UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

PEDRO HENRIQUE COSTA MAIA

MOBIPY – BIBLIOTECA PARA ANÁLISE DE PADRÕES DE MOBILIDADE DE USUÁRIOS

CAMPINA GRANDE - PB

2019

PEDRO HENRIQUE COSTA MAIA

MOBIPY – BIBLIOTECA PARA ANÁLISE DE PADRÕES DE MOBILIDADE DE USUÁRIOS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Claudio Elízio Calazans Campelo.

CAMPINA GRANDE - PB

2019

M217m Maia, Pedro Henrique Costa.
Mobipy – Biblioteca para análise de padrões de mobilidade de usuários. / Pedro Henrique Costa Maia. – 2019.

9 f.

Orientador: Prof. Dr. Claudio Elízio Calazans Campelo.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Biblioteca Python. 2. Geoinformática. 3. Mobilidade de usuários - padrões. 4. Padrões de movimentos de usuários. 5. Clusterização. 6. Dataframe. 7. Cálculo de padrão de mobilidade de usuários. I. Campelo, Claudio Elízio Calazans. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

PEDRO HENRIQUE COSTA MAIA

MOBIPY – BIBLIOTECA PARA ANÁLISE DE PADRÕES DE MOBILIDADE DE USUÁRIOS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Claudio Elízio Calazans Campelo
Orientador – UASC/CEEI/UFCG

Professor Dr. Andrey Elísio Monteiro Brito
Examinador – UASC/CEEI/UFCG

Professor Dr. Tiago Lima Massoni
Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 25 de novembro 2019.

CAMPINA GRANDE - PB

Mobipy - Biblioteca para análise de padrões de mobilidade de usuários

Trabalho de Conclusão de Curso

Pedro Henrique Costa Maia (Aluno), Cláudio Campelo (Orientador)*
Departamento de Sistemas e Computação
Universidade Federal de Campina Grande
Campina Grande, Paraíba - Brasil

RESUMO

Com a maior disponibilidade de bases de dados georreferenciados, expandiu-se o interesse em analisá-los em pesquisas que buscam ter uma melhor compreensão dos padrões de mobilidade de pessoas, especialmente em centros urbanos. Esse conhecimento pode contribuir para diferentes campos de pesquisa, além de ser útil para melhorar a infraestrutura das grandes cidades. Diante dessa abrangência de aplicações, criaram-se várias métricas na literatura para inferir padrões de movimento de usuários, entretanto nem todas estão disponíveis para uso em outros trabalhos. Os algoritmos utilizados podem estar indisponíveis, incompatíveis, ou precisando de ajustes, dificultando o reuso. Neste trabalho, apresentamos a Mobipy, uma biblioteca em Python que reúne métricas e funções frequentemente utilizadas para o cálculo de padrões de mobilidade de usuários. Ela foi desenvolvida com foco na usabilidade e na compatibilidade com vários conjuntos de dados, facilitando as tarefas de pesquisa e análise de dados. Para validação, a biblioteca foi testada com dados reais. Esperamos que a Mobipy forneça novas possibilidades para os pesquisadores e desenvolvedores, enriquecendo suas análises e gerando novo conhecimento.

Palavras-chave: *Biblioteca Python, Geoinformática, Padrões de mobilidade de usuários.*

ABSTRACT

With the increased availability of georeferenced databases, the interest in analyzing them in researches that require understanding mobility patterns of people, specially in urban centers, has expanded. This knowledge may help researches in different fields, as well as being useful for improving the infrastructure of large cities. Given this range of applications, several metrics were proposed in the literature to infer patterns of user movement, however, they are not frequently available for use in other studies. The algorithms used may be unavailable, incompatible, or in need of adjustments, making it hard to be reused. In this paper, we present Mobipy, a Python library that brings together metrics and functions frequently used to calculate user mobility patterns. It was developed with a focus on usability and compatibility with multiple data sets, facilitating the tasks of research and data analysis. For validation, the library was tested with real-world data. We hope Mobipy will provide new possibilities for researchers, enriching their analyses and generating new knowledge.

Keywords: *Python Library, Geoinformatics, User Mobility Patterns.*

1 INTRODUÇÃO

O estudo de padrões de mobilidade urbana de usuários é importante para vários tipos de aplicações [1]. O uso desse conhecimento pode fornecer dados úteis para o planejamento urbano de grandes cidades [2], visto que, ao saber como as pessoas se movimentam, estratégias mais eficientes podem ser usadas em relação à infraestrutura, segurança pública, tráfego urbano, entre outros.

A grande quantidade de dados geolocalizados gerados pelo uso massivo de redes sociais e outros aplicativos instalados em *smartphones* possibilitaram pesquisas que analisam desde a dispersão de uma doença [3], até o comportamento humano diante de um desastre natural [4]. A localização, a trajetória e os hábitos dos usuários que podem ser inferidos a partir de dados geo-temporais (dados que contém o local e o momento em que foi coletado) também podem ajudar em sistemas de recomendação [5], em estratégias de desenvolvimento sócio-econômico [6], entre outras áreas.

Diante da abrangência dessas aplicações, várias técnicas foram criadas para extrair padrões de movimento de usuários. Tais técnicas poderiam ser utilizadas em outros trabalhos, o que, segundo Jerônimo et. al. [7], ao serem combinados com dados correlacionados, podem extrair conhecimento relevante. Ou seja, visualizar os dados com base em diferentes métricas pode fornecer ao pesquisador insights importantes sobre os padrões de mobilidade dos usuários.

Entretanto, aplicar essas métricas em um dataset nem sempre é uma tarefa simples. Existem casos em que, apesar da métrica estar implementada e documentada, os dados não são compativeis, adicionando um novo trabalho ao pesquisador de organizar os dados a fim de executar o algoritmo. Além disso, existem casos em que a métrica não está nem implementada, sendo disponibilizado apenas um pseudocódigo ou até mesmo apenas uma descrição textual do algoritmo.

O processo de aplicação de métricas sobre os dados geotemporais deve ser simples e fácil para o pesquisador/desenvolvedor, de forma que este possa dedicar mais tempo à realização de suas análises. Neste contexto, desenvolvemos a *Mobipy: A Python Library for User Mobility Patterns*[1], uma biblioteca em Python que reúne múltiplas funções para cálculo de métricas de padrões de mobilidade. Essas funções foram selecionadas com base em sua relevância e algumas

[1]Link do repositório: github.com/pedrohcm/mobipy

adaptadas para maior abrangência em outras pesquisas. A Mobipy tem como objetivo fornecer um ferramental útil e fácil de usar a pesquisadores e desenvolvedores em geoinformática.

O restante deste artigo está estruturado da seguinte forma. A Seção 2 inclui conceitos básicos a serem utilizados nas seções seguintes. A Seção 3 mostra uma visão geral da biblioteca, com as métricas e funções auxiliares. A Seção 4 inclui demonstrações do sistema em uso com dados reais. Por fim, a Seção 5 mostra as lições aprendidas durante o desenvolvimento, além de apresentar pontos a serem desenvolvidos em trabalhos futuros.

2 CONCEITOS BÁSICOS

Nesta seção serão mostradas algumas definições e tecnologias que foram utilizadas no desenvolvimento da biblioteca.

2.1 Clusterização

Com a maior disponibilidade de dados geoespaciais, aumentou-se também a procura de maneiras de realizar análises exploratórias nesses dados. Para tanto, um dos processos mais utilizados é a clusterização dos dados. Han et al. [8] explicam que tal processo agrupa um conjunto de objetos em classes ou clusters de maneira em que os objetos agrupados tenham alta similaridade entre si. Esse procedimento possibilita a identificação de relações e características que podem existir implicitamente nos bancos de dados espaciais.

Na literatura existem vários métodos de clusterização, sejam eles baseados em wavelets [9], densidade dos dados [10], presença de obstáculos físicos [11], entre outros. Nesse trabalho será utilizado o algoritmo de clusterização DBSCAN (Density Based Spatial Clustering of Applications with Noise) [12] . Este algoritmo foi escolhido por não exigir que o usuário especifique a quantidade de clusters a serem criados, além de precisar apenas de dois parâmetros para funcionar, deixando sua utilização mais simples.

A Figura 1 mostra um exemplo do funcionamento do algoritmo. Nota-se que houve a criação de três clusters, que possuem características espaciais semelhantes e foram agrupados. Existem pontos pretos ao redor dos clusters, chamados de noise, representando dados localizados em regiões de baixa densidade. Essa abordagem será utilizada no cálculo de algumas métricas da biblioteca, que serão descritas na Seção 3.1.

2.2 Dataframe

Realizar operações em um grande dataset pode ser uma tarefa custosa para a máquina. Portanto, é necessário o uso de uma estrutura que consiga filtrar, iterar e acessar dados de maneira eficiente. Para isso, a Mobipy utiliza como estrutura primária de dados o *DataFrame* da biblioteca Pandas [14]. Essa estrutura oferece ferramentas poderosas e flexíveis que ajudam nas funções que serão mostradas na Seção 3.

Essencialmente, o *DataFrame* é uma estrutura de dados bidimensional, composta por linhas e colunas, remetendo a uma planilha. Ele pode ser criado a partir de arquivos, páginas da web ou dados gerados por código. A Figura 2 mostra um exemplo da estrutura de um *DataFrame* com o dataset que utilizamos para realizar os testes da biblioteca, descritos na Seção 4.

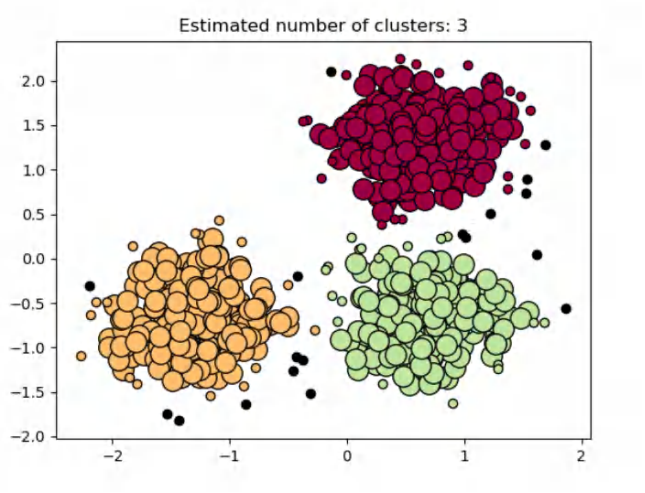

Figura 1: Exemplo de funcionamento do algoritmo DBSCAN, utilizando Scikit-learn [13].

	local_time	latitude	longitude	horizontal_accuracy	speed_accuracy	time	speed	tz	userid
0	1.270654e+09	46.4509	6.86953	87.5226	16.992	1.270661e+09	4.320	-7200.0	6181.0
1	1.270654e+09	46.4509	6.86941	68.2173	11.988	1.270661e+09	6.624	-7200.0	6181.0
2	1.270654e+09	46.4508	6.86946	113.5750	11.988	1.270661e+09	6.624	-7200.0	6181.0
3	1.270654e+09	46.4507	6.86932	56.7157	11.988	1.270661e+09	6.624	-7200.0	6181.0
4	1.270654e+09	46.4506	6.86910	75.9002	9.504	1.270661e+09	3.528	-7200.0	6181.0
...	...	...	...	...	...	...	...	...	...

Figura 2: Descrição do *DataFrame* a ser utilizado na Seção 4.

A partir da criação do dataframe, torna-se possível realizar manipulações nos dados de maneira simples, além de fornecer informações úteis para análises exploratórias a serem realizadas com esses dados.

3 ARQUITETURA E PROJETO DA SOLUÇÃO

A Mobipy é uma biblioteca em Python que visa facilitar o cálculo de métricas e padrões de mobilidade de usuários a partir de dados geo-temporais. Ela tem como foco a facilidade e versatilidade de uso, possibilitando filtrar dados antes da execução dos cálculos, além de prover *insights* úteis sobre os datasets analisados. Inicialmente, há cinco métricas presentes na biblioteca, podendo cada uma ser utilizada para múltiplas aplicações. Devido à flexibilidade da Mobipy, novas funções podem ser adicionadas, bem como as existentes serem expandidas para casos mais específicos.

Inicialmente, foi realizado uma pesquisa em busca de quais métricas viriam a ser implementadas na Mobipy. Dentre elas, algumas estavam originalmente em outras linguagens, necessitando apenas de uma adaptação para Python e suas bibliotecas. Entretanto, outras só haviam um pseudo-código ou apenas uma descrição da métrica, o que levou a um desenvolvimento de uma abordagem que tanto se adequasse à estrutura da Mobipy, como fosse generalizada a fim de ser utilizada em diferentes situações. Tal generalização é necessária devido ao formato de entrada de dados para as funções da Mobipy (*DataFrame*). Como explicado na subseção 2.2, a vantagem

de usar o *DataFrame* é a alta performance ao processar uma grande quantidade de dados, o que é essencial para o cálculo das métricas.

3.1 Métricas

As próximas subseções descrevem os algoritmos de cálculo de métricas de mobilidade implementados na Mobipy e suas funções auxiliares.

3.1.1 Raio de Giro. A métrica do raio de giro é comumente utilizada em dados de *tweets* [15], visto que ela consegue medir quão longe e quão frequentemente um usuário se movimenta [7]. A métrica mede as distâncias acumuladas do centro de massa das trajetórias de um usuário, indicando sua cobertura espacial. O valor resultante é diretamente proporcional à distância das viagens em relação a um centro de massa, podendo fornecer conhecimento sobre seus padrões de movimento.

Para este cálculo, a Mobipy pode receber já o centro de massa ou a própria função que calcula a métrica realiza o processamento necessário. Dessa forma, a função requer apenas um parâmetro: pontos (p) e o centro de massa (p_c), sendo este último opcional. A métrica raio de giro (r_g) é calculada conforme exibido nas Equações 1 e 2.

$$r_g = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - p_c)^2} \tag{1}$$

$$p_c = \frac{1}{n}\sum_{i=1}^{n} p_i \tag{2}$$

3.1.2 Distância Total de Deslocamentos. Outra métrica disponível na Mobipy é a distância total de deslocamentos, que é a soma das distâncias entre os trajetos consecutivos realizados pelo usuário. Essa métrica pode ser útil, por exemplo, para identificar usuários que trabalham como motoristas de aplicativos, taxistas, entre outros.

3.1.3 Centros de Atividades. A partir dos dados do usuário, e aplicando o algoritmo do DBSCAN, mostrado na Seção 2.1, pode-se inferir os centros de atividades. Esses centros podem representar locais em que o usuário permanece com mais frequência, sendo possível processar tais locais por conta da técnica de clusterização aplicada.

3.1.4 Detecção de Local de Residência. O local da casa do usuário pode expressar condições sociais que, por sua vez, podem influenciar como os cidadãos se movem dentro de um centro urbano [7]. Neste caso, a Mobipy considera apenas atividades realizadas no período noturno, e em dias da semana, para minimizar a incidência da falsos-positivos. Dessa forma, a técnica de clusterização do DBSCAN é aplicada novamente aqui para identificar o local de residência do usuário. Embora seja baseado em uma heurística simples, Jerônimo et al. [7] observou que esse método se mostrou útil na identificação de residências e que pode ser reusado em outras pesquisas.

3.1.5 Agrupamento por Proximidade. Agrupar dois datasets pode fornecer conhecimento importante sobre como eles se relacionam. Um exemplo seria relacionar centros de atividades de um usuário com Pontos de Interesse (POI), tais como restaurantes, praças e escolas. Isso pode fornecer dados importantes para empresas sobre os tipos de pessoas que frequentam tais locais, quanto tempo passam, outros locais frequentados por essas pessoas, entre outros.

A função de agrupamento por proximidade recebe dois datasets, A e B, e relaciona todos os itens de B que estiverem mais próximos de itens de A. A saída do algoritmo é uma lista contendo todos os elementos de A e seus respectivos elementos de B mais próximos. Opcionalmente, se o dataset A contiver dados temporais, como *timestamps*, a métrica retornará também os quartis da duração em que cada elemento de A ficou próximo dos elementos de B. Esses valores são usados para gerar um *box-plot* para uma melhor visualização da métrica.

Porém, esse processamento requer um intensivo cálculo de distância entre cada elemento $A_i \in A$ e todos os elementos de B, o que pode ser custoso dependendo da quantidade de elementos em cada dataset. Para isso, o algoritmo realiza um "corte" no dataset B visando aumentar o desempenho ao não precisar calcular a distância para todos os elementos. Esse corte é realizado a partir da criação de um *bounding-box* ao redor do elemento A_i, que filtra os elementos de B para considerar apenas elementos que estiverem dentro dessa estrutura. Esse parâmetro é chamado de *searchTolerance*, que usa como padrão 50m.

Além disso, há também um parâmetro que indica a tolerância, em metros, da distância entre dois elementos de B para serem considerados próximos do mesmo elemento A. Isso ajuda a considerar elementos de B que podem pertencer a mais de um elemento de A.

3.2 Funções Auxiliares para o Cálculo das Métricas

A Mobipy também dispõe de funções auxiliares que têm como entrada os mesmos tipos usados no cálculo das métricas, ou seja, podem ser utilizadas com o mesmo dataset, expandindo seu conjunto de aplicações. Dentre as funções, existem as utilizadas pelas próprias métricas, como o cálculo de distância entre dois pontos, centro de massa, itens de um dataframe mais próximos de um ponto, execução do DBSCAN, entre outros. Essas funções podem ser utilizadas de maneira avulsa, sendo preciso apenas importar o módulo *utils* da Mobipy.

3.3 Funções Auxiliares para os Datasets

A Mobipy inclui algumas funções auxiliares para leitura dos datasets, descritas a seguir.

3.3.1 Identificador de Dados. Diante da vasta quantidade de datasets disponíveis na internet, é fácil se deparar com tipos diferentes de dados, organizados de várias maneiras diferentes. Como a Mobipy deve ser utilizado com um dataset, foi pensado em uma maneira de dar suporte a uma variedade maior destes. Para isso, a biblioteca conta com uma estrutura chamada *DataIdentifier*. Essa estrutura contém os seguintes atributos:

- latitude;
- longitude;
- startTime;
- endTime;
- timestamp;
- itemId.

Tabela 1: Exemplo de funcionamento do filtro de dados com o *Selector*.

	Exemplo	Filtro
(1)	"2019-04-23"e "2019-04-29"	23 à 29 de Abril de 2019
(2)	"000101"	Quarta-feira e Domingo
(3)	"110001110000011110000000"	0-2h, 5-8h, 13-17h

Esses atributos se referem a como os valores correspondentes se encontram no dataset. Por exemplo, o campo latitude pode estar como *lat* ou *geolat*, então o usuário pode criar uma instância do *DataIdentifier* com os nomes de cada um dos campos, para que os algoritmos consigam acessar os dados normalmente, sem precisar alterar a estrutura da base de dados.

3.3.2 Filtro de Dados. A Mobipy também pode filtrar os dados dos datasets antes dos cálculos, sendo útil para o usuário que deseja calcular métricas em apenas um intervalo de tempo, sem precisar criar outro dataset antes de utilizar a biblioteca. O usuário pode filtrar o dataset a partir de:

(1) Intervalo de datas;
(2) Dia da semana;
(3) Intervalo de horário do dia (considerando o dia dividido em 24 intervalos de uma hora).

O filtro foi implementado de maneira que não seja necessário especificar o fuso-horário, sendo compatível diretamente com tipos de dados temporais presentes nos datasets. A entrada (1) é composta de duas *strings* no formato de data, enquanto a (2) e a (3) são configuradas a partir de uma *string* contendo valores entre 0 e 1. O usuário precisa apenas especificar a entrada com valores 1 para considerar e 0 para ignorar. São sete valores a serem configurados para (2) e 24 valores para (3). A Tabela 1 mostra um exemplo de funcionamento de cada um dos filtros de acordo com uma entrada.

Tais parâmetros permitem o cálculo de métricas, por exemplo, usando dados de um certo mês, porém considerando apenas os fins de semana das 19h às 23h. Outro exemplo seria aplicar as métricas apenas para dados coletados à noite, para comparar os resultados com os dados coletados durante o dia, entre outros. Dessa forma, os filtros podem fornecer novas perspectivas sobre a mobilidade dos usuários, aumentando as possibilidades para o pesquisador.

4 SISTEMA EM OPERAÇÃO

Esta seção apresenta exemplos de cenários de execução do sistema para o cálculo das métricas descritas na Seção 3.1. Para estes estudos de caso, utilizamos um *dataset* com dados reais e produzimos artefatos para visualização das saídas das funções.

4.1 Dataset

Os dados a serem utilizados nos testes são originados de uma pesquisa realizada pela Nokia, chamada Mobile Data Challenge (MDC) [16, 17]. Nesta pesquisa, realizada entre 2009 e 2011, na cidade de Lake Geneva, na Suiça, foram utilizados sensores embutidos em *smartphones* com o objetivo de registrar dados contínuos de geolocalização de cerca de 200 pessoas.

4.2 Cenário I

No primeiro cenário, o sistema foi executado para avaliar as métricas de Raio de Giro (Seção 3.1.1), Distância Total de Deslocamentos (Seção 3.1.2) e Centros de Atividades (Seção 3.1.3). Para tanto, foram selecionados os dados de um usuário presente no *dataset* cuja trajetória tenha sido curta, a fim de proporcionar uma melhor visualização dos resultados. A Figura 3 mostra, em verde, os dados geográficos deste usuário.

Figura 3: Dados do usuário I no mapa.

Ao executar a função do raio de giro, não foi especificado o ponto médio. Portanto, o algoritmo fez o cálculo dessa localização antes de seguir sua execução. O ponto médio encontrado foi *(46.501, 6.694)*, identificado na Figura 4 no círculo amarelo no centro do mapa. Esse ponto foi então utilizado como base para identificar a maior distância. Após a execução, a função retornou o raio de giro, que é de aproximadamente $7665m$.

A partir dos dados geográficos, a execução da métrica da distância total calculou que o usuário I teve uma trajetória total de $38{,}6km$ durante o intervalo de tempo de 07 a 20 de Abril de 2010. O cálculo do Centro de Atividades identificou 2 elementos, cada um deles envolvendo os dados nos extremos do mapa, como visto na Figura 4.

Figura 4: Ponto médio e os dois centros de atividades do usuário I resultantes das funções *Raio de Giro* e *Detecção de Centros de Atividades*, respectivamente.

4.3 Cenário II

No segundo cenário experimental, executamos o algoritmo de Detecção de Local de Residência. Para tal, foi selecionado outro usuário com muito mais dados, visto que, desta forma, são gerados mais clusters, para que o algoritmo identifique aquele que corresponde ao local de residência.

Figura 5: Dados geográficos do usuário II.

Como descrito na Seção 3.1.4, o algoritmo seleciona os dados enviados no período noturno e nos dias da semana. Das *49.557* linhas do Dataframe, *17.958* foram consideradas. Destas, foram criados 7 clusters, como exibido no mapa da Figura 6, onde cada cluster é mostrado em uma cor diferente. A partir deles, infere-se que o mais denso, ou seja, o local onde o usuário passa mais tempo à noite em dias da semana, é sua residência. Na Figura 6 é mostrada a residência identificada, juntamente com os outros clusters criados.

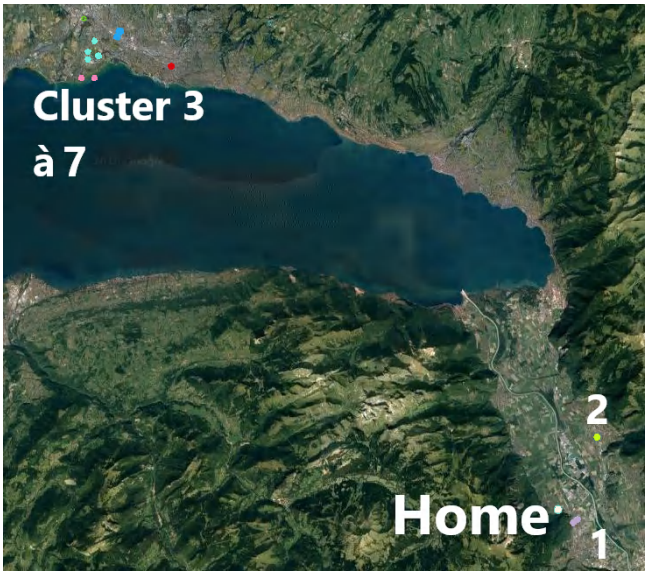

Figura 6: Clusters produzidos pelo algoritmo de *Cálculo de Residência*.

4.4 Cenário III

Para produção do terceiro e último cenário, usamos dois datasets. O primeiro consiste de regiões de parada, que são aglutinações de dados de um usuário. O segundo consiste de POIs obtidos do Google Maps[2]. Ressalta-se que este é apenas um cenário de aplicação da métrica de agrupamento por proximidade (Seção 3.1.5), que, neste caso, pode fornecer informações sobre o perfil dos locais que o usuário mais frequenta. Os datasets foram passados como parâmetro da função, enquanto que a saída da função (incluindo os POIs mais próximos de cada região de parada) foi desenhada no mapa da Figura 7.

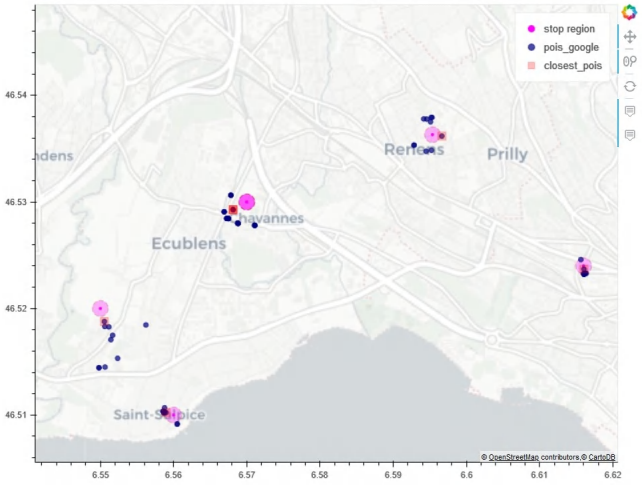

Figura 7: Agrupamento de regiões de parada e pontos de interesse.

Com base na Figura 7, pode-se observar as regiões de parada *(stop region)* (em rosa) com seus POIs mais próximos, limitados em 10 (em azul escuro). O POI mais próximo *(closest pois)* é identificado no mapa pelo quadrado vermelho em sua volta.

5 EXPERIÊNCIA E LIÇÕES APRENDIDAS

Para a implementação da biblioteca, foi escolhida a linguagem Python tanto pela grande abrangência de ferramentas de processamento de dados geoespaciais [18–20] como pela facilidade de codificação, legibilidade e importação em outros projetos em diferentes sistemas. Também houve decisões acerca da estrutura do Mobipy em relação a quais e como cada uma das métricas seriam traduzidas para um algoritmo funcional, bem como este fosse eficiente e generalista a ponto de ser compatível com uma variedade de datasets.

Adicionalmente, foram feitas decisões em prol da performance dos algoritmos, visto que geralmente são usados datasets com milhares de registros, e operações como as feitas no cálculo das métricas tendem a ser custosas em questão de tempo e processamento.

Em relação a aprimoramentos que podem ser realizados na Mobipy em trabalhos futuros, podem ser listados:

[2]Os POIs foram obtidos programaticamente, a partir da API do Google Maps, e então utilizados diretamente para execução da ferramenta, sem armazená-los após a execução, conforme diretrizes especificadas na licença de uso da API

- Implementação de novas métricas ou evoluções das métricas existentes;
- Filtros mais avançados para os datasets;
- Integração com APIs de bancos de dados;
- Mecanismo nativo para visualização dos resultados em plots.

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus por me dar suporte nos momentos difíceis durante a graduação. Aos meus pais, por todo esforço que dedicaram na minha educação. Ao meu parceiro Felipe, pela paciência e por ter me apoiado desde o inicio.

Ao professor Campelo por ter sido um excelente orientador e ter me guiado desde o PIBIC até o final desse trabalho. Ao laboratório LACINA, em especial Tales, que foi de grande ajuda no desenvolvimento da Mobipy.

Por fim, agradeço a todos os meus amigos que me deram forças nessa jornada.

REFERÊNCIAS

[1] Samiul Hasan, Xianyuan Zhan, and Satish V Ukkusuri. Understanding urban human activity and mobility patterns using large-scale location-based data from online social media. In *Proceedings of the 2nd ACM SIGKDD international workshop on urban computing*, page 6. ACM, 2013.

[2] Jing Yuan, Yu Zheng, and Xing Xie. Discovering regions of different functions in a city using human mobility and pois. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 186–194. ACM, 2012.

[3] Gonzalo M Vazquez-Prokopec, Donal Bisanzio, Steven T Stoddard, Valerie Paz-Soldan, Amy C Morrison, John P Elder, Jhon Ramirez-Paredes, Eric S Halsey, Tadeusz J Kochel, Thomas W Scott, et al. Using gps technology to quantify human mobility, dynamic contacts and infectious disease dynamics in a resource-poor urban environment. *PloS one*, 8(4):e58802, 2013.

[4] Xuan Song, Quanshi Zhang, Yoshihide Sekimoto, and Ryosuke Shibasaki. Prediction of human emergency behavior and their mobility following large-scale disaster. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 5–14. ACM, 2014.

[5] Xiangjie Kong, Feng Xia, Jinzhong Wang, Azizur Rahim, and Sajal K Das. Time-location-relationship combined service recommendation based on taxi trajectory data. *IEEE Transactions on Industrial Informatics*, 13(3):1202–1212, 2017.

[6] Luca Pappalardo, Dino Pedreschi, Zbigniew Smoreda, and Fosca Giannotti. Using big data to study the link between human mobility and socio-economic development. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 871–878. IEEE, 2015.

[7] Caio Libânio Melo Jerônimo, Cláudio Elízio Calazans Campelo, and Cláudio de Souza Baptista. Using open data to analyze urban mobility from social networks. *JIDM*, 8(1):83, 2017.

[8] Jiawei Han, Micheline Kamber, and Anthony KH Tung. Spatial clustering methods in data mining. *Geographic data mining and knowledge discovery*, pages 188–217, 2001.

[9] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. Wavecluster: a wavelet-based clustering approach for spatial data in very large databases. *The VLDB Journal—The International Journal on Very Large Data Bases*, 8(3-4):289–304, 2000.

[10] Xin Wang and Howard J Hamilton. Dbrs: a density-based spatial clustering method with random sampling. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 563–575. Springer, 2003.

[11] Osmar R Zaïane and Chi-Hoon Lee. Clustering spatial data when facing physical constraints. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 737–740. IEEE, 2002.

[12] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[14] Wes McKinney. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14, 2011.

[15] Junjun Yin, Yizhao Gao, Zhenhong Du, and Shaowen Wang. Exploring multi-scale spatiotemporal twitter user mobility patterns with a visual-analytics approach. *ISPRS International Journal of Geo-Information*, 5(10):187, 2016.

[16] Juha K Laurila, Daniel Gatica-Perez, Imad Aad, Olivier Bornet, Trinh-Minh-Tri Do, Olivier Dousse, Julien Eberle, Markus Miettinen, et al. The mobile data challenge: Big data for mobile computing research. Technical report, 2012.

[17] Niko Kiukkonen, Jan Blom, Olivier Dousse, Daniel Gatica-Perez, and Juha Laurila. Towards rich mobile phone datasets: Lausanne data collection campaign. *Proc. ICPS, Berlin*, 68, 2010.

[18] Chris Garrard. *Geoprocessing with Python*. Manning Publications Co., 2016.

[19] Anita Graser and Victor Olaya. Processing: A python framework for the seamless integration of geoprocessing tools in qgis. *ISPRS International Journal of Geo-Information*, 4(4):2219–2245, 2015.

[20] Zdena Dobesova. Programming language python for data processing. In *2011 International Conference on Electrical and Control Engineering*, pages 4866–4869. IEEE, 2011.