Mobipy - A Python Library for Analyzing Mobility Patterns

Pedro Henrique Costa Maia[1], **Claudio E. C. Campelo**[1]

[1]Systems and Computing Department
Federal University of Campina Grande (UFCG)
Campina Grande – PB – Brazil

pedro.maia@ccc.ufcg.edu.br, campelo@dsc.ufcg.edu.br

Abstract. *With the increase in the availability of georeferenced databases, interest in analyzing them in research that requires an understanding of people's mobility patterns, especially in urban centers, has increased. Given this range of applications, several metrics have been proposed in the literature to infer patterns of people's movement, however, they are often not available for use in other studies. In this article, we introduce Mobipy, a Python library that brings together metrics and functions frequently used to calculate people's mobility patterns. It was developed with a focus on usability and compatibility with multiple datasets, facilitating research and data analysis tasks. We hope that Mobipy will provide researchers with new possibilities, enriching their analysis and generating new knowledge.*

1. Introduction

The study of urban mobility is important for various types of applications [Hasan et al. 2013]. Methods of identifying and interpreting people's mobility can provide useful knowledge for urban planning in large cities [Yuan et al. 2012], since, by knowing how people move, more efficient strategies can be used in relation to infrastructure, public safety, urban traffic, among others.

The large amount of geolocation data generated by the massive use of social networks and other applications installed on smartphones enabled researches that analyze different social phenomena, from the spread of a disease [Vazquez-Prokopec et al. 2013], to human behavior in the face of a natural disaster [Song et al. 2014]. The location, trajectory and habits of people that can be inferred from geo-temporal data can also help develop recommendation systems [Kong et al. 2017], define strategies for socio-economic development [Pappalardo et al. 2015], generate alternatives to census data [Jerônimo et al. 2017], among other applications.

Given the scope of these applications, several techniques were created to extract movement patterns from spatio-temporal data. However, applying these metrics to a dataset is not always a simple task. There are cases in which, despite the metric being implemented and documented, the data are not compatible, adding a new job to the researcher to organize the data in order to execute the algorithm. In addition, there are cases in which the metric is not even implemented, with only a pseudocode available or even just a textual description of the algorithm.

The process of applying metrics on geotemporal data must be simple and easy for the researcher/developer, so that he can dedicate more time to carry out his analyzes. In

this context, we developed the *Mobipy: A Python Library for Mobility Patterns* [1], that brings together multiple functions for calculating metrics of mobility patterns. These functions were selected based on their relevance and some of them have adapted for greater coverage in other research. Mobipy aims at providing geoinformatics researchers and developers with a useful and easy to use tool.

The rest of this article is structured as follows. Section 2 define some basic concepts used in the following sections. Section 3 shows an overview of the library, with metrics and auxiliary functions. Section 4 includes demos of the system in use with real data. Finally, Section 5 concludes the paper and points to future work.

2. Basic concepts

This section introduces some concepts and technologies that were adopted in the development of the library.

2.1. Clustering

With the increased availability of geospatial data, the demand for ways to conduct exploratory analyzes on these data has also increased. For this, one of the most used processes is data clustering. As discussed by [Han et al. 2001], this process groups a set of objects into classes or clusters so that the grouped objects are highly similar to each other. This procedure makes it possible to identify relationships and characteristics that may exist implicitly in spatial databases.

Many different clustering methods have been proposed in the literature: based on wavelets [Sheikholeslami et al. 2000], data density [Wang and Hamilton 2003], presence of physical obstacles [Zaïane and Lee 2002], among others. In this work, the clustering algorithm DBSCAN (Density Based Spatial Clustering of Applications with Noise) [Ester et al. 1996] will be used. This algorithm was chosen because it does not require the user to specify the number of clusters to be created, and needs only two parameters to work, making its use simpler.

Figure 1 shows an example of how the algorithm works. In this example, three clusters were generated, based on similar spatial characteristics of their containing elements. In addition, it can be seen that there are black dots around the clusters, called noise, representing data located in low density regions. This approach will be used in the calculation of some metrics in the library, which will be described in Section 3.1.

2.2. Dataframe

Performing operations on a large dataset can be a costly task for the machine. Therefore, it is necessary to use a structure that can efficiently filter, iterate and access data. For this purpose, Mobipy uses as its primary data structure the DataFrame of the Pandas library [McKinney 2011]. This structure offers powerful and flexible tools that help manipulate the functions that will be presented in Section 3.

Essentially, DataFrame is a two-dimensional data structure, consisting of rows and columns, referring to a spreadsheet. It can be created from files, web pages or code-generated data. Table 1 shows an example of the structure of a *DataFrame* with the dataset we used to perform the library tests, described in Section 4.

[1] Repository link: github.com/pedrohcm/mobipy

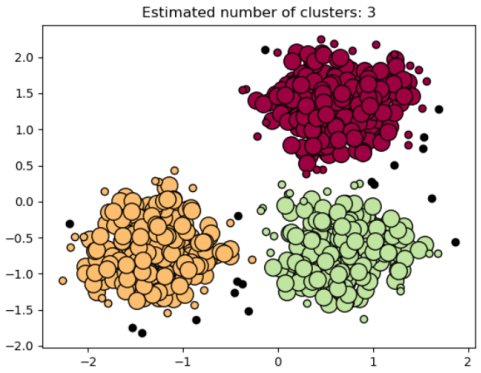

Figure 1. Example of operation of the DBSCAN algorithm, using Scikit-learn.

Table 1. Description of *DataFrame* used in Section 4.

	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	11.998	1.270661e+09	6.624	-7200.0	6181.0
2	1.270654e+09	46.4508	6.86946	113.5750	11.998	1.270661e+09	6.624	-7200.0	6181.0
3	1.270654e+09	46.4507	6.86932	56.7157	11.998	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

From the creation of the dataframe, it becomes possible to perform manipulations on the data in a simple way, in addition to providing useful information for exploratory analyzes to be carried out with this data.

3. Solution Architecture and Design

Mobipy is a code library writen in Python programming language that aims at facilitating the calculation of metrics and patterns of mobility of from geo-temporal data. Common input data are those produced by smartphones and from geotagged posts from social network, although other data in the same format can also be used. It focuses on ease and versatility of use, making it possible to filter data before performing the calculations, in addition to providing *insights* on the analyzed datasets. Currently, there are five metrics in the library, each of which can be used for multiple applications. Due to Mobipy's flexibility, new functions can be added, as well as existing ones expanded to more specific cases.

Initially, a literature review was carried out to identify candidate metrics to be implemented in Mobipy. Among them, some were originally in other languages, needing certain adaptation for Python and its libraries. However, others only had a pseudo-code or just a textual description, which led to the development of an approach that both suited the structure of Mobipy and was generalized in order to be used in different situations. Such generalization is necessary due to the data entry format for Mobipy's functions (*DataFrame*). As explained in the subsection 2.2, the advantage of using *DataFrame* is the high performance when processing a large amount of data, which is essential for calculating metrics.

3.1. Metrics

The next subsections describe the mobility metrics calculation algorithms implemented in Mobipy and its auxiliary functions.

3.1.1. Radius of Gyration

The metric Radious of Gyration (a.k.a. Radious of Inertia) is commonly used in data from *tweets* [Yin et al. 2016], as it can measure how far and how often a user moves [Jerônimo et al. 2017]. The metric measures the accumulated distances from the center of mass of a user's trajectories, indicating their spatial coverage. The resulting value is directly proportional to the travel distance in relation to a center of mass, and can provide knowledge about its movement patterns.

For this computation, Mobipy can receive as an input the center of mass (previously calculated by the user) or, alternatively, it can be calculated by the library. Thus, the function only requires one parameter: points (p), since the center of mass (p_c), is optional. The turning radius (r_g) metric is calculated as shown in Equations 1 and 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. Total Distance Displacement

Another metric available at Mobipy is the total travel distance, which is the sum of the distances between the consecutive routes performed by the user. This metric can be useful, for example, to identify users who work as application drivers, taxi drivers, among others.

3.1.3. Activity Centers

From the user data, and applying the DBSCAN algorithm, shown in Section 2.1, it is possible to infer the activity centers. These centers can represent places where the user remains most frequently, being possible to process such places due to the clustering technique applied.

3.1.4. Home Location Detection

The location of the user's home can express social conditions that, in turn, can influence how citizens move within an urban center [Jerônimo et al. 2017]. In this case, Mobipy considers only activities carried out at night, and on weekdays, to minimize the incidence of false positives. Thus, the DBSCAN clustering technique is applied again here to identify the user's place of residence. Although based on a simple heuristic,

[Jerônimo et al. 2017] noted that this method proved to be useful in identifying residences and that it can be reused in other research.

3.1.5. Group by Closeness

Grouping two datasets together can provide important insight into how they relate. An example would be to link a user's activity centers with Points of Interest (POI), such as restaurants, squares and schools. This can provide important data for companies about the types of people who visit such places, how much time they spend, other places frequented by these people, among others.

The proximity grouping function takes two datasets, A and B, and lists all items in B that are closest to items in A. The output of the algorithm is a list containing all the elements of A and their respective elements of B. Optionally, if the A dataset contains temporal data, such as *timestamps*, the metric will also return the quartiles of the duration that each element of A was close to the elements of B. These values are used to generate a *box-plot* for a better view of the metric.

However, this processing requires an intensive calculation of the distance between each element $A_i \in A$ and all elements of B, which can be costly depending the number of elements in each dataset. For this, the algorithm performs a "cut" in the dataset B in order to increase performance by not having to calculate the distance for all elements. This cut is performed by creating a *bounding-box* around the A_i element, which filters the elements from B to consider only elements that are within that structure. This parameter is called *searchTolerance*, which defaults to 50m.

In addition, there is also a parameter that indicates the tolerance, in meters, of the distance between two elements of B to be considered close to the same element A. This helps to consider elements of B that can belong to more than one element of A.

3.2. Helper Functions for Calculating Metrics

Mobipy also has auxiliary functions that have as input the same types used in the calculation of the metrics, that is, they can be used with the same dataset, expanding its set of applications. Among the functions, there are those used by the metrics themselves, such as the calculation of distance between two points, center of mass, dataframe items closer to a point, execution of DBSCAN, among others. These functions can be used separately, just importing the Mobipy emph utils module.

3.3. Helper Functions for Datasets

Mobipy includes some auxiliary functions for reading the datasets, described below.

3.3.1. Data Identifier

Given the vast amount of datasets available on the internet, it is easy to come across different types of data, organized in many different ways. Since Mobipy must be used with a dataset, it was thought of as a way to support a greater variety of these. For this, the library has a structure called *DataIdentifier*. This structure contains the following attributes:

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

These attributes refer to how the corresponding values are found in the dataset. For example, the latitude field can be like *lat* or *geolat*, so the user can create an instance of *DataIdentifier* with the names of each of the fields, so that the algorithms can access the data normally, without having to change the database structure.

3.3.2. Data Filter

Mobipy can also filter data from datasets before calculations, being useful for the user who wants to calculate metrics in just one time interval, without having to create another dataset before using the library. The user can filter the dataset from:

1. Date range;
2. Weekday (monday to sunday);
3. Time interval of the day (considering the day divided into 24 hourly intervals).

The filter was implemented in such a way that it is not necessary to specify the time zone, being directly compatible with types of temporal data present in the datasets. The input (1) is composed of two *strings* in date format, while (2) and (3) are configured from a *string* containing binary values (0 or 1), where 1 means the item is of interest and 0 indicates the item must be ignored. There are seven values to be set for (2) and 24 values for (3). Table 2 shows an example of how each filter works according to an input.

Table 2. Example of data filter operation with *Selector*.

Example	Filter
“2019-04-23” “2019-04-29”	April 23 to 29, 2019
“1001000”	Wednesday and Sunday
“110001110000011110000000”	0-2h, 5-8h, 13-17h

Such parameters allow the calculation of metrics, for example, using data from a certain month, but considering only weekends from 19h to 23h. Another example would be to apply the metrics only to data collected at night, to compare the results with data collected during the day, among others. In this way, filters can provide new perspectives on user mobility, increasing the possibilities for the researcher.

4. Examples of Usage

This section presents examples of system execution scenarios for calculating the metrics described in Section 3.1. For these case studies, we use a *dataset* with real data and produce artifacts to visualize the outputs of the functions.

4.1. Dataset

The data to be used in the tests come from a survey carried out by Nokia, called Mobile Data Challenge (MDC) [Laurila et al. 2012, Kiukkonen et al. 2010]. In this survey, carried out between 2009 and 2011, in the city of Lake Geneva, Switzerland, sensors embedded in smartphones were used in order to record continuous geolocation data of about 200 people.

4.2. Scenario I

In the first scenario, the system was run to evaluate the Radius of Gyration metric (Section 3.1.1), Total Distance Displacement (Section 3.1.2) and Activity Centers (Section 3.1.3). For that, data from a user present in the dataset whose trajectory has been short was selected, in order to provide a better visualization of the results. Figure 2 shows, in green, the geographic data of this user.

Figure 2. User 1 data on the map.

When executing the Radius of Gyration function, the midpoint was not specified. Therefore, the algorithm did the calculation of this location before following its execution. The midpoint found was *(46.501, 6.694)*, identified in Figure 3 in the yellow circle in the center of the map. This point was then used as a basis to identify the greatest distance. After execution, the function returned the Radius of Gyration, which is approximately 7665 *m*.

From the geographic data, the execution of the total distance displacement calculated that user 1 had a total trajectory of 38.6 *km* during the time interval from 07 to 20 April 2010. The calculation of the Activities identified 2 elements, each involving data at the ends of the map, as seen in Figure 3.

4.3. Scenario II

In the second experimental scenario, we performed the Home Location Detection algorithm. For this, another user was selected with much more data, since, in this way, more clusters are generated, so that the algorithm identifies the one that corresponds to the home location.

As described in Section 3.1.4, the algorithm selects the data sent at night and on the days of the week. Out of the *49.557* Dataframe lines, *17.958* were considered. Then,

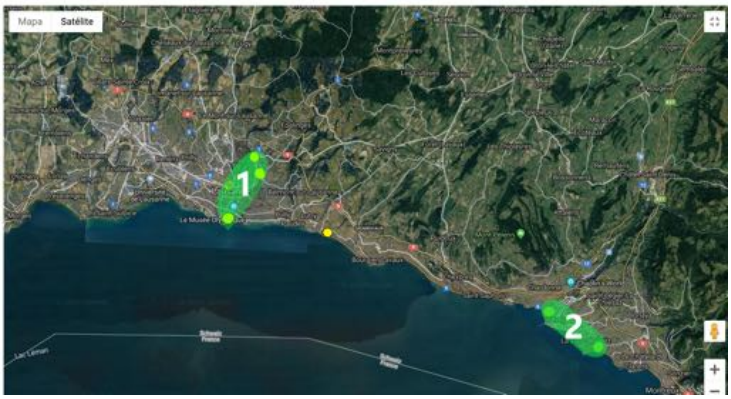

Figure 3. Midpoint and the two user 1 activity centers resulting from the *Radius of Gyration* and *Activity Center Detection* functions, respectively.

Figure 4. Geographic data of user 2.

from those remaining ones, 7 clusters were created, as shown in the map of Figure 5, where each cluster is shown in a different color. From them, it appears that the densest regions (i.e., the place where the user spends the most time at night on weekdays) are their home. In Figure 5 the identified home location is shown, together with the other clusters created.

4.4. Scenario III

For the third and final scenario, we used two datasets. The first consists of stop regions, which are aggregations of a user's data. The second consists of POIs obtained from Google Maps [2]. It is noteworthy that this is only a scenario of application of the proximity grouping metric (Section 3.1.5), which, in this case, can provide information about the profile of the places that the user most frequently visits. The datasets were passed as a function parameter, while the function output (including the POIs closest to each stop

[2]POIs were obtained programmatically, from the Google Maps API, and then used directly to run the tool, without storing them after execution, according to guidelines specified in the API license

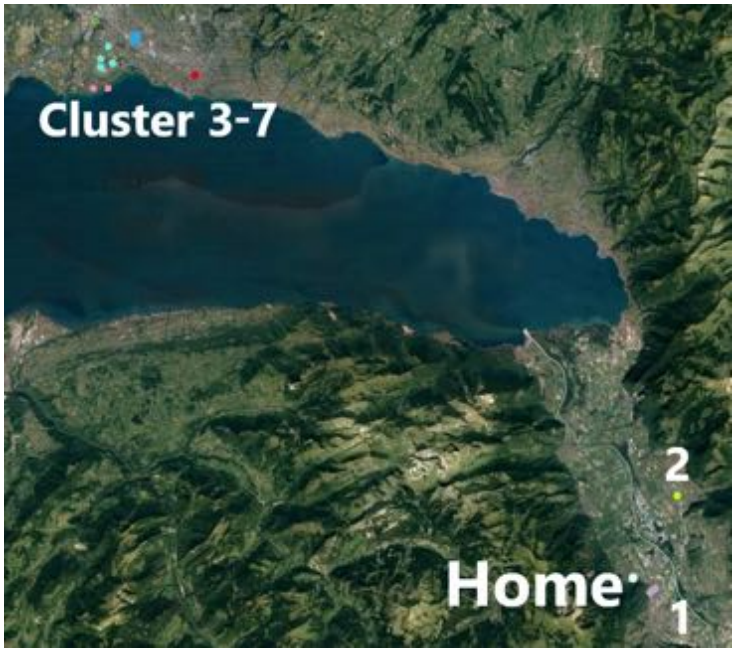

Figure 5. Clusters produced by the *Home Location Detection* algorithm.

region) was drawn on the map of Figure 6.

Based on Figure 6, it is possible to observe the stop regions *(stop region)* (in pink) with their closest POIs, limited to 10 (in dark blue). The nearest POI *(closest because)* is identified on the map by the red square around it.

5. Final Remarks

For the implementation of the library, the Python language was chosen due to the wide range of geospatial data processing tools available [Garrard 2016, Graser and Olaya 2015, Dobesova 2011] as well as to the ease of coding, readability and import in other projects in different systems. There were also decisions about the structure of Mobipy in relation to which and how each of the metrics would be translated into a functional algorithm, as well as that it would be efficient and general to the point of being compatible with a variety of datasets. Furthermore, decisions were made in favor of the performance of the algorithms, since datasets with thousands of records are generally used, and operations such as those performed in the calculation of the metrics tend to be costly in terms of time and processing.

Regarding improvements that can be made at Mobipy in future work, the following can be listed:

(i) implementation of additional metrics or evolutions of existing ones;
(ii) development of more advanced dataset filters;
(iii) integration with database APIs;
(iv) native tool for viewing plots results.

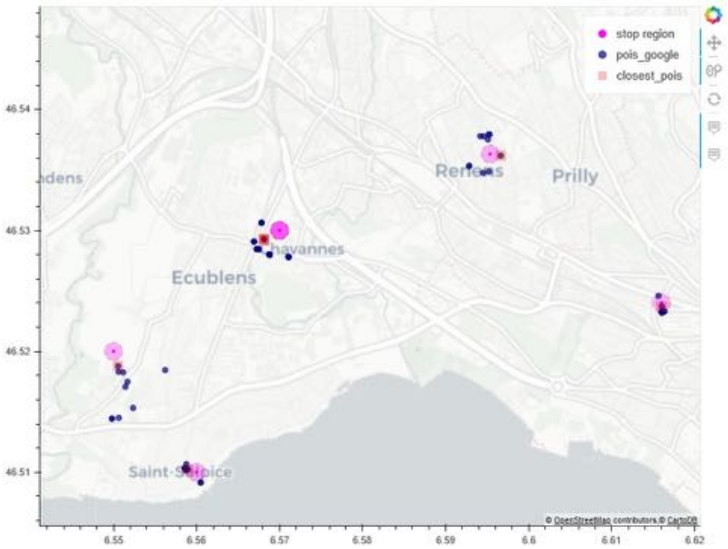

Figure 6. Grouping of stop regions and points of interest.

References

Dobesova, Z. (2011). Programming language python for data processing. In *2011 International Conference on Electrical and Control Engineering*, pages 4866–4869. IEEE.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

Garrard, C. (2016). *Geoprocessing with Python*. Manning Publications Co.

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

Han, J., Kamber, M., and Tung, A. K. (2001). Spatial clustering methods in data mining. *Geographic data mining and knowledge discovery*, pages 188–217.

Hasan, S., Zhan, X., and Ukkusuri, S. V. (2013). 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.

Jerônimo, C. L. M., Campelo, C. E. C., and de Souza Baptista, C. (2017). Using open data to analyze urban mobility from social networks. *JIDM*, 8(1):83.

Kiukkonen, N., Blom, J., Dousse, O., Gatica-Perez, D., and Laurila, J. (2010). Towards rich mobile phone datasets: Lausanne data collection campaign. *Proc. ICPS, Berlin*, 68.

Kong, X., Xia, F., Wang, J., Rahim, A., and Das, S. K. (2017). Time-location-relationship combined service recommendation based on taxi trajectory data. *IEEE Transactions on Industrial Informatics*, 13(3):1202–1212.

Laurila, J. K., Gatica-Perez, D., Aad, I., Bornet, O., Do, T.-M.-T., Dousse, O., Eberle, J., Miettinen, M., et al. (2012). The mobile data challenge: Big data for mobile computing research. Technical report.

McKinney, W. (2011). pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14.

Pappalardo, L., Pedreschi, D., Smoreda, Z., and Giannotti, F. (2015). 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.

Sheikholeslami, G., Chatterjee, S., and Zhang, A. (2000). 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.

Song, X., Zhang, Q., Sekimoto, Y., and Shibasaki, R. (2014). 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.

Vazquez-Prokopec, G. M., Bisanzio, D., Stoddard, S. T., Paz-Soldan, V., Morrison, A. C., Elder, J. P., Ramirez-Paredes, J., Halsey, E. S., Kochel, T. J., Scott, T. W., et al. (2013). Using gps technology to quantify human mobility, dynamic contacts and infectious disease dynamics in a resource-poor urban environment. *PloS one*, 8(4):e58802.

Wang, X. and Hamilton, H. J. (2003). Dbrs: a density-based spatial clustering method with random sampling. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 563–575. Springer.

Yin, J., Gao, Y., Du, Z., and Wang, S. (2016). Exploring multi-scale spatiotemporal twitter user mobility patterns with a visual-analytics approach. *ISPRS International Journal of Geo-Information*, 5(10):187.

Yuan, J., Zheng, Y., and Xie, X. (2012). 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.

Zaïane, O. R. and Lee, C.-H. (2002). Clustering spatial data when facing physical constraints. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 737–740. IEEE.