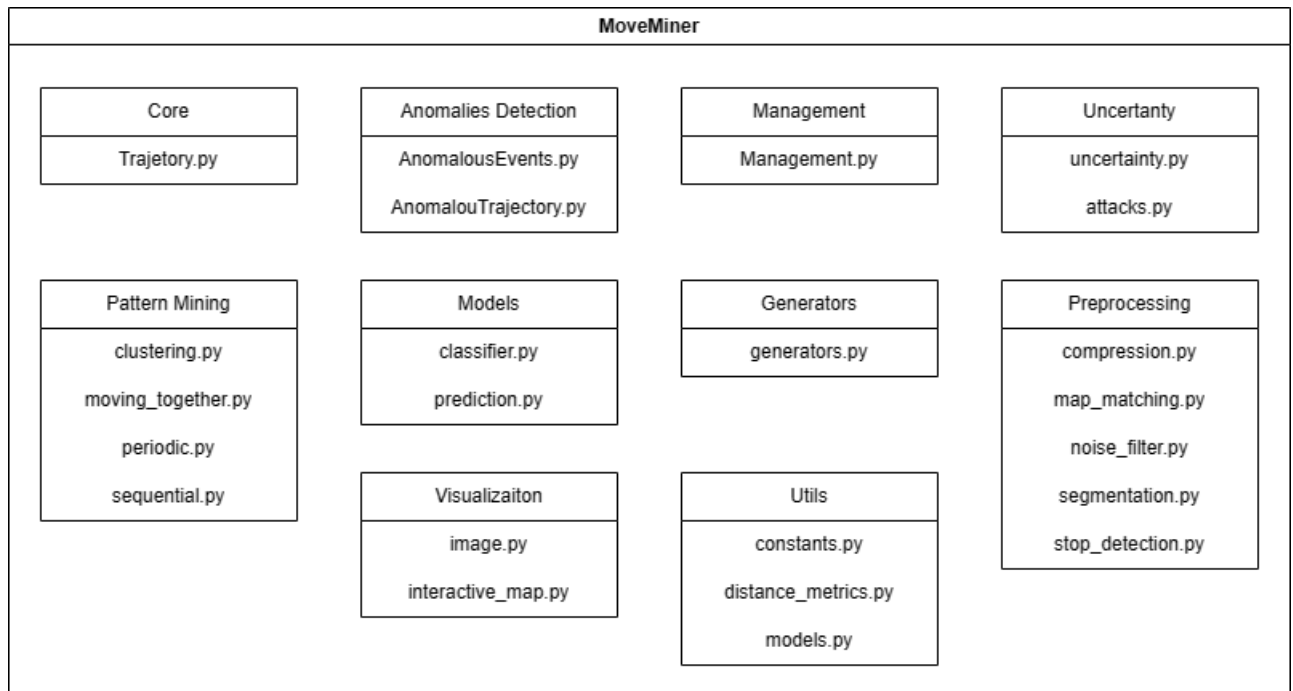
Tabela 14 – Número de bibliotecas com suporte para cada tarefa.

Etapas	Tarefas	Número de bibliotecas
Tipo de entidades	Trajetoária	14
móveis	Filtragem de ruídos	6
Técnicas de pré-processamento	Deteção de pontos de parada	7
	Compressão de trajetórias	6
	Segmentação de trajetória	5
	<i>Map-matching</i>	0
Recursos para gerenciamento de dados	Consultas em Intervalo	6
	Consultas KNN	2
	Similaridade de Trajetórias	2
Métodos para lidar com incerteza	Reduzindo a Incerteza	3
	Análise de Privacidade	1
Gerar mobilidade	Gerar Trajetória	4
Técnicas de mineração de padrões	<i>Moving Together</i>	0
	Agrupamento de Trajetórias	9
	Padrões Sequenciais	1
	Padrões Periódicos	1
Técnicas para predição ou classificação de trajetórias	Predição	1
	Classificação	0
Técnicas para detecção de anomalias	Detectando Trajetórias Atípicas	1
	Deteção de Eventos Atípicos	1
Bibliotecas para visualização de mobilidade	Imagens	12
	Gráficos	12
	Mapas Interativos	7

Tabela 15 – Número de tarefas suportadas por cada biblioteca.

Bibliotecas	Número de tarefas suportadas
PyMove	14
Scikit-Mobility	13
MovingPandas	12
Traja	11
Mobilipy	8
PTRAIL	8
Tracktable	8
Mobilkit	7
Yupi	5
MobVis	5
Teetool	5
Trackintel	4
Mobipy	3
SMAFramework	3
MOCHA	3

Figura 18 – MoveMiner - Estrutura de Pastas.



Fonte: Autores.

suportadas por outra biblioteca comparada no Capítulo 3. Além disso, em algumas tarefas as soluções presentes na MoveMiner podem apresentar vantagens em relação às disponibilizadas por outras bibliotecas, como será explorado a seguir.

5.2.1 Representação de Trajetória

Antes de seguir para o pré-processamento de dados, é importante explicar a forma como as trajetórias são representadas. Uma parte considerável das bibliotecas que se propõem à análise de trajetórias possuem uma forma única de representação de trajetórias. No caso da MoveMiner, a representação da trajetória é responsabilidade da classe *Trajectory*.

O uso da biblioteca GeoPandas é comum entre as bibliotecas mais populares para armazenamento de trajetórias, como o MovingPandas (GRASER; DRAGASCHNIG, 2020) e o Scikit-Mobility (PAPPALARDO et al., 2022). É possível usar uma instância do tipo Point da biblioteca Shapely para que cada linha da tabela represente um ponto de uma trajetória, como ocorre na MoveMiner e em aplicações como o MovingPandas.

Uma vantagem adicional de usar um GeoDataFrame é a capacidade de filtrar os pontos dentro de limites espaciais. A indexação temporal também é possível, combinando isso com a capacidade de filtrar os pontos. Isso permite gerenciar e filtrar os dados de forma eficiente para futuras análises.

Outro ponto importante é a forma como as distâncias entre dois pontos são calculadas por padrão. Caso os pontos da trajetória sejam geoespaciais, a distância calculada será a *haversine*, usando um elipsoide de referência. Caso contrário, a distância calculada será a euclidiana. Isso é indicado pelo atributo *geo*.

5.2.2 Pré-processamento

Após a representação da trajetória, é necessário realizar tarefas de pré-processamento, como a filtragem de ruídos, segmentação, compressão, *map-matching* e detecção de paradas.

5.2.2.1 Filtragem de ruídos

O método de detecção de ruído usado é o algoritmo descrito na Subseção 4.2.3, que é capaz de detectar a maior subsequência coerente com um modelo físico. Neste caso, está sendo usado com um limite simples de velocidade máxima. Este algoritmo também foi comparado com outros métodos para detecção de *outliers* (DUARTE; SAKR, 2023), onde, após testes em 4 *datasets*, concluiu-se que é o algoritmo mais confiável, com alto desempenho de acurácia, precisão e *recall*. No caso da comparação, a biblioteca usada foi a MoveTK (CUSTERS et al., 2021), uma biblioteca C++ que implementa o mesmo critério de velocidade usado na MoveMiner.

5.2.2.2 Segmentação

5.2.2.2.1 MoveMiner

A biblioteca, apresenta três formas de segmentar uma trajetória, espacial, temporal e por pontos de paradas. Segmentação de trajetórias por pontos de parada, é uma forma de segmentação muito comum, como citado anteriormente. Um exemplo de como a biblioteca pode ser usada para obter a segmentação de uma trajetória está disponível a seguir.

Algoritmo 1 Exemplo de Segmentação com MoveMiner

```
distance_segmentation = moveminer.DistanceSegmentation(0.1)
segs = distance_segmentation.segment(traj)
```

Figura 19 – Trajetória Original

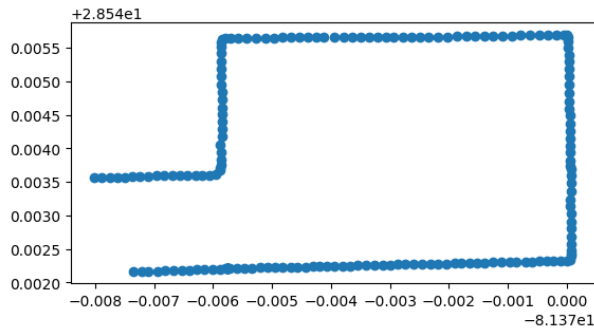
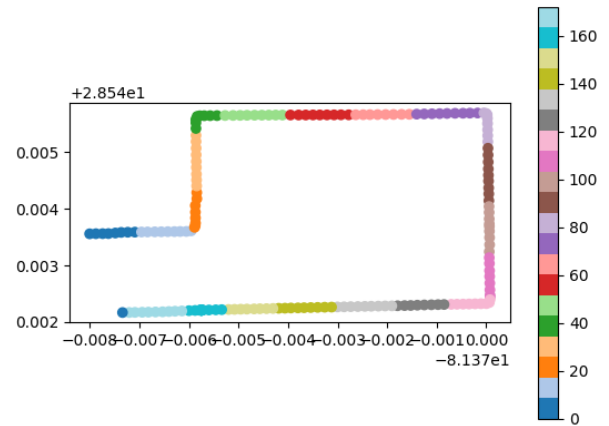


Figura 20 – Trajetória Segmentada



No Algoritmo 1, é apresentado um exemplo de como segmentar uma trajetória. Adiante, nas Figuras 19 e 20, também são apresentadas a trajetória original e a trajetória segmentada.

5.2.2.2.2 Equivalente com Pymove

Algoritmo 2 Exemplo de Segmentação com Pymove

```
segments = segmentation.by_max_dist(move_df, max_dist_between_adj
j_points=16)
```

Figura 21 – Pymove - Trajetória Original

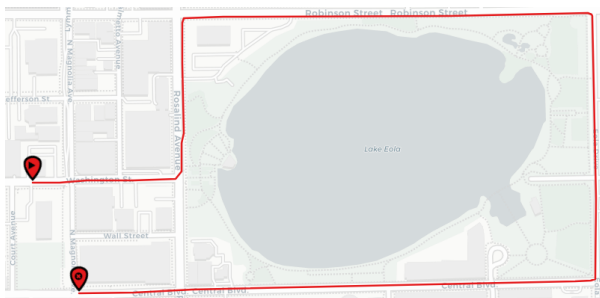
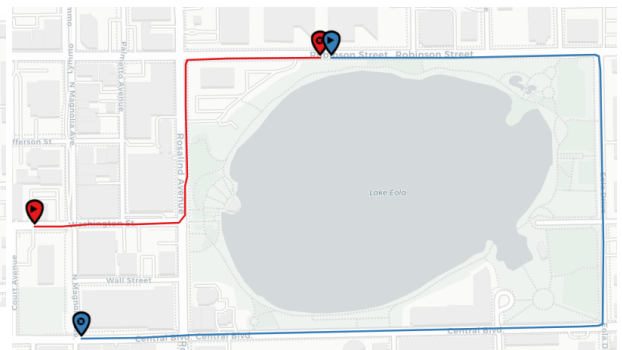


Figura 22 – Pymove - Trajetória Segmentadas



5.2.2.2.3 Equivalente com MovingPandas

Algoritmo 3 Exemplo de Segmentação com MovingPandas

```
split = mpd.StopSplitter(traj).split(min_duration=timedelta(secon
ds=0.0001), max_diameter=0.0001)
```

Figura 23 – MovingPandas - Trajetória Original

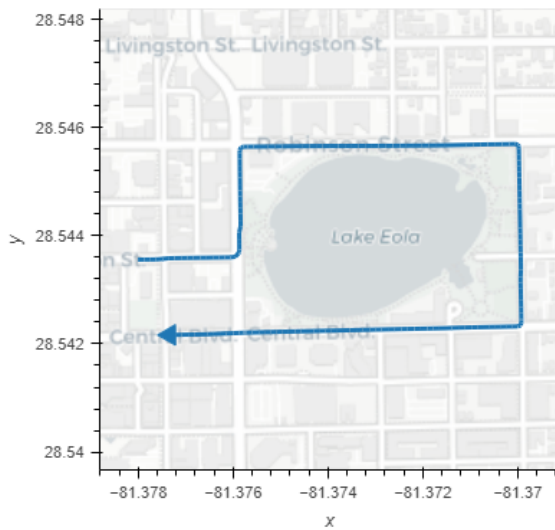
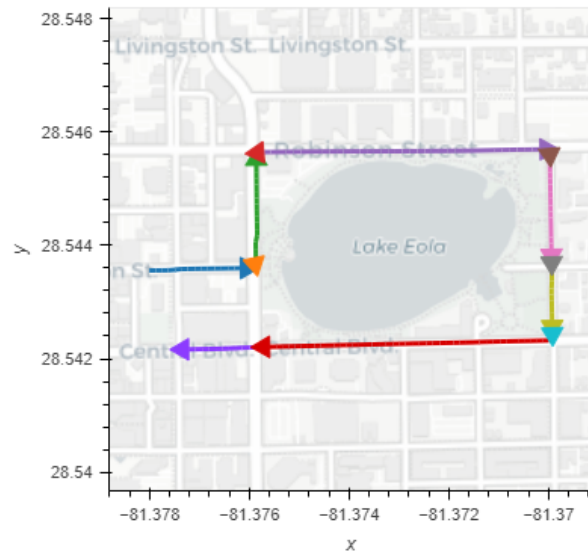


Figura 24 – MovingPandas - Trajetória Segmentadas



5.2.2.2.4 Equivalente com Scikit-Mobility

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

5.2.2.3 Compressão

5.2.2.3.1 MoveMiner

Algoritmo 4 Exemplo de compressão de trajetória com MoveMiner

```
ssc = moveminer.compression.SimpleSpatialCompression(0.2)
compressed_traj = ssc(traj)
```

Figura 25 – Trajetória Original

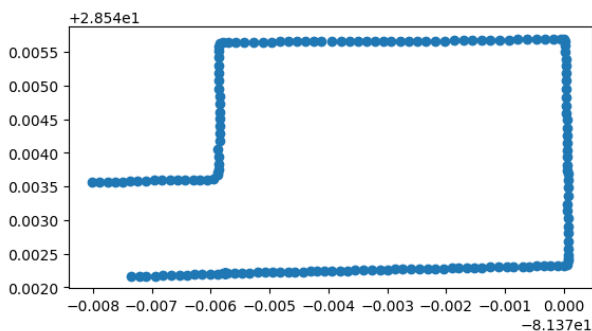
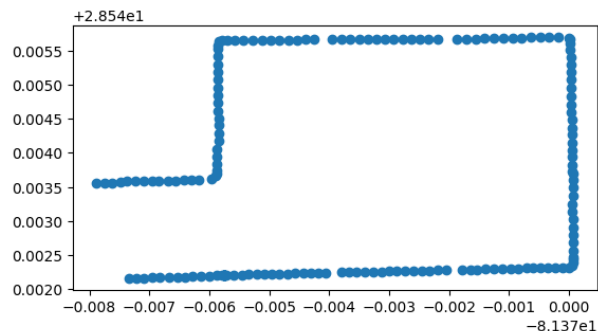


Figura 26 – Trajetória Comprimida



Como pode ser visto nas Figuras 25 e 26, a compressão remove pontos muito próximos da trajetória, sendo a proximidade excessiva determinada por um limite de distância. Resultados estes, consequentes do Algoritmo 4, acima.

5.2.2.3.2 Equivalente com Pymove

Algoritmo 5 Exemplo de compressão com Pymove

```

compressed_df = compression.compress_segment_stop_to_point(move_df)
print('{} points compressed'.format(len(move_df) - len(compressed_df)))
# 12 points compressed

```

Figura 27 – Pymove - Trajetória Original

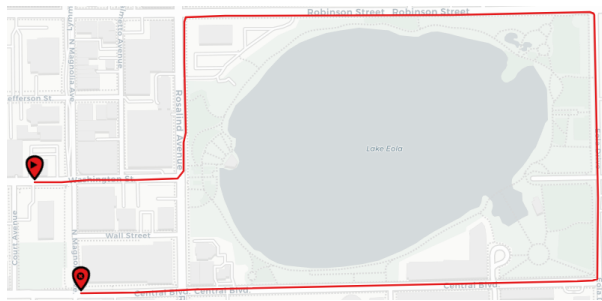
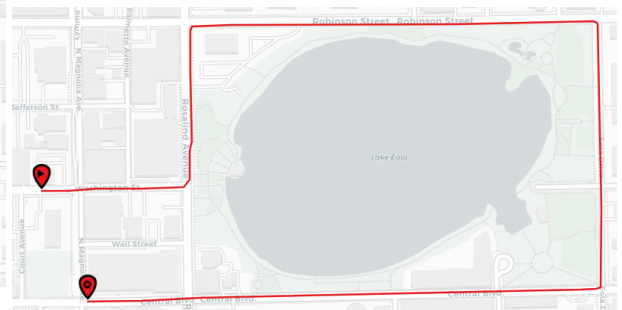


Figura 28 – Pymove - Trajetória Comprimada



5.2.2.3.3 Equivalente com MovingPandas

Algoritmo 6 Exemplo de compressão com MovingPandas

```

compressed = mpd.DouglasPeuckerGeneralizer(traj).generalize(tolerance=0.001)

```

Figura 29 – MovingPandas - Trajetória Original

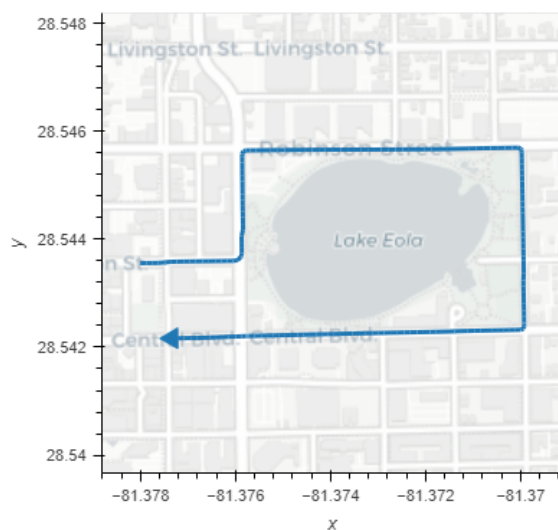
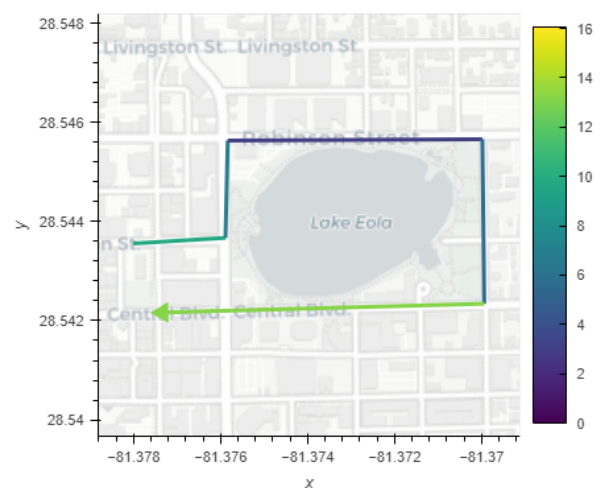


Figura 30 – MovingPandas - Trajetória Comprimada



5.2.2.3.4 Equivalente com Scikit-Mobility

Algoritmo 7 Exemplo de compressão com Scikit-Mobility

```
tdf_c = compression.compress(tdf, spatial_radius_km=0.1)
```

Figura 31 – Scikit-Mobility - Trajetória Original

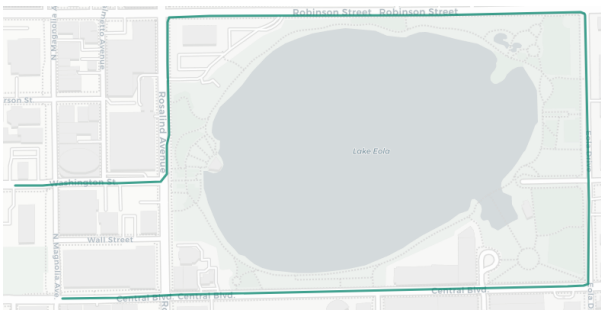
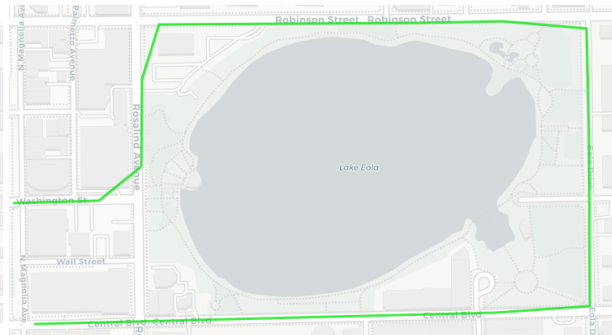


Figura 32 – Scikit-Mobility - Trajetória Comprimida



5.2.2.4 Map-Matching

5.2.2.4.1 MoveMiner

Algoritmo 8 Exemplo de *map-matching* de trajetória com MoveMiner

```
map_matching = moveminer.map_matching.MapMatching('drive')
traj_matched = map_matching.map_matching(traj_blur)
```

Figura 33 – Trajetória com *blur*

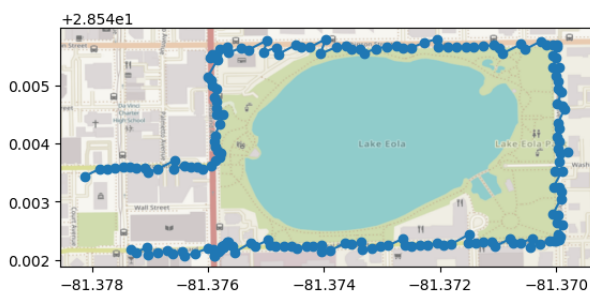
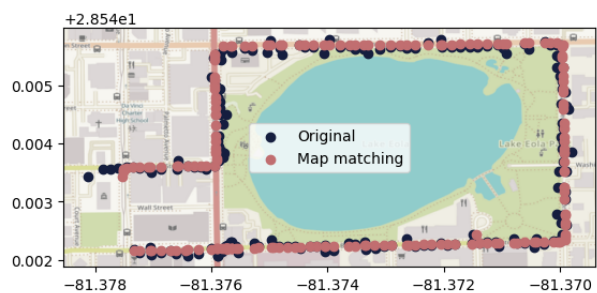


Figura 34 – Trajetória após *map-matching*



5.2.2.4.2 Equivalente com Pymove

Apesar de possuir uma *plugin*, chama *pymove-osmnx*¹, para realizar tal tarefa, O suporte encontra-se descontinuado, não está funcionando e não recebe manutenção há 3 anos. Desta forma, a biblioteca não possui um recurso para solucionar este problema.

¹ <https://github.com/InsightLab/pymove-osmnx>

5.2.2.4.3 Equivalente com MovingPandas

A biblioteca MovingPandas não possui um recurso específico para esta tarefa.

5.2.2.4.4 Equivalente com Scikit-Mobility

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

5.2.2.5 Detecção de pontos de parada

5.2.2.5.1 MoveMiner

Algoritmo 9 Exemplo de Detecção de pontos de parada com MoveMiner

```
sd_speed = moveminer.stop_detection.SDSpeed(0)
stops = sd_speed.detect_stops(traj)
```

Figura 35 – Trajetória Original

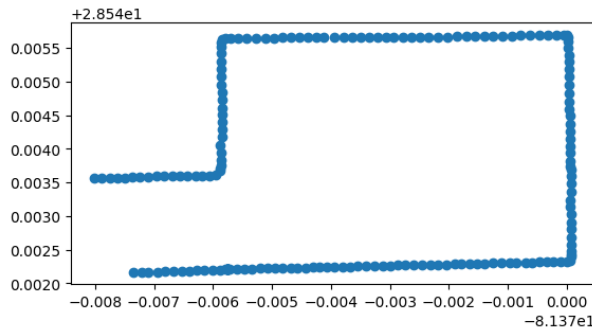
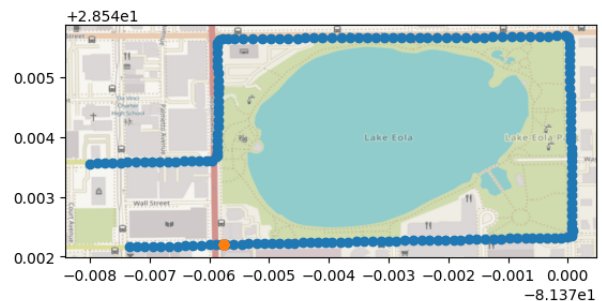


Figura 36 – Trajetória após detecção de pontos de parada



Coerente com os resultados do Capítulo 4, MoveMiner disponibiliza um método para detecção de pontos de parada baseado em velocidade, além de um baseado em distância e tempo. Contudo, soluções para outras tarefas também pode ser aplicadas neste problema como foi mostrado na comparação métodos para detecção de pontos de parada.

5.2.2.5.2 Equivalente com Pymove

Algoritmo 10 Exemplo de Detecção de pontos de parada com Pymove

```
move_df = stay_point_detection.create_or_update_move_stop_by_dist_
_time(move_df, dist_radius=0, time_radius=1)
```

Figura 37 – Pymove - Trajetória Original

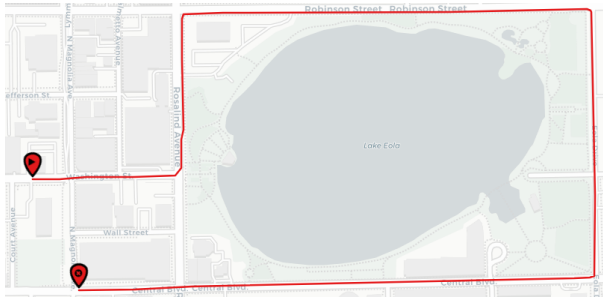
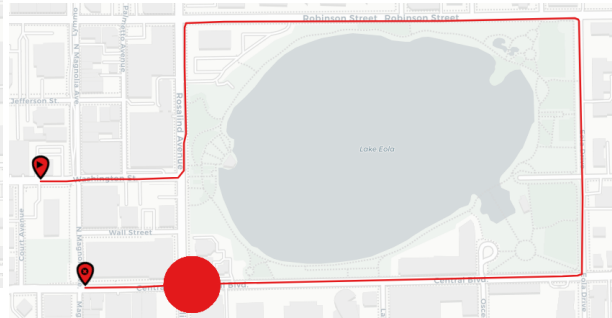


Figura 38 – Pymove - Detecção de pontos de parada



5.2.2.5.3 Equivalente com MovingPandas

Algoritmo 11 Exemplo de Detecção de pontos de parada com MovingPandas

```

detector = mpd.TrajectoryStopDetector(traj)
min_duration = timedelta(seconds=0.0001)
max_diameter = 0.0001
stop_points = detector.get_stop_points(min_duration=min_duration,
max_diameter=max_diameter)

```

Figura 39 – MovingPandas - Trajetória Original

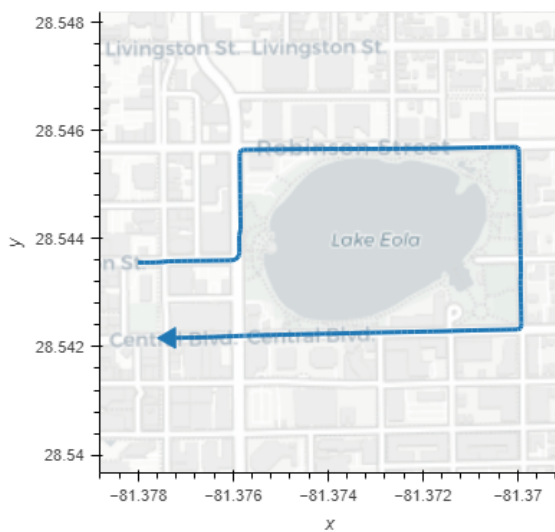
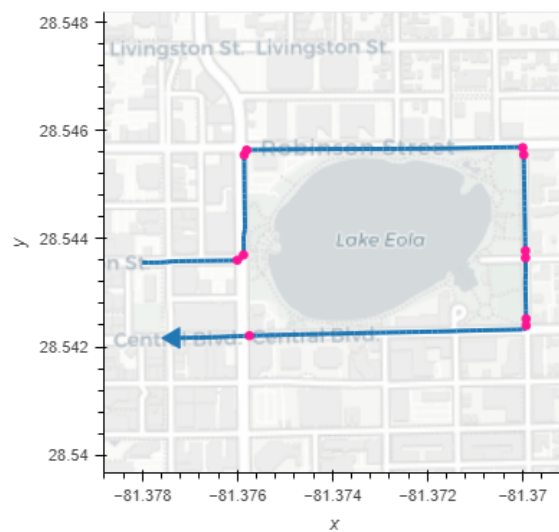


Figura 40 – MovingPandas - Detecção de pontos de parada



5.2.2.5.4 Equivalente em Scikit-Mobility

Algoritmo 12 Exemplo de Detecção de pontos de parada com Scikit-Mobility

```

tdf_s = detection.stay_locations(
    tdf_c,
    minutes_for_a_stop=0,
    spatial_radius_km=0.2,
    stop_radius_factor=0,
)

```

Figura 41 – Scikit-Mobility - Trajetória Original

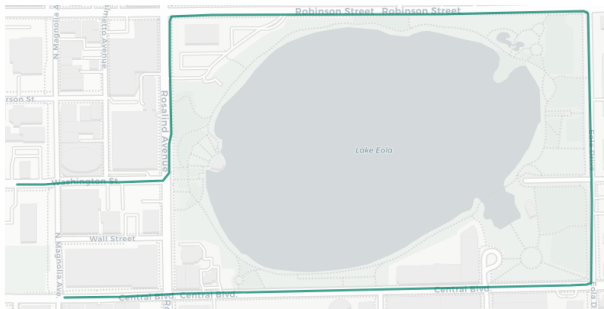
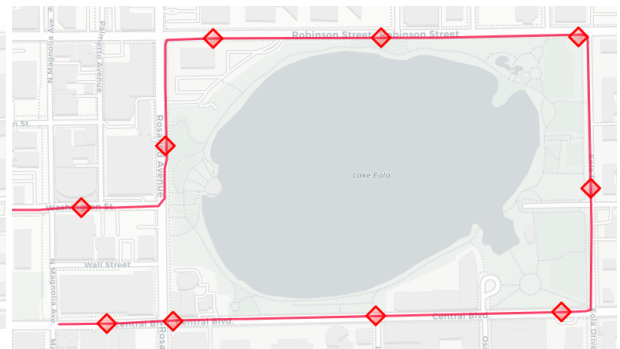


Figura 42 – Scikit-Mobility - Detecção de pontos de parada



5.2.3 Gerenciamento de Dados

Como citado anteriormente, a biblioteca proposta possui a capacidade de realizar filtragem de pontos por critérios espaciais e temporais, através do uso do GeoPandas em sua base e indexação temporal dos pontos. Além disso, a biblioteca também disponibiliza soluções para realizar buscas por similaridade e KNN, exemplificado a seguir.

5.2.3.1 Busca por critério espacial

5.2.3.1.1 MoviMiner

Algoritmo 13 Exemplo de *spacial query* em trajetória com MoveMiner

```

bbox = (-81.376006, 28.542165, -81.369924, 28.545695)
spacial_query = moveminer.management.SpacialQuery()
result = spacial_query.query(traj, bbox)

```

Figura 43 – Trajetória Original

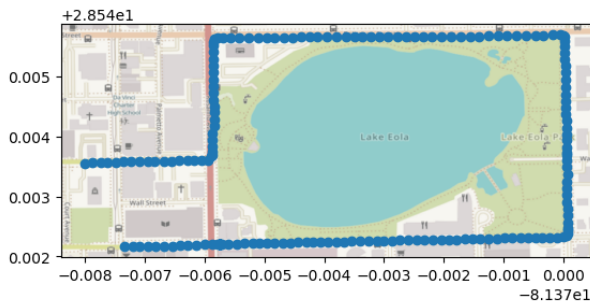
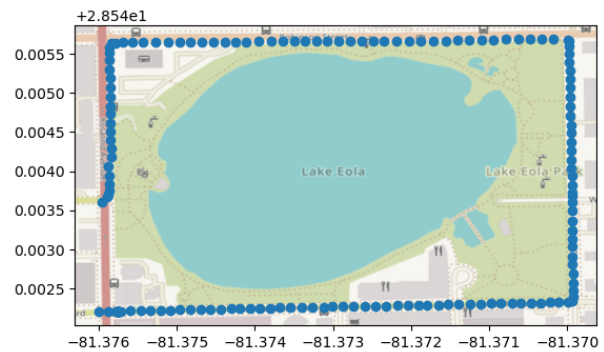


Figura 44 – Trajetória após filtragem espacial



Como pode ser visto nas Figuras 43 e 44, a busca espacial retorna os pontos de uma trajetória que estão dentro de um intervalo especificado.

5.2.3.1.2 Equivalente em Pymove

Algoritmo 14 Exemplo de *spacial query* em trajetória com Pymove

```
bbox = (28.542165, -81.376006, 28.545695, -81.369924)
filter_df = filters.by_bbox(move_df, bbox)
```

Figura 45 – Pymove - Trajetória Original

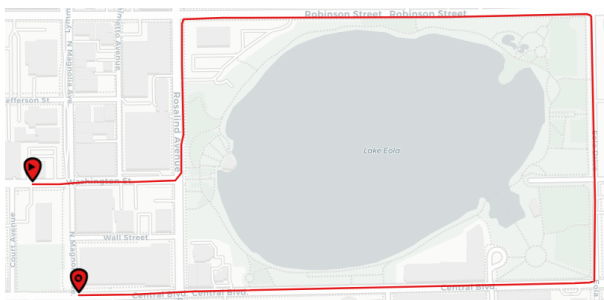


Figura 46 – Pymove - Trajetória após filtragem espacial



5.2.3.1.3 Equivalente em MovingPandas

Algoritmo 15 Exemplo de *spacial query* em trajetória com MovingPandas

```
query = traj.to_point_gdf().clip_by_rect(-81.376006,
28.542165, -81.369924, 28.545695)
```

Figura 47 – MovingPandas - Trajetória Original

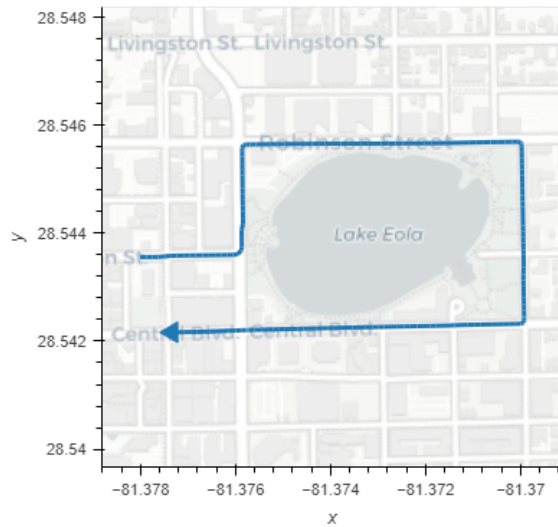
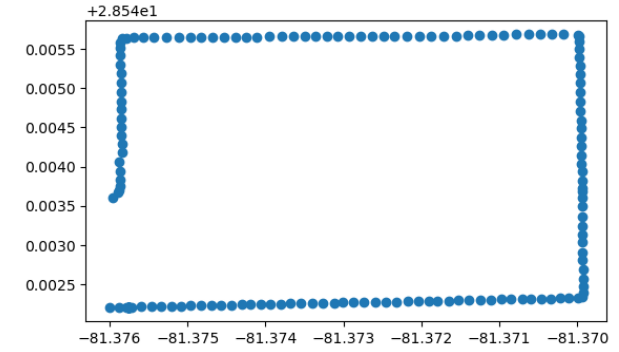


Figura 48 – MovingPandas - Trajetória após filtragem espacial



5.2.3.1.4 Equivalente em Scikit-Mobility

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

5.2.3.2 Busca por critério temporal

5.2.3.2.1 MoviMiner

Algoritmo 16 Exemplo de *time query* em trajetória com MoveMiner

```
start = '2023-11-15_18:14:26.725897'
end = '2023-11-15_18:14:41.725897'
time_query = moveminer.management.TimeQuery()
result = time_query.temporal_query(traj, start, end)
```

Figura 49 – Trajetória Original

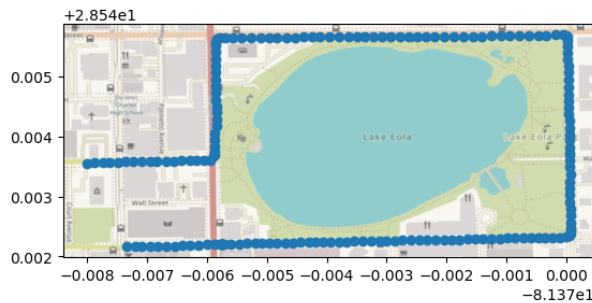
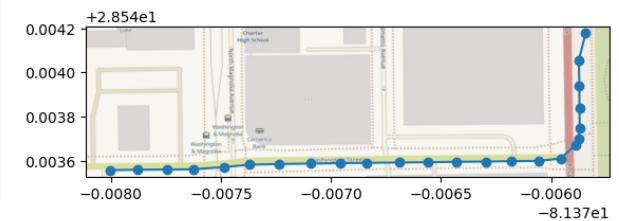


Figura 50 – Trajetória após filtragem temporal



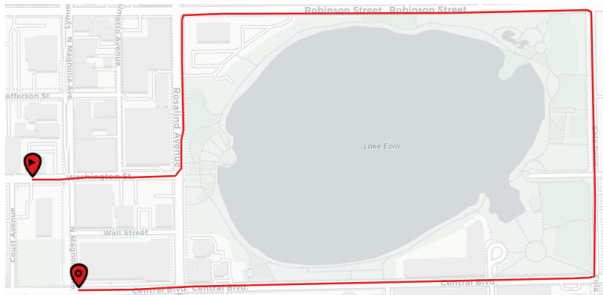
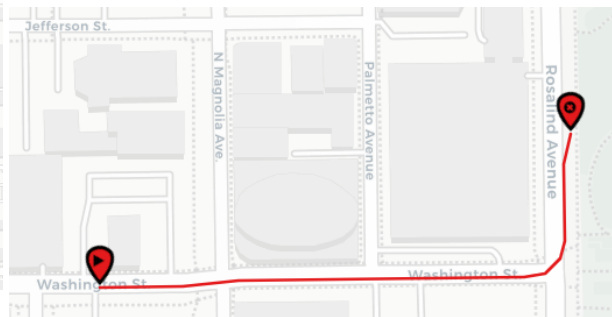
De forma semelhante à busca espacial, a busca temporal filtra os pontos de uma trajetória para retornar apenas aqueles dentro de um intervalo, como pode ser visto nas Figuras 49 e 50.

5.2.3.2.2 Equivalente em Pymove

Algoritmo 17 Exemplo de *time query* em trajetória com Pymove

```
start = '2023-11-15_18:14:26.725897'
end = '2023-11-15_18:14:41.725897'
filter_df = filters.by_datetime(move_df, start, end)
```

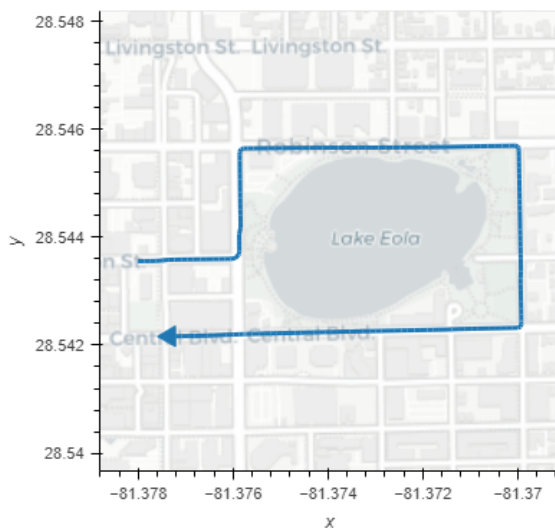
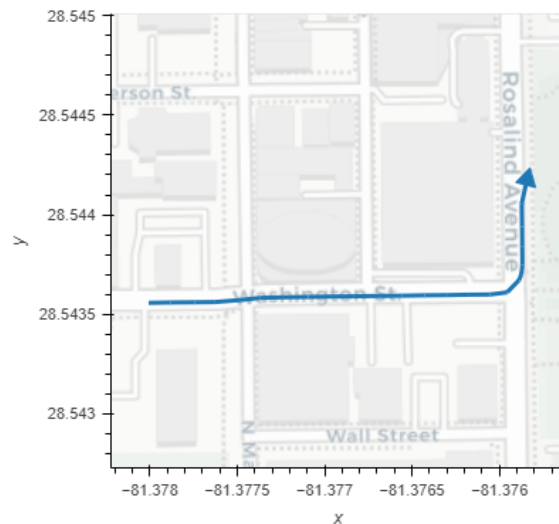
Figura 51 – Pymove - Trajetória Original

Figura 52 – Pymove - Trajetória após filtra-
gem temporal

5.2.3.2.3 Equivalente em MovingPandas

Algoritmo 18 Exemplo de *time query* em trajetória com MovingPandas

```
start = '2023-11-15_18:14:26.725897'
end = '2023-11-15_18:14:41.725897'
query = traj.get_segment_between(start, end)
```

Figura 53 – MovingPandas - Trajetória Ori-
ginalFigura 54 – MovingPandas - Trajetória após
filtragem temporal

5.2.3.2.4 Equivalente em Scikit-Mobility

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

5.2.3.3 Busca por critério KNN

5.2.3.3.1 MoveMiner

Algoritmo 19 Exemplo de *KNN query* em trajetória com MoveMiner

```
geo = LineString([
    Point(-81.37538454278695, 28.544885825638552),
    Point(-81.37535419684313, 28.54517238822342),
    Point(-81.37510763683406, 28.54527901583028),
])
knn_query = moveminer.management.KnnQuery()
result = knn_query.knn_query(traj, 7, geo)
```

Figura 55 – Trajetória Original

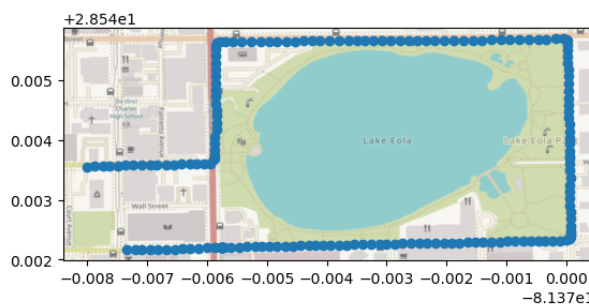
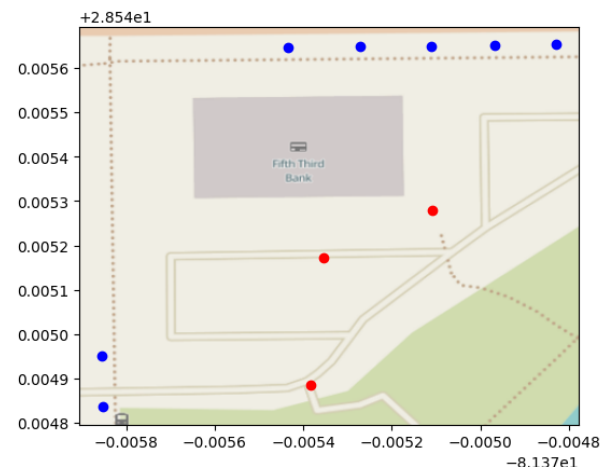


Figura 56 – Resultado de *query KNN*



O método KNN é outro método importante de busca. Ele retorna os k pontos mais próximos de um objeto geométrico. No exemplo do Algoritmo 19, sete pontos foram retornados, como pode ser visto nas Figuras 55 e 56.

5.2.3.3.2 Equivalente em Pymove

Algoritmo 20 Exemplo de *KNN query* em trajetória com Pymove

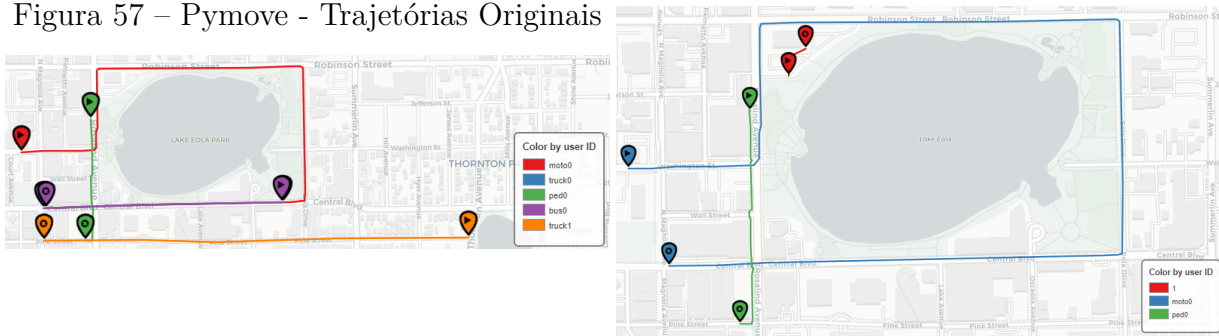
```

knn_query = pd.DataFrame({
    'lat': [28.544885825638552, 28.54517238822342, 28.545279015830
28],
    'lon': [-81.37538454278695, -81.37535419684313, -81.37510763683
406],
    'id': [1, 1, 1]
})
result = query.knn_query(knn_query, move_df, id_='id', k=2, dist
ance='MEDP')

```

Figura 58 – Pymove - Resultado de *query KNN*

Figura 57 – Pymove - Trajetórias Originais



Observa-se que o método de consulta da biblioteca Pymove é consideravelmente diferente do método da MoveMiner. Embora ambas busquem os k vizinhos mais próximos, a Pymove retorna trajetórias inteiras, enquanto a MoveMiner retorna apenas os pontos mais próximos. Portanto, os dois métodos têm objetivos diferentes.

5.2.3.3.3 Equivalente em MovingPandas

A biblioteca MovingPandas não possui um recurso específico para esta tarefa.

5.2.3.3.4 Equivalente em Scikit-Mobility

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

5.2.3.4 Busca por critério de Similaridade

5.2.3.4.1 MoveMiner

Algoritmo 21 Exemplo de busca por similaridade em trajetória com MoveMiner

```
truck0 = moveminer.Trajectory(df=df1, x='x', y='y', t='datetime')
ped0 = moveminer.Trajectory(df=df2, x='x', y='y', t='datetime')
bus0 = moveminer.Trajectory(df=df3, x='x', y='y', t='datetime')
truck1 = moveminer.Trajectory(df=df4, x='x', y='y', t='datetime')
ts = [ped0, bus0, truck1]
similarity_query = moveminer.management.SimilarityQuery()
result = similarity_query.similarity_query(truck0, ts)
```

Figura 59 – Grupo de Trajetórias

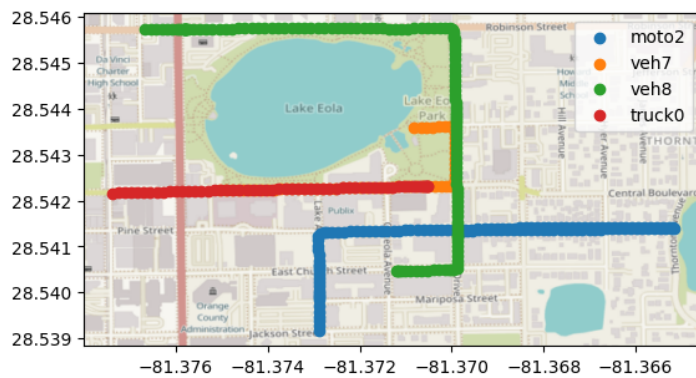
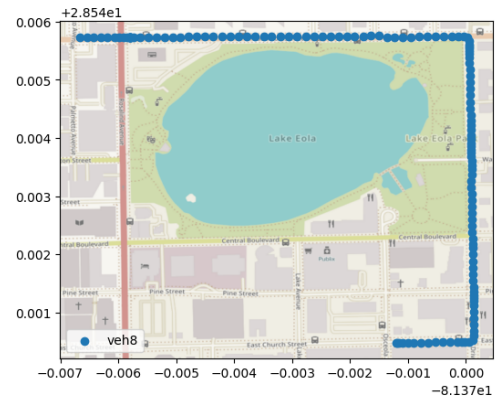


Figura 60 – truck0



Nenhuma das demais bibliotecas apresenta uma solução para busca por similaridade, como mostrado no Capítulo 3.

A biblioteca MoveMiner oferece um método de busca por similaridade. A busca por similaridade retorna a trajetória mais similar à trajetória de entrada. O Algoritmo 21 ilustra como a busca por similaridade funciona. No exemplo, a trajetória mais similar à trajetória *truck0* é a trajetória *veh8*, como pode ser visto nas Figuras 59 e 60.

5.2.3.4.2 Equivalente em Pymove

A biblioteca Pymove não possui um recurso específico para esta tarefa.

5.2.3.4.3 Equivalente em MovingPandas

A biblioteca MovingPandas não possui um recurso específico para esta tarefa.

5.2.3.4.4 Equivalente em Scikit-Mobility

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

5.2.4 Incerteza

5.2.4.0.1 MoveMiner

Algoritmo 22 Exemplo de redução de incerteza em trajetória com MoveMiner

```
smoother = moveminer.uncertainty.Smoother(w=49)
traj_smoothed = smoother.smooth(traj)
```

Figura 61 – Trajetória Original

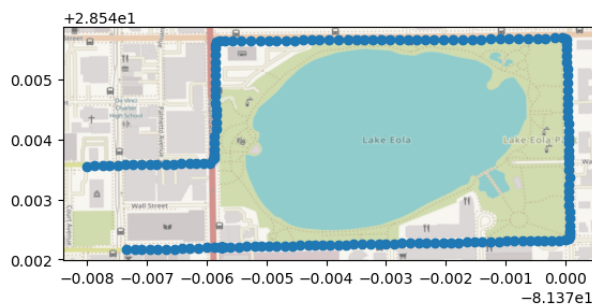
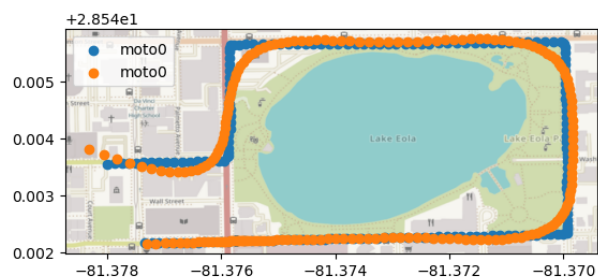


Figura 62 – Trajetória Suavizada



A classe *Smoother*, realiza uma suavização nas curvas da trajetória, o que pode ser muito útil em trajetórias de objetos que não estão restritos a malhas de vias de uma cidade, como embarcações, aeronaves e animais.

5.2.4.0.2 Equivalente em Pymove

A biblioteca Pymove não possui um recurso específico para esta tarefa.

5.2.4.0.3 Equivalente em MovingPandas

Algoritmo 23 Exemplo de redução de incerteza em trajetória com MovingPandas

```
smooth = mpd.KalmanSmootherCV(traj).smooth(process_noise_std=0.1,
measurement_noise_std=10)
```

Figura 63 – MovingPandas - Trajetória original

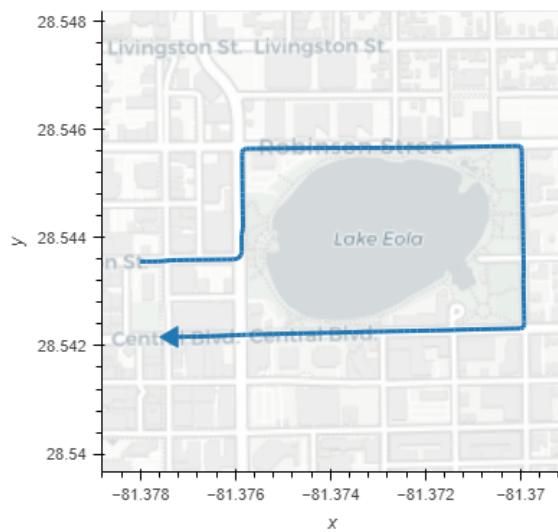
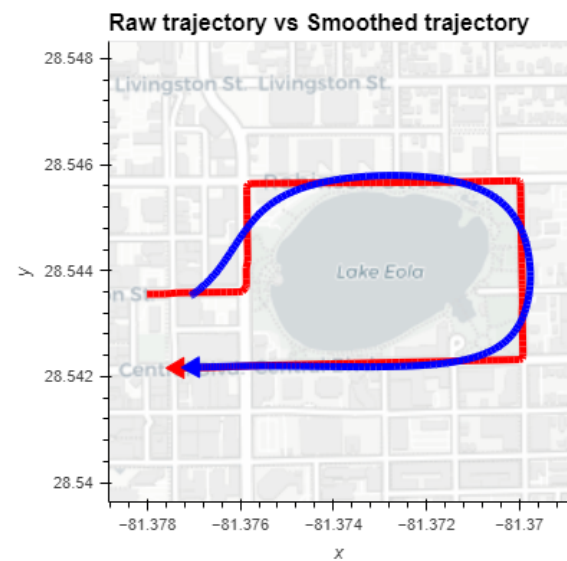


Figura 64 – MovingPandas - Trajetória Suavizada



No Algoritmo 23, um exemplo do método para suavização de trajetória da biblioteca MovingPandas é mostrado, aplicando Kalman Filter (WELCH; BISHOP et al., 1995) com um modelo de velocidade constante. E, nas Figuras 63 e 64 o resultado pode ser visualizado.

5.2.4.0.4 Equivalente em Scikit-Mobility

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

5.2.5 Privacidade

5.2.5.0.1 MoveMiner

Algoritmo 24 Exemplo de *LocationAttack* em trajetória com MoveMiner

```
location_attack = moveminer.LocationAttack(knowledge_length=2)
risk = location_attack.assess_risk(traj)
```

Algoritmo 25 Exemplo de *HomeWorkAttack* em trajetória com MoveMiner

```
home_work_attack = moveminer.HomeWorkAttack(knowledge_length=2)
risk = home_work_attack.assess_risk(traj)
```

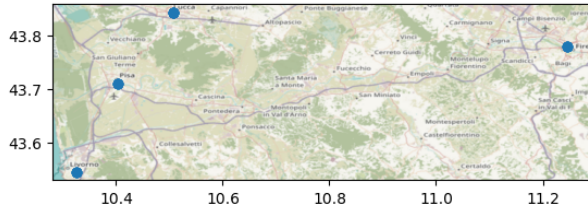
Figura 65 – *Check-ins*

Figura 66

	uid	risk
0	1	0.333333
1	2	0.500000
2	3	0.333333
3	4	0.333333
4	5	0.250000
5	6	0.250000
6	7	0.500000

Figura 67

	uid	risk
0	1	0.25
1	2	0.25
2	3	0.25
3	4	0.25
4	5	1.00
5	6	1.00
6	7	1.00

No exemplo acima, um *dataset* de *check-ins* em diferentes cidades foi utilizado. Cada UID representa a trajetória de um objeto pelas cidades. O resultado do Algoritmo 24 pode ser visto na Figura 66. A figura mostra a probabilidade de identificação de cada objeto, a partir de um conhecimento prévio de dois locais frequentados pelo objeto. De forma semelhante, a Figura 67 mostra a probabilidade de identificação da casa e do local de trabalho do usuário, a partir do conhecimento de dois locais onde o objeto passou. O Algoritmo 25 foi utilizado para calcular essas probabilidades.

5.2.5.0.2 Equivalente em Pymove

A biblioteca Pymove não possui um recurso específico para esta tarefa.

5.2.5.0.3 Equivalente em MovingPandas

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

5.2.5.0.4 Equivalente em Scikit-Mobility

Os métodos implementados na biblioteca MoveMiner possuem resultados idênticos aos da biblioteca Scikit-Mobility. No entanto, a biblioteca Scikit-Mobility oferece uma gama maior de opções para ataques, como listado abaixo:

- HomeWorkAttack;
- LocationAttack;
- LocationFrequencyAttack;
- LocationProbabilityAttack;
- LocationProportionAttack;
- LocationSequenceAttack;
- LocationTimeAttack.

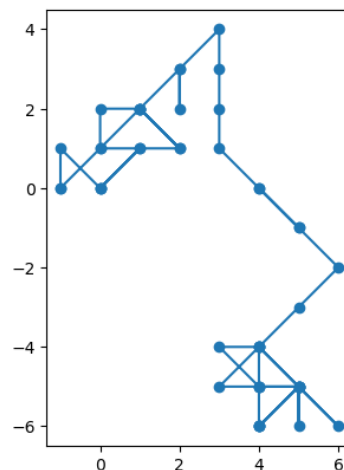
5.2.6 Gerar Mobilidade

5.2.6.0.1 MoveMiner

Algoritmo 26 Exemplo de geração de trajetória com MoveMiner

```
random_walk = moveminer.generators.RandomWalk(50)
traj = random_walk.generator()[0]
```

Figura 68 – Trajetória Sintética



O Algoritmo 26 mostra como gerar uma trajetória com 50 pontos usando o método RandomWalk. A trajetória gerada pode ser vista na Figura 68.

5.2.6.0.2 Equivalente em Pymove

A biblioteca Pymove não possui um recurso específico para esta tarefa.

5.2.6.0.3 Equivalente em MovingPandas

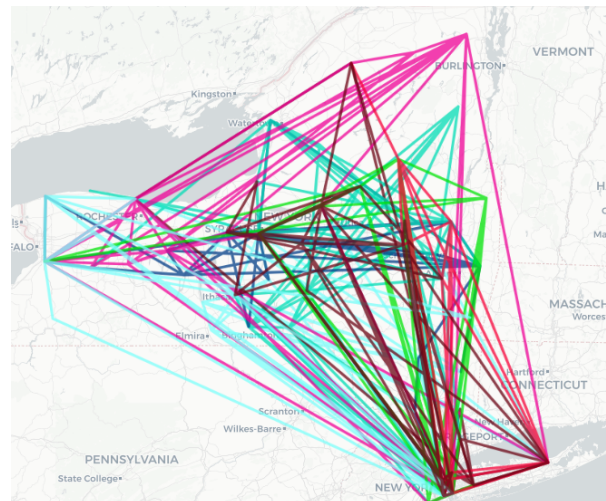
A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

5.2.6.0.4 Equivalente em Scikit-Mobility

Algoritmo 27 Exemplo de geração de trajetória com Scikit-Mobility

```
url = skmob.utils.constants.NY_COUNTIES_2011
tessellation = gpd.read_file(url)
start_time = pd.to_datetime('2019/01/01_08:00:00')
end_time = pd.to_datetime('2019/01/5_08:00:00')
depr = DensityEPR()
tdf = depr.generate(start_time, end_time, tessellation, relevance
_column='population', n_agents=100)
```

Figura 69 – Scikit-Mobility - Trajetória Sintética



O método para geração de mobilidade, mostrado no Algoritmo 27, gera trajetórias através dos centroides de um *tessellation*, que são conjuntos de formas geométricas, geralmente representando regiões. No caso em questão, os dados utilizados são de Nova York, como é visível na Figura 69.

5.2.7 Mineração de Padrões

5.2.7.1 Agrupamento

5.2.7.1.1 MoveMiner

Algoritmo 28 Exemplo de agrupamento de trajetória com MoveMiner (DBSCAN)

```
cluster = moveminer.clustering.DBSCAN(0.018)
traj_clustered = cluster.db_scan(traj)
```

Figura 70 – Trajetórias Originais

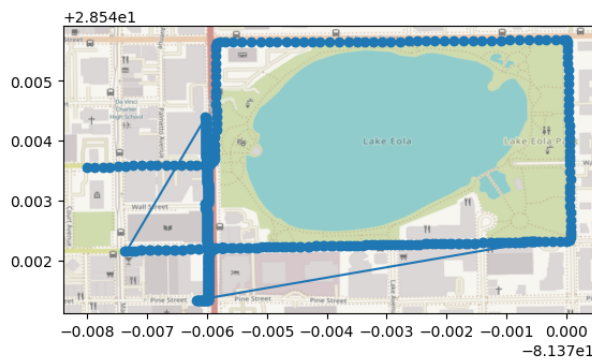
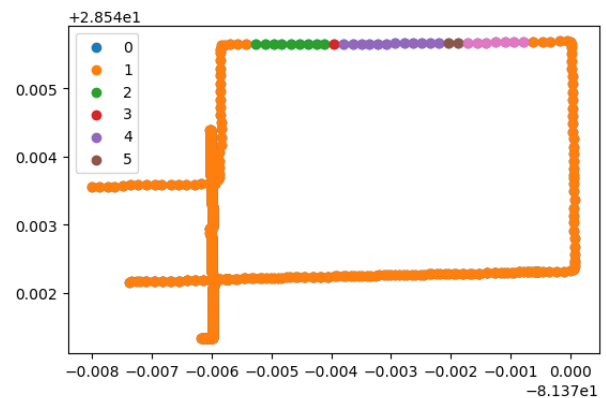


Figura 71 – Trajetória após DBSCAN



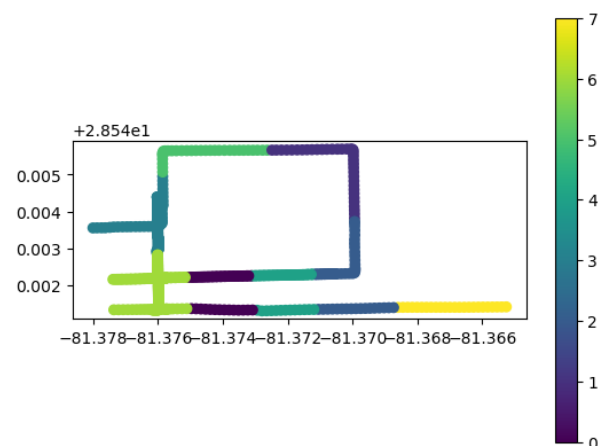
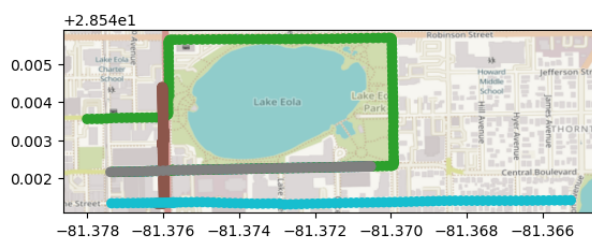
O Algoritmo 28 mostra como usar a biblioteca MoveMiner para agrupar pontos de trajetórias usando o algoritmo DBSCAN (ESTER et al., 1996b). A trajetória agrupada pode ser vista na Figura 71.

Algoritmo 29 Exemplo de agrupamento de trajetória com MoveMiner (K-means)

```
cluster = moveminer.clustering.Kmeans(k=8)
traj_clustered = cluster.k_means(traj)
```

Figura 73 – Trajetória após K-means

Figura 72 – Trajetórias Originais



O Algoritmo 29 mostra como usar a biblioteca MoveMiner para agrupar pontos de trajetórias usando o algoritmo K-means (LLOYD, 1982). A trajetória agrupada pode ser vista na Figura 73.

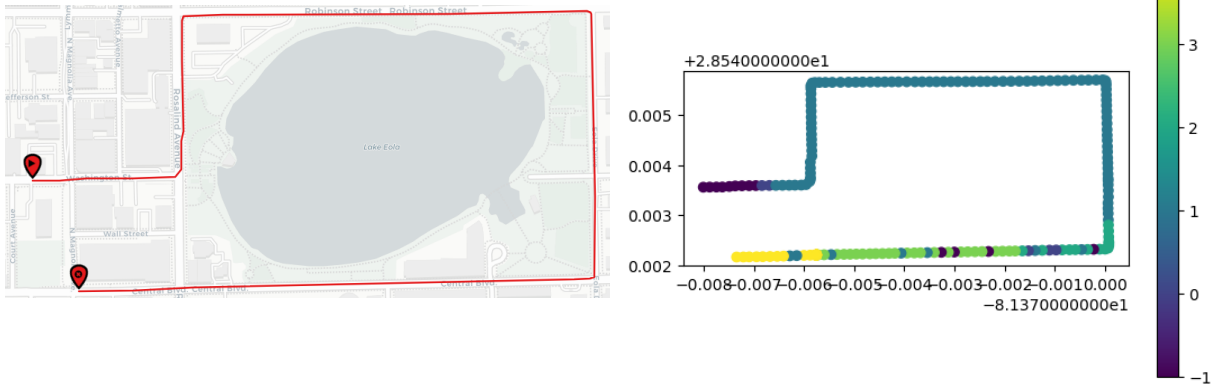
5.2.7.1.2 Equivalente em Pymove

Algoritmo 30 Exemplo de agrupamento de trajetória com Pymove (K-means)

```
clusters = clustering.dbscan_clustering(move_df,
cluster_by='speed', min_sample=2)
```

Figura 75 – Pymove - Trajetória após DBSCAN

Figura 74 – Pymove - Trajetórias Originais

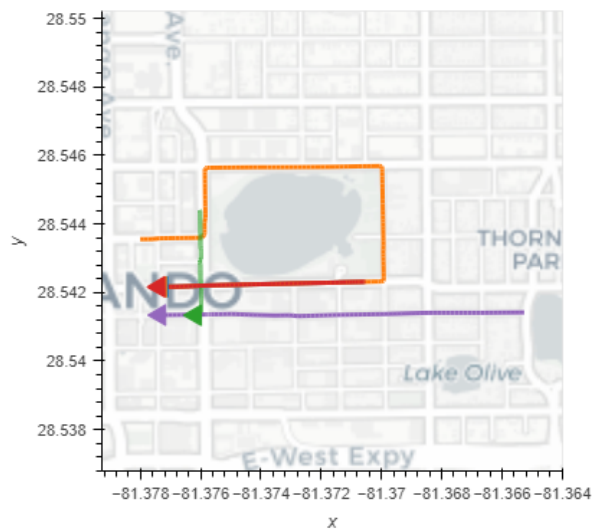


5.2.7.1.3 Equivalente em MovingPandas

Algoritmo 31 Exemplo de agrupamento de trajetória com MovingPandas

```
aggregator = mpd.TrajectoryCollectionAggregator(
    tc,
    max_distance=0.0005,
    min_distance=0.00001,
    min_stop_duration=timedelta(minutes=1),
)
flows = aggregator.get_flows_gdf()
pts = aggregator.get_significant_points_gdf()
clusters = aggregator.get_clusters_gdf()
```

Figura 76 – MovingPandas - Trajetórias Originais



Para esta comparação, o método de agregação da biblioteca MovingPandas está sendo considerado como uma forma de agrupamento de trajetórias. Um exemplo de resultado é exibido nas Figuras 76 e 77. Neste resultado, é possível notar que os pontos próximos foram combinados e suas conexões resumidas em fluxos.

5.2.7.1.4 Equivalente em Scikit-Mobility

Algoritmo 32 Exemplo de agrupamento de trajetória com Scikit-Mobility

```
cluster = clustering.cluster(tdf, cluster_radius_km=0.012,
min_samples=1)
```

Figura 78 – Scikit-Mobility - Trajetórias Originais

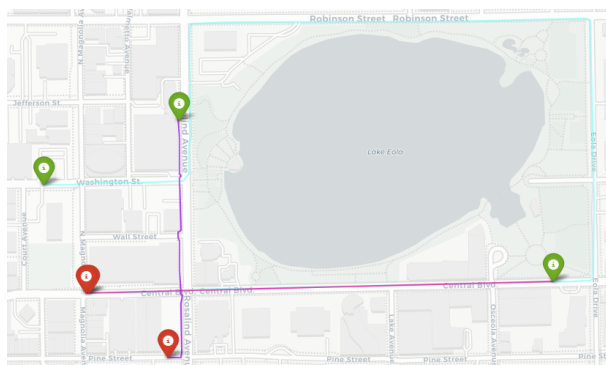


Figura 79 – Scikit-Mobility - Trajetória após Agregação

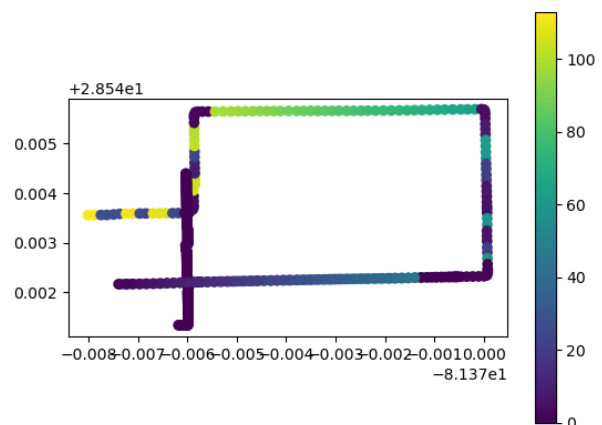


Figura 80 – Trajetórias Originais

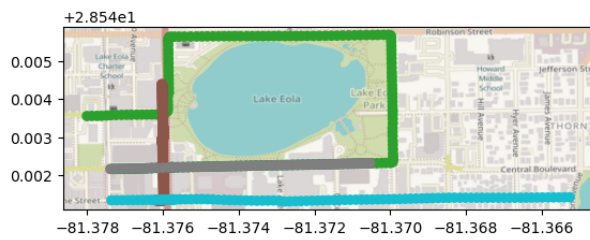
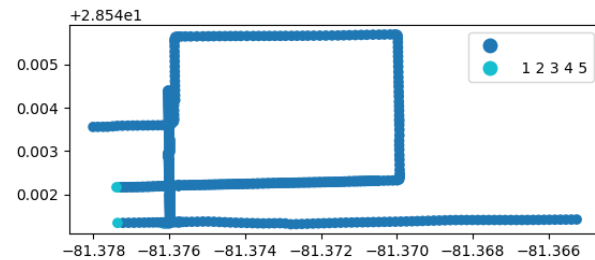


Figura 81 – Encontro de Trajetórias



5.2.7.2 Moving Together

5.2.7.2.1 MoveMiner

Algoritmo 33 Exemplo de mineração de padrões de trajetória com MoveMiner (Encounter)

```
encounter_miner = moveminer.moving_together.Encounter(3, 0.005)
patterns = encounter_miner.encounter(traj)
```

O padrão de encontro, minerado pelo Algoritmo 33, busca encontrar trajetórias que possuem o mesmo destino, opcionalmente ao mesmo tempo. Os resultados estão apresentados nas Figuras 80 e 81. Além disso, é importante destacar que, como mostrado na Tabela 14, nenhuma das outras bibliotecas listadas suporta esta tarefa.

5.2.7.2.2 Equivalente em Pymove

A biblioteca Pymove não possui um recurso específico para esta tarefa.

5.2.7.2.3 Equivalente em MovingPandas

A biblioteca MovingPandas não possui um recurso específico para esta tarefa.

5.2.7.2.4 Equivalente em Scikit-Mobility

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

5.2.7.3 Padrões Sequenciais

5.2.7.3.1 MoveMiner

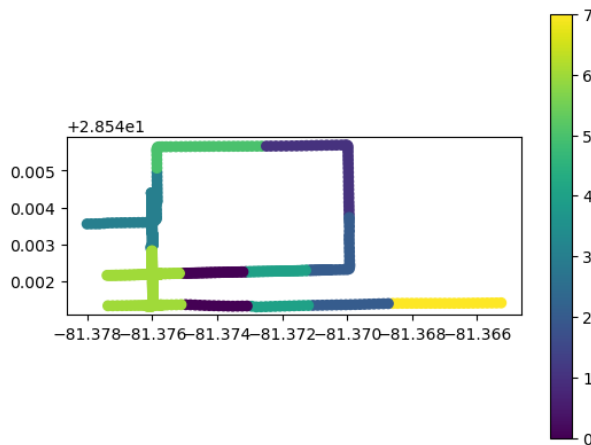
Algoritmo 34 Exemplo de mineração de padrões de trajetória com MoveMiner (PrefixSpan)

```

uids, seqs = mobilib.sequential.cluster_sequences(clustered_traj)
ps_miner = moveminer.sequential.PrefixSpan(3)
patterns = []
for i in ps.prefix_span([], seqs):
    patterns.append(i)
patterns
# ([6], 5)
# ([2], 4)
# ([2, 4], 4)
# ([2, 4, 0], 4)
# ([2, 4, 0, 6], 4)
# ([2, 4, 6], 4)
# ([2, 0], 4)
# ([2, 0, 6], 4)
# ([2, 6], 4)
# ([4], 4)
# ([4, 0], 4)
# ([4, 0, 6], 4)
# ([4, 6], 4)
# ([0], 4)
# ([0, 6], 4)

```

Figura 82 – Conjunto de trajetórias após K-means



O algoritmo PrefixSpan (PEI et al., 2004), aplicado no algoritmo 34, requer que o conjunto de dados tenha sido pré-processado por um algoritmo de clusterização, como

DBSCAN ou K-means, como mostrado na Figura 82. Como resultado, o algoritmo retorna uma lista de tuplas, com dois elementos: o primeiro elemento é uma sequência de *clusters*, e o segundo elemento é um número que representa a frequência do padrão.

5.2.7.3.2 Equivalente em Pymove

A biblioteca Pymove não possui um recurso específico para esta tarefa.

5.2.7.3.3 Equivalente em MovingPandas

A biblioteca MovingPandas não possui um recurso específico para esta tarefa.

5.2.7.3.4 Equivalente em Scikit-Mobility

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

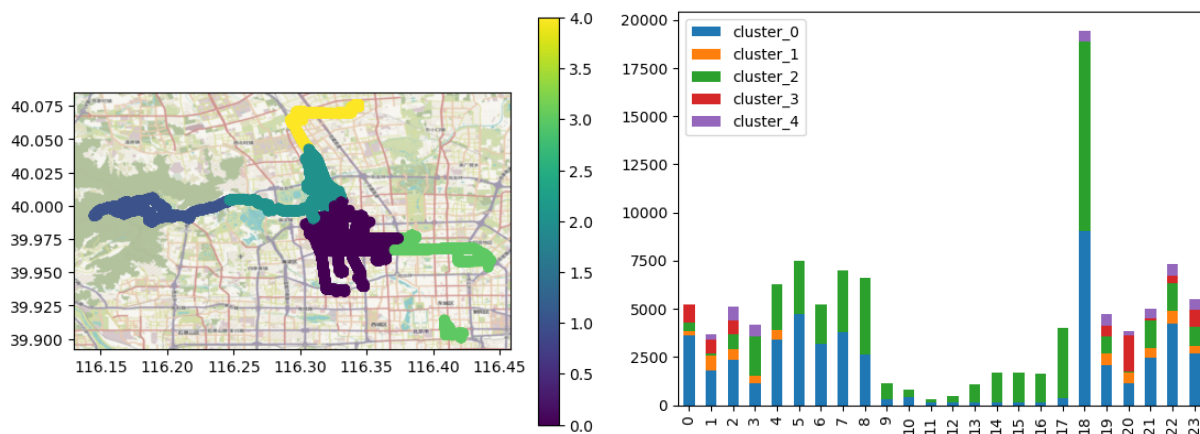
5.2.7.4 Padrões Periódicos

5.2.7.4.1 MoveMiner

Algoritmo 35 Exemplo de mineração de padrões periódicos de trajetória com MoveMiner

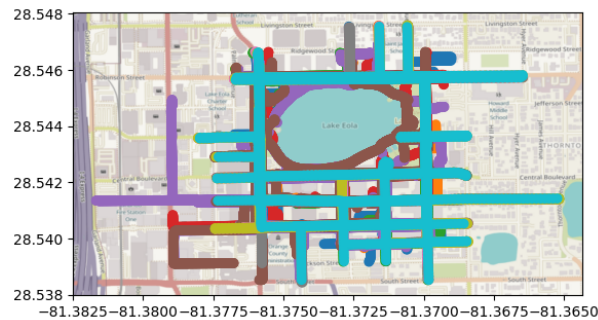
```
periodic_miner = moveminer.periodic.Periodic()
pattern = periodic_miner.periodic(clustered_traj)
```

Figura 83 – Conjunto de trajetórias após K-means



O resultado do Algoritmo 35 é uma visualização da probabilidade de um objeto estar em qualquer um dos *clusters* durante cada hora do dia, exibida na Figura 84. Além disso, neste caso, assim como na mineração de padrões sequenciais, é necessário que o conjunto

Figura 85 – Conjunto de dados



de dados tenha sido pré-processado por um algoritmo de agrupamento, como mostrado na Figura 83. Neste caso específico, os dados foram retirados do *dataset* Geolife (ZHENG et al., 2010).

5.2.7.4.2 Equivalente em Pymove

A biblioteca Pymove não possui um recurso específico para esta tarefa.

5.2.7.4.3 Equivalente em MovingPandas

A biblioteca MovingPandas não possui um recurso específico para esta tarefa.

5.2.7.4.4 Equivalente em Scikit-Mobility

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

5.2.8 Modelagem

5.2.8.1 Classificação

5.2.8.1.1 MoveMiner

O Algoritmo 36 demonstra uma forma direta de treinar e classificar trajetórias usando a representação da biblioteca MoveMiner. Recurso este que, apesar de simples, não é disponibilizado por nenhuma outra biblioteca presente na comparação, como visto na Tabela 14. O conjunto de dados usado para a classificação, estão visíveis na Figura 85.

5.2.8.1.2 Equivalente em Pymove

A biblioteca Pymove não possui um recurso específico para esta tarefa.

Algoritmo 36 Exemplo de classificação de trajetória com MoveMiner

```

# Define features and label
features :[str]= [ 'speed', 'angle' ]
label :str= 'stop'

# divide train and test
train_X, train_y = train.gdf[features], train.gdf[label]
val_X, val_y = val.gdf[features], val.gdf[label]
test_X, test_y = test.gdf[features], test.gdf[label]

# instance and fit model
tdff = moveminer.model.TrajDeepFeedForward(train_X.shape[-1], 1)
class_weight = {False: 0.5, True: 1}
tdff.compile()
tdff.fit(train_X, train_y, val_X, val_y,
        class_weight=class_weight)

# test model
pred_y = tdff.predict(test_X)
test_y = np.asarray(test_y).astype(bool)

# compute metrics
ac = metrics.accuracy_score(test_y, pred_y)
pr = metrics.precision_score(test_y, pred_y)
pe = metrics.recall_score(test_y, pred_y)
f1 = metrics.f1_score(test_y, pred_y)

# plot results
print(f'Accuracy:{ac}') # Accuracy: 0.2181555502046713
print(f'Precision:{pr}') # Precision: 0.03837366728718904
print(f'Recall:{pe}') # Recall: 1.0
print(f'F1:{f1}') # F1: 0.07391109481318503

```

5.2.8.1.3 Equivalente em MovingPandas

A biblioteca MovingPandas não possui um recurso específico para esta tarefa.

5.2.8.1.4 Equivalente em Scikit-Mobility

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

5.2.8.2 Predição**5.2.8.2.1** MoveMiner

Algoritmo 37 Exemplo de predição de trajetória com MoveMiner

```

# Define features and label
features :[str]= [
    'speed',
    'stop',
    'timestep',
    'angle',
    constants.LATITUDE
]
labels :[str]= ['speed']

# get features
train_df = train.gdf[features]
val_df = val.gdf[features]
test_df = test.gdf[features]

# instance a WindowGenerator
input_width = 2
OUT_STEPS = 2
window_generator = WindowGenerator( input_width, OUT_STEPS,
    OUT_STEPS, train_df, val_df, test_df, labels)

# instance, compile and fit model
traj_lstm = moveminer.model.TrajLSTM()
traj_lstm.compile_and_fit(window_generator)

# evaluate model
val_perf = traj_lstm.evaluate(window_generator.val)
performace = traj_lstm.evaluate(window_generator.test, verbose=0)

# predict
window_generator.plot(traj_lstm.lstm, plot_col='speed',
    max_subplots=5)

```

Figura 86 – Divisão da trajetória

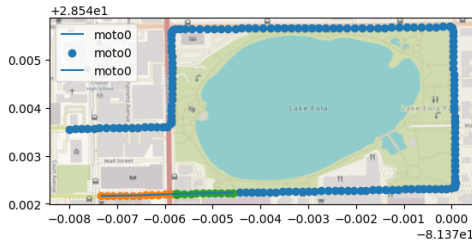
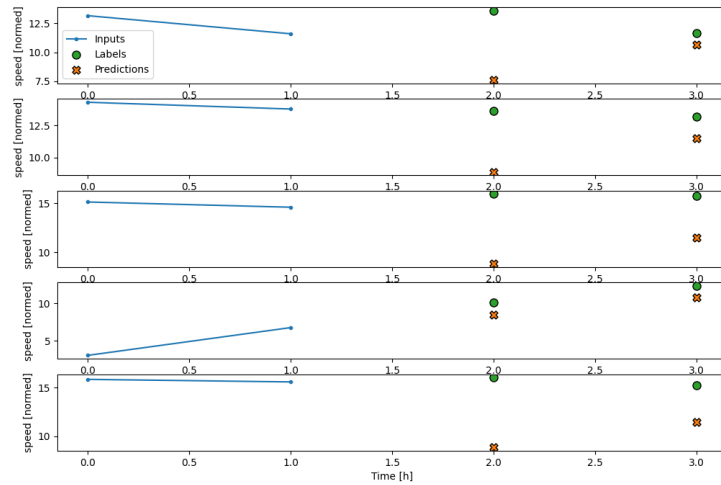


Figura 87 – Predição de velocidade



No Algoritmo 37, é mostrado uma forma de realizar a predição de uma trajetória, através da biblioteca MoveMiner. No caso, a trajetória apresentada na Figura 86, foi dividida em três partes, treino, validação e teste, para então ser usada para treinar o modelo. Para facilitar o treinamento do modelo a classe WindowGenerator é usada para definir a janela de conhecimento anterior e a quantidade de pontos que serão previstos, no caso dois. Por fim, o modelo é testado em cinco exemplos, os resultados estão visíveis na Figura 87.

5.2.8.2.2 Equivalente em Pymove

A biblioteca Pymove não possui um recurso específico para esta tarefa.

5.2.8.2.3 Equivalente em MovingPandas

A biblioteca MovingPandas não possui um recurso específico para esta tarefa.

5.2.8.2.4 Equivalente em Scikit-Mobility

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

5.2.9 Detecção de Anomalias

5.2.9.1 Detectando Trajetórias Atípicas

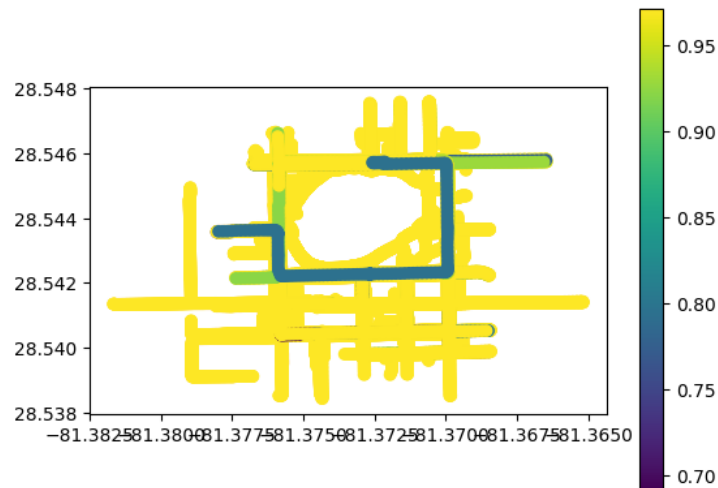
5.2.9.1.1 MoveMiner

O método disponível pela biblioteca MoveMiner para detecção de trajetórias, usa o algoritmo Local Outlier Factor (LOF) (BREUNIG et al., 2000) para calcular a pontuação

Algoritmo 38 Exemplo de detecção de trajetórias atípicas com MoveMiner

```
DOT = DetectOutlierTrajectory()
outlier_traj = DOT.detect_outlier_trajectory(outlier_traj, 0.7)
```

Figura 88 – Trajetórias atípicas



de *outlier* para cada ponto e a pontuação de uma trajetória é dada pela soma dos pontos contidos nela. Exemplificado, no Algoritmo 38 e na Figura 88.

5.2.9.1.2 Equivalente em Pymove

A biblioteca Pymove não possui um recurso específico para esta tarefa.

5.2.9.1.3 Equivalente em MovingPandas

A biblioteca MovingPandas não possui um recurso específico para esta tarefa.

5.2.9.1.4 Equivalente em Scikit-Mobility

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

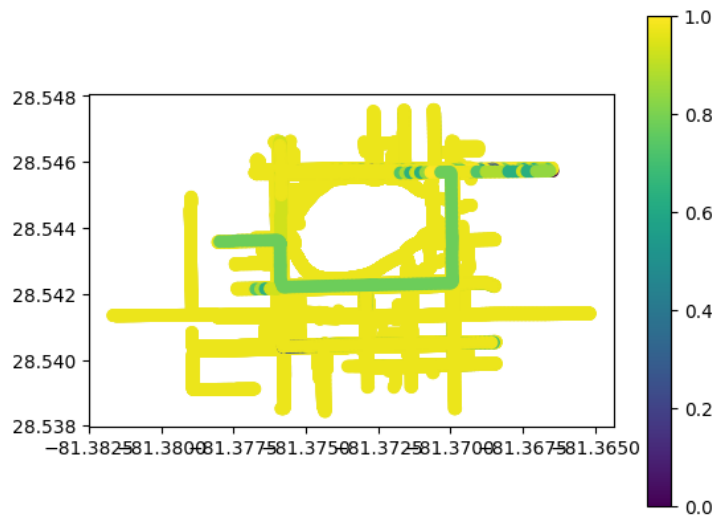
5.2.9.2 Detecção de Eventos Atípicos

5.2.9.2.1 MoveMiner

Algoritmo 39 Exemplo de detecção de eventos atípicos com MoveMiner

```
anomalous_events = AnomalousEvents()
outlier_traj = anomalous_events.anomalous_events(db_traj, outlier_thresho
```

Figura 89 – Eventos Atípicos



Como descrito anteriormente, a biblioteca MoveMiner usa o LOF para computar a pontuação de *outlier* para cada ponto. Através desta pontuação, os eventos atípicos podem ser encontrados, exemplificado no Algoritmo 39 e na Figura 89.

5.2.9.2.2 Equivalente em Pymove

Algoritmo 40 Exemplo de detecção de eventos atípicos com Pymove

```
outliers = semantic.outliers(move_df, jump_coefficient=0.1)
```

Figura 90 – Pymove - Trajetória Original

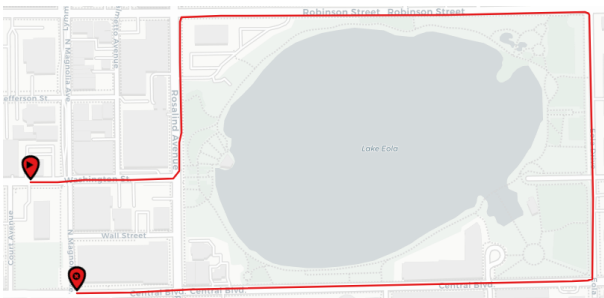
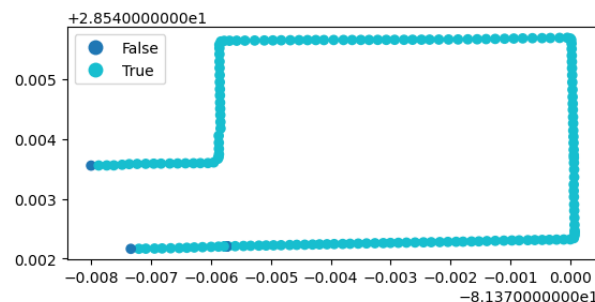


Figura 91 – Pymove - Detecção de Eventos Atípicos



Para detectar eventos atípicos, a biblioteca Pymove utiliza um método para identificar *outliers* através de *thresholds* para métricas de distância. Este método é exemplificado no Algoritmo 40 e nas Figuras 90 e 91.

5.2.9.2.3 Equivalente em MovingPandas

A biblioteca MovingPandas não possui um recurso específico para esta tarefa.

5.2.9.2.4 Equivalente em Scikit-Mobility

A biblioteca Scikit-Mobility não possui um recurso específico para esta tarefa.

5.2.10 Visualização

Por fim, a MoveMiner oferece um módulo de visualização, que foi usado para gerar as imagens de trajetórias deste capítulo. Através do auxílio de bibliotecas como GeoPandas, é possível visualizar trajetórias e formas geométricas por meio de imagens estáticas. Além disso, é possível gerar mapas interativos usando a biblioteca Folium (JOURNOIS et al., 2022), que é integrada à MoveMiner.

As outras bibliotecas comparadas, Pymove, MovingPandas e Scikit-Mobility, possuem seus próprios métodos de visualização, que tiveram seus resultados usados na comparação.

5.3 Resultados

Tabela 16 – Suporte a tarefas pela MoveMiner.

Etapa	Tarefas	MoveMiner
Tipo de entidades	Trajectoria	+
móveis	Filtragem de ruídos	+
Técnicas de pré-processamento	Deteção de pontos de parada	+
	Compressão de trajetórias	+
	Segmentação de trajetória	+
	<i>Map-matching</i>	+
Recursos para gerenciamento de dados	Consultas em Intervalo	+
	Consultas KNN	+
	Similaridade de Trajetórias	+
Métodos para lidar com incerteza	Reduzindo a Incerteza	+
	Análise de Privacidade	+
Gerar mobilidade	Gerar Trajetória	+
Técnicas de minerações de padrões	<i>Moving Together</i>	+
	Agrupamento de Trajetórias	+
	Padrões Sequenciais	+
	Padrões Periódicos	+
Técnicas para predição ou classificação de trajetórias	Predição	+
	Classificação	+
Técnicas para detecção de anomalias	Detectando Trajetórias Atípicas	+
	Deteção de Eventos Atípicos	+
Bibliotecas para visualização de mobilidade	Imagens	+
	Gráficos	+
	Mapas Interativos	+

Este capítulo apresenta a MoveMiner², uma nova biblioteca *open-source* para mineração de dados de trajetória. A MoveMiner oferece uma ampla gama de recursos e funcionalidades, tornando-a uma ferramenta valiosa para pesquisadores e desenvolvedores.

A MoveMiner oferece uma série de vantagens sobre as bibliotecas existentes. Em primeiro lugar, ela suporta todas as tarefas relacionadas à mineração de trajetória, conforme mostrado na Tabela 16.

Em segundo lugar, a MoveMiner fornece soluções para tarefas que nenhuma outra biblioteca no estado da arte fornece, como *Moving Together*, classificação e *map-matching*, conforme mostrado na Tabela 14.

Além disso, este capítulo mostrou uma comparação entre os métodos equivalentes nas três bibliotecas com maior cobertura de tarefas, como exibido na Tabela 14.

² <https://github.com/edgarsoliveira1/MoveMiner>

6 Conclusão

Nesta dissertação, foram apresentados os resultados de uma pesquisa sobre mineração de dados de trajetória. O primeiro capítulo apresentou uma comparação qualitativa entre as bibliotecas Python *open-source* para mineração de dados de trajetória, focando no suporte às tarefas necessárias para extração de conhecimento. O segundo capítulo apresentou uma comparação entre métodos para detecção de pontos de parada, focando em precisão, *recall* e F1. O terceiro capítulo apresentou uma nova biblioteca para mineração de dados de trajetória, desenvolvida com base nos dois primeiros capítulos.

Os resultados da comparação das bibliotecas mostrou que há várias bibliotecas para mineração de dados de trajetória disponíveis, porém nenhuma é capaz de apresentar suporte a todas as tarefas necessárias para a mineração. Além disso, também foi mostrado que algumas tarefas não eram suportadas por nenhuma biblioteca presente no estado da arte. Por fim, uma instância real da metodologia criada para comparação foi implementada e apresentada usando dados reais.

A comparação de métodos mostrou a existência de diversos métodos para detecção de pontos de parada, cada um com suas próprias vantagens e desvantagens. Além disso, um método para geração de *ground-truth* foi apresentado e implementado usando dados do OSM e o software de simulação SUMO, gerando um *dataset* que também foi disponibilizado. Por último, os dados gerados foram usados para realizar a comparação dos métodos, onde os métodos baseados em velocidade se saíram melhor.

A nova biblioteca apresentada na dissertação oferece uma variedade de recursos que a tornam uma ferramenta útil para a mineração de dados de trajetória. A biblioteca suporta todas as tarefas de mineração apresentadas, incluindo a identificação de padrões como *Moving Together*, Classificação de trajetória, *Map-matching*.

A pesquisa apresentada nesta dissertação contribui para o avanço do conhecimento sobre mineração de dados de trajetória. Os resultados da pesquisa podem ser usados para melhorar o desenvolvimento de bibliotecas e algoritmos para mineração de dados de trajetória.

6.1 Trabalhos futuros

A seguir, são apresentadas algumas recomendações específicas para trabalhos futuros:

Avaliar o desempenho da nova biblioteca em diferentes cenários de aplicação: A nova biblioteca pode ser avaliada em diferentes cenários de aplicação, como o monitoramento

de veículos, o rastreamento de pessoas e a análise de tráfego. A avaliação deve considerar fatores como a precisão, a eficiência e a facilidade de uso.

Desenvolver novos métodos de mineração de dados de trajetória que sejam mais precisos e eficientes: Novos métodos de mineração de dados de trajetória podem ser desenvolvidos usando técnicas de aprendizado de máquina, processamento de sinal ou uma combinação de ambas. Os novos métodos devem ser avaliados para garantir que sejam mais precisos e eficientes do que os métodos existentes.

Estender a biblioteca para suportar novas tarefas de mineração: A biblioteca pode ser estendida para suportar novas tarefas de mineração, como a identificação de padrões complexos, a análise de redes sociais e a previsão de comportamentos.

Melhorar a eficiência da biblioteca: A biblioteca pode ser melhorada para aumentar sua eficiência, especialmente para tarefas que envolvem grandes conjuntos de dados.

Tornar a biblioteca mais acessível: A biblioteca pode ser tornada mais acessível para usuários de diferentes níveis de conhecimento, fornecendo documentação e exemplos mais completos.

Além dessas recomendações específicas, também são possíveis trabalhos futuros mais amplos, como:

Realizar uma análise mais aprofundada de outras tarefas assim como a detecção de pontos de parada: A comparação apresentada na dissertação considerou apenas a precisão, o *recall* e o F1. Outros fatores, como a robustez e a sensibilidade, também podem ser considerados.

Acreditamos que esses trabalhos futuros contribuirão para o avanço da pesquisa em mineração de dados de trajetória.

Referências

- ABUL, O.; BONCHI, F.; NANNI, M. Never walk alone: Uncertainty for anonymity in moving objects databases. In: IEEE. *2008 IEEE 24th international conference on data engineering*. [S.l.], 2008. p. 376–385. Citado na página 45.
- ASLAK, U.; ALESSANDRETTI, L. Infostop: scalable stop-location detection in multi-user mobility data. *arXiv preprint arXiv:2003.14370*, 2020. Citado 4 vezes nas páginas 31, 73, 74 e 79.
- BERESFORD, A. R.; STAJANO, F. Location privacy in pervasive computing. *IEEE Pervasive computing*, IEEE, v. 2, n. 1, p. 46–55, 2003. Citado 2 vezes nas páginas 45 e 48.
- BRÁZ, M. C. Implementação de algoritmos para análise de similaridade de trajetória na biblioteca pymove. *Monografia. Universidade Federal do Ceará*, 2020. Citado 4 vezes nas páginas 24, 31, 56 e 75.
- BREUNIG, M. M. et al. Lof: identifying density-based local outliers. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. [S.l.: s.n.], 2000. p. 93–104. Citado na página 114.
- CELES, C.; BOUKERCHE, A.; LOUREIRO, A. A. From mobility traces to knowledge: Design guidance for intelligent vehicular networks. *IEEE Network*, IEEE, v. 34, n. 4, p. 227–233, 2020. Citado na página 35.
- CELES, C. et al. Improving vanet simulation with calibrated vehicular mobility traces. *IEEE Transactions on Mobile Computing*, IEEE, v. 16, n. 12, p. 3376–3389, 2017. Citado na página 45.
- CHOW, C.-Y.; MOKBEL, M. F.; AREF, W. G. Casper* query processing for location services without compromising privacy. *ACM Transactions on Database Systems (TODS)*, ACM New York, NY, USA, v. 34, n. 4, p. 1–48, 2009. Citado na página 45.
- CUSTERS, B. et al. Maximum physically consistent trajectories. *ACM Transactions on Spatial Algorithms and Systems*, ACM New York, NY, v. 7, n. 4, p. 1–33, 2021. Citado 5 vezes nas páginas 31, 73, 75, 79 e 86.
- DENG, D. et al. Spatial-temporal data science of covid-19 data. In: IEEE. *2021 IEEE 15th International Conference on Big Data Science and Engineering (BigDataSE)*. [S.l.], 2021. p. 7–14. Citado na página 29.
- DUARTE, M. M.; SAKR, M. Outlier detection and cleaning in trajectories: A benchmark of existing tools. In: *EDBT/ICDT Workshops*. [S.l.: s.n.], 2023. Citado 5 vezes nas páginas 31, 73, 75, 82 e 86.
- EERLAND, W. et al. Teetool—a probabilistic trajectory analysis tool. *Journal of Open Research Software*, Ubiquity Press, v. 5, n. 1, 2017. Citado 3 vezes nas páginas 24, 31 e 59.

ESTER, M. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: *kdd*. [S.l.: s.n.], 1996. v. 96, n. 34, p. 226–231. Citado na página 74.

ESTER, M. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: *kdd*. [S.l.: s.n.], 1996. v. 96, n. 34, p. 226–231. Citado na página 105.

FREITAS, C. argosfilter: Argos locations filter. *R package version 0.63*, 2012. Citado na página 75.

GIDOFALVI, G.; HUANG, X.; PEDERSEN, T. B. Privacy-preserving data mining on moving object trajectories. In: IEEE. *2007 International Conference on Mobile Data Management*. [S.l.], 2007. p. 60–68. Citado na página 45.

GRASER, A.; DRAGASCHNIG, M. Exploring movement data in notebook environments. In: *IEEE VIS 2020 - MoVis*. [S.l.: s.n.], 2020. Citado 5 vezes nas páginas 24, 31, 54, 75 e 85.

GUDMUNDSSON, J.; KREVELD, M. V. Computing longest duration flocks in trajectory data. In: *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*. [S.l.: s.n.], 2006. p. 35–42. Citado na página 46.

GUDMUNDSSON, J.; KREVELD, M. van; SPECKMANN, B. Efficient detection of motion patterns in spatio-temporal data sets. In: *Proceedings of the 12th annual ACM international workshop on Geographic information systems*. [S.l.: s.n.], 2004. p. 250–257. Citado na página 46.

HAIDRI, S. et al. Ptrail—a python package for parallel trajectory data preprocessing. *SoftwareX*, Elsevier, v. 19, p. 101176, 2022. Citado 4 vezes nas páginas 24, 31, 53 e 75.

HARIHARAN, R.; TOYAMA, K. Project lachesis: parsing and modeling location histories. In: SPRINGER. *International Conference on Geographic Information Science*. [S.l.], 2004. p. 106–124. Citado na página 73.

JEUNG, H.; SHEN, H. T.; ZHOU, X. Convoy queries in spatio-temporal databases. In: IEEE. *2008 IEEE 24th International Conference on Data Engineering*. [S.l.], 2008. p. 1457–1459. Citado na página 46.

JEUNG, H. et al. Discovery of convoys in trajectory databases. *arXiv preprint arXiv:1002.0963*, 2010. Citado na página 46.

JORDAHL, K. et al. geopandas/geopandas: v0. 6.0. *Zenodo*, 2019. Citado na página 54.

JOURNOIS, M. et al. python-visualization/folium: v0. 11.0. *Zenodo*, 2022. Citado na página 117.

KHARRAT, A. et al. Clustering algorithm for network constraint trajectories. In: *Headway in Spatial Data Handling*. [S.l.]: Springer, 2008. p. 631–647. Citado na página 47.

LI, Q. et al. Mining user similarity based on location history. In: *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*. [S.l.: s.n.], 2008. p. 1–10. Citado 4 vezes nas páginas 31, 73, 74 e 79.

- LI, Z. et al. *Swarm: Mining relaxed temporal moving object clusters*. [S.l.], 2010. Citado na página 46.
- LLOYD, S. Least squares quantization in pcm. *IEEE transactions on information theory*, IEEE, v. 28, n. 2, p. 129–137, 1982. Citado na página 105.
- LOPEZ, P. A. et al. Microscopic traffic simulation using sumo. In: IEEE. *2018 21st international conference on intelligent transportation systems (ITSC)*. [S.l.], 2018. p. 2575–2582. Citado na página 77.
- MAIA, P. H.; CAMPELO, C. E.; GRANDE-PB-BRAZIL, C. Mobipy-a python library for analyzing mobility patterns. In: *GeoInfo*. [S.l.: s.n.], 2020. p. 141–152. Citado 3 vezes nas páginas 24, 31 e 55.
- MARTIN, H. et al. Trackintel: An open-source python library for human mobility analysis. *Computers, Environment and Urban Systems*, v. 101, p. 101938, 2023. Citado 3 vezes nas páginas 24, 31 e 51.
- MORENO, R.; OLIVEIRA, C. Avaliacao de desempenho de redes em malha sem fio em cenarios de cidades inteligentes. In: *Anais da X Escola Regional de Computação do Ceará, Maranhão e Piauí*. Porto Alegre, RS, Brasil: SBC, 2022. p. 109–118. ISSN 0000-0000. Disponível em: <<https://sol.sbc.org.br/index.php/ercemapi/article/view/21965>>. Citado na página 29.
- MORRIS, B.; TRIVEDI, M. Learning trajectory patterns by clustering: Experimental studies and comparative evaluation. In: IEEE. *2009 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.], 2009. p. 312–319. Citado na página 48.
- NAZIA, N. et al. Methods used in the spatial and spatiotemporal analysis of covid-19 epidemiology: a systematic review. *International Journal of Environmental Research and Public Health*, MDPI, v. 19, n. 14, p. 8267, 2022. Citado na página 29.
- NERGIZ, M. E.; ATZORI, M.; SAYGIN, Y. Towards trajectory anonymization: a generalization-based approach. In: *Proceedings of the SIGSPATIAL ACM GIS 2008 International Workshop on Security and Privacy in GIS and LBS*. [S.l.: s.n.], 2008. p. 52–61. Citado na página 45.
- NOGUEIRA, T. P. et al. A statistical method for detecting move, stop, and noise: A case study with bus trajectories. *Journal of Information and Data Management*, v. 9, n. 3, p. 214–214, 2018. Citado 5 vezes nas páginas 25, 31, 73, 74 e 79.
- OLIVEIRA, E. et al. Tratamento e análise de dados de mobilidade urbana: Uma metodologia teórica e prática. In: *ERCEMAPI 2023 - Minicursos ()*. [S.l.: s.n.], 2023. Citado 2 vezes nas páginas 32 e 39.
- OLIVEIRA, E. et al. Uma análise comparativa de técnicas de detecção de pontos de parada em ambientes urbanos. In: *ERCEMAPI 2023 - Artigos Completos ()*. [S.l.: s.n.], 2023. Citado na página 32.
- PAPPALARDO, L. et al. scikit-mobility: A python library for the analysis, generation, and risk assessment of mobility data. *Journal of Statistical Software*, v. 103, n. 4, p. 1–38, 2022. Disponível em: <<https://www.jstatsoft.org/index.php/jss/article/view/v103i04>>. Citado 5 vezes nas páginas 24, 31, 56, 75 e 85.

- PEI, J. et al. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on knowledge and data engineering*, IEEE, v. 16, n. 11, p. 1424–1440, 2004. Citado na página 109.
- PLESKOWICZ, M. *Mobilipy-from raw GPS data to mobility data*. [S.l.], 2022. Citado 3 vezes nas páginas 24, 31 e 52.
- REBACK, J. et al. pandas-dev/pandas: Pandas 1.0. 5. *Zenodo*, 2020. Citado na página 50.
- REYES, A. et al. yupi: Generation, tracking and analysis of trajectory data in python. *Environmental Modelling & Software*, Elsevier, v. 163, p. 105679, 2023. Citado 3 vezes nas páginas 24, 31 e 51.
- RINTOUL, M. D. *ML for Trajectories: Bridging the Gap from Computer to Analyst with Tracktable*. [S.l.], 2020. Citado 3 vezes nas páginas 24, 31 e 57.
- RODRIGUES, D. O.; VILLAS, L. Smaframework: Arcabouço para integração de dados urbanos cientes da correlação espaço-temporal. In: SBC. *Anais Estendidos do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.l.], 2019. p. 113–120. Citado 3 vezes nas páginas 24, 31 e 58.
- SAABITH, A. S.; FAREEZ, M.; VINOTHRAJ, T. Python current trend applications-an overview. *International Journal of Advance Engineering and Research Development*, v. 6, n. 10, 2019. Citado na página 30.
- SEIDEL, D. P.; DOUGHERTY, E. R.; GETZ, W. M. Exploratory movement analysis and report building with r package stmmove. *bioRxiv*, Cold Spring Harbor Laboratory, p. 758987, 2019. Citado na página 75.
- SHENK, J. et al. Traja: A python toolbox for animal trajectory analysis. *Journal of Open Source Software*, v. 6, n. 63, p. 3202, 2021. Citado 3 vezes nas páginas 24, 31 e 54.
- SILVA, L. N. et al. Mobvis: A framework for analysis and visualization of mobility traces. In: IEEE. *2022 IEEE Symposium on Computers and Communications (ISCC)*. [S.l.], 2022. p. 1–6. Citado 3 vezes nas páginas 24, 31 e 52.
- SOUZA, F. R. de et al. Mocha: A tool for mobility characterization. In: *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. [S.l.: s.n.], 2018. p. 281–288. Citado 3 vezes nas páginas 24, 31 e 58.
- SPANG, R. et al. Making sense of the noise: integrating multiple analyses for stop and trip classification. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Copernicus Publications Göttingen, Germany, v. 48, p. 435–441, 2022. Citado 2 vezes nas páginas 25 e 75.
- SPANG, R. et al. The staga-dataset: Stop and trip annotated gps and accelerometer data of everyday life. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Copernicus GmbH, v. 48, p. 443–448, 2022. Citado 2 vezes nas páginas 75 e 76.
- STEVENS, J.-L. R.; RUDIGER, P.; BEDNAR, J. A. Holoviews: Building complex visualizations easily for reproducible science. In: SCIPY. *Proceedings of the 14th Python in Science Conference*. [S.l.], 2015. p. 61–69. Citado na página 54.

- TANG, L.-A. et al. A framework of traveling companion discovery on trajectory data streams. *ACM Transactions on Intelligent Systems and Technology (TIST)*, ACM New York, NY, USA, v. 5, n. 1, p. 1–34, 2014. Citado na página 46.
- TANGLU'AN, Z. et al. Discover of traveling companions from streaming trajectories. *ICDE, WashingtonDC, USA*, 2012. Citado na página 46.
- TERROVITIS, M.; MAMOULIS, N. Privacy preservation in the publication of trajectories. In: IEEE. *The Ninth international conference on mobile data management (mdm 2008)*. [S.l.], 2008. p. 65–72. Citado na página 45.
- UBALDI, E. et al. Mobilkit: A python toolkit for urban resilience and disaster risk management analytics using high frequency human mobility data. *arXiv preprint arXiv:2107.14297*, 2021. Citado 3 vezes nas páginas 24, 31 e 53.
- VARGAS-MUNOZ, J. E. et al. Openstreetmap: Challenges and opportunities in machine learning and remote sensing. *IEEE Geoscience and Remote Sensing Magazine*, IEEE, v. 9, n. 1, p. 184–199, 2020. Citado na página 77.
- WELCH, G.; BISHOP, G. et al. An introduction to the kalman filter. Chapel Hill, NC, USA, 1995. Citado 2 vezes nas páginas 78 e 101.
- YUAN, H.; LI, G. A survey of traffic prediction: from spatio-temporal data to intelligent transportation. *Data Science and Engineering*, Springer, v. 6, p. 63–85, 2021. Citado na página 29.
- ZHENG, K. et al. On discovery of gathering patterns from trajectories. In: IEEE. *2013 IEEE 29th international conference on data engineering (ICDE)*. [S.l.], 2013. p. 242–253. Citado na página 46.
- ZHENG, K. et al. Online discovery of gathering patterns over trajectories. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 26, n. 8, p. 1974–1988, 2013. Citado na página 46.
- ZHENG, Y. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, ACM New York, NY, USA, v. 6, n. 3, p. 1–41, 2015. Citado 6 vezes nas páginas 29, 30, 39, 73, 76 e 83.
- ZHENG, Y. et al. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, Citeseer, v. 33, n. 2, p. 32–39, 2010. Citado na página 111.