A Trajectory Cleaning Framework for Trajectory Clustering

Agzam Idrissov
Department of Computing Science
University of Alberta, Canada
idrissov@ualberta.ca

Mario A. Nascimento
Department of Computing Science
University of Alberta, Canada
mario.nascimento@ualberta.ca

ABSTRACT

Trajectory clustering is the process of grouping similar trajectories according to a similarity distance. Several methods for trajectory clustering have been proposed. However, most of these methods require initial data preprocessing which are seldom explained. To that end, this paper proposes a framework for cleaning trajectory data, which consists of three steps: stop detection, missing segment interpolation and inaccuracy removal. Using Nokia's Mobile Data Challenge dataset as a sample application for our framework and objective metric, we show that our data preprocessing approach improves the quality of trajectory clustering.

1. INTRODUCTION

Clustering is a process of organizing similar objects into groups. Initially, most algorithms in this field were designed for point data (e.g. DBSCAN [5] and OPTICS [3]). However, recent proliferation of position tracking devices has lead to enormous amounts of generated trajectory data, which resulted in a new research direction called *trajectory data clustering*. Depending on the task, given a set of trajectories, one may want to find clusters of objects that followed the same path or detect groups that moved together for given period of time. For instance, this information can be useful in urban planning when one may notice frequent routes that were not covered by public transportation [7]. While there are plenty of methods that are aimed towards clustering of trajectories (such as [10], [6]), not many of them discuss the data preprocessing step which is crucial for the final result. The closest work to our framework is proposed in [1]. In particular, the authors use the same stop detection algorithm that was used in our work. However, their work concentrates more on semantic enrichment of trajectories with the help of the users, whereas the emphasis of our work is on improving the automatization of trajectory data preprocessing and on the effects of data cleaning on the final clustering.

In this paper, we present a framework for cleaning trajectory data that consists of three phases: stop detection, missing segment interpolation, and inaccuracy removal. Using the Nokia-MDC [9] dataset and an objective quality metric, we evaluate our trajectory preprocessing framework by clustering both raw and the cleaned dataset and comparing the results. Our results show that indeed, our proposed preprocessing yields clusters of better quality.

This paper is structured as follows. Next, we briefly present a discussion about clustering and distance functions in general. Section 3 details each of the three steps that comprise our proposed framework. Then, some preliminary results are presented in Section 4 and some conclusions are offered in Section 5.

2. PRELIMINARIES

2.1 Clustering Algorithms

According to [8], trajectory clustering methods are divided into 3 main groups: model-based, distance based and visual-aided. Model-based methods attempt to describe the whole dataset by generating a suitable function of time (e.g. [6]). Distance-based methods use specially designed distance functions that are meant to show similarity between objects. This allows breaking the whole trajectory clustering process into two steps: 1) calculation of distances between trajectories according to the defined distance function and 2) actual clustering using a known clustering algorithm. In our work we will use this clustering approach to evaluate our findings. Lastly, visual-aided methods rely on human expert's judgment, who will interactively change clustering settings to achieve the desired clustering result. One such visual clustering framework can be found in [2].

Our work is based on a well-known clustering algorithm, DBSCAN [5], which is briefly described in the following. DBSCAN is a clustering method that is based on the notion of cluster density. Given two parameters that define cluster density, $minPts$ and Eps, clusters are formed by core points and their neighbours. A *core point* is a point that has a minimum number of points ($minPts$) within Eps distance. For each neighbour of a core point, the algorithm tries to expand the current cluster based on the same core point condition. That way, DBSCAN allows forming of clusters with arbitrary shapes. We will use a modification of this algorithm during the stop detection step and the algorithm itself during actual trajectory clustering.

2.2 Distance Functions

Most distance functions used for trajectory data are adopted from time series (data) mining domain. Dynamic Time Warping (DTW) [4] distance is designed to evaluate similarity between objects that have different speeds. Given two

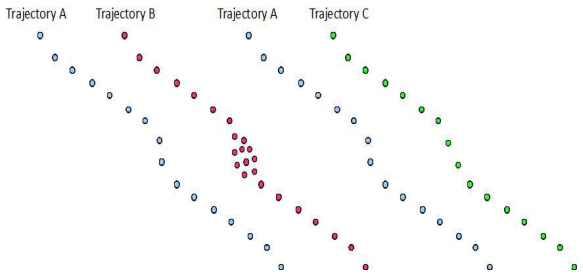

Figure 1: Trajectories with and without stop

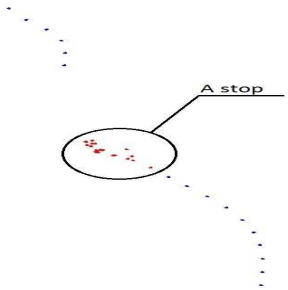

Figure 2: Stop detected by estimated *Eps* parameter

sequences of time series, all points of these sequences are warped to each other so that the resulting distance would be minimal. The major advantage of DTW over Euclidean distance is its ability to take into account stretching and compression of sequences. In this work, we used DTW as our distance function during the clustering step to illustrate our point that data preprocessing can improve clustering. Nevertheless, other distance functions could be used for comparison too (e.g. LCSS [12]).

3. TRAJECTORY CLUSTERING FRAMEWORK

3.1 Stop Detection

Our stop detection method is motivated by [11]. In that work, the authors propose an algorithm called CB-SMOT that finds stops - nearby places on the trajectory where object spent a relatively large time without leaving those locations. In our work we will use that algorithm to find these stops and remove them. We will also show that it is possible to make users specify only one parameter for stop detection, instead of two, as in the original paper.

The intuition behind removal of these stops is that many distance functions are quite sensitive to them. As shown in Figure 1, because DTW processes each point in a sequence, the distance between trajectory A (without stops) and similar trajectory B (with stops), would be larger compared to distance between trajectory A and trajectory C (without stops from trajectory B). This may lead to decreased accuracy of trajectory clustering.

CB-SMOT algorithm is based on the DBSCAN point clustering algorithm. The difference between these two algorithms is that instead of the $minPts$ parameter that is used in DBSCAN, CB-SMOT uses $minTime$ parameter to estimate cluster density with respect to how much time an object spent at particular locations.

Stop clusters (groups of points on the trajectory) are also formed based on core point condition. A point is considered a core point with respect to Eps and $minTime$, if $minTime \leq T_{last} - T_{first}$, where T_{last} (T_{first}) is the latest (the earliest) timestamp in the Eps-neighborhood of the core point. Lastly, for each core point the cluster is expanded further by including all the density-reachable points from the Eps-neighborhood.

One of the inconveniences of CB-SMOT is that a user should specify two parameters $minTime$ and Eps based on the trajectory data domain. The authors attempted to alleviate the choice of one parameter, Eps, by using another parameter called $area$, which represents approximate proportion of points that can form stops. While this allows the user to choose a normalized value between 0 and 1 instead of absolute value of Eps. The user, however, is still required to enter this parameter based on his/her knowledge about the data.

In our work, we tried to find a way to automate the selection of both parameters. While the automatic choice of $minTime$ is left for future work, we came to the observation that Eps parameter can be easily estimated by taking the mean of all the distances between consecutive points of a trajectory. Our preliminary experiments show that using this mean is sufficient to detect all the major stops on a trajectory. Results of the automatization of this step are illustrated in Figure 2. After all the stops on a trajectory are found, we remove them and fill in the created gap with points generated at the next step: *Missing Segment Interpolation*, which is described next.

3.2 Missing Segment Interpolation

Sometimes a GPS receiver loses signal and cannot record the object's current position. As a consequence, some trajectories contain gaps without sampling points as shown in Figure 2. In our work, we have implemented a simple interpolation technique that would emulate missing sampling points to achieve further improvement of clustering results.

First, let us define a *missing segment* in the context of this paper. A *missing segment* is part of a trajectory that, according to a given GPS sampling rate and object's movement direction, should be there, but is missing.

An obvious way to fill this gap would be to simply link the (temporarily) closest two points before and after the perceived gap. This would likely lead to problems because it effectively ignores the sampling rate and thus pieces of the missed subtrajectory. As the DTW distance compares the trajectories in a piece-wise fashion, this would impact the quality of the distance measure. Therefore, a more informed process is needed.

Let P_i and P_{i+1} be consecutive points on a trajectory with timestamps t_{P_i} and $t_{P_{i+1}}$, and let ϕ and ψ be the *interpolation* and *trajectory breaking* thresholds, respectively. If the difference between t_{P_i} and $t_{P_{i+1}}$ is larger than the interpolation threshold ϕ, we use interpolation to "complete" this missing segment. However, if this difference also exceeds the second threshold, *trajectory breaking threshold* ψ, we will not interpolate this segment. Instead, we will break down the trajectory into two separate trajectories. The motivation behind this is that for the clustering step we need trajectories of some reasonable length that would take into account only user's actual movement and neglect stops.

In other words, given a trajectory history of a user, if there

is no GPS signal for ψ amount of time (ψ is set to 3 minutes) between two consecutive points, we will separate the given trajectory into two subtrajectories. We note that this trajectory partitioning is performed at the very beginning of the trajectory preprocessing even before stop detection.
Suppose there is such a missing segment on a trajectory and our goal is to fill in this missing segment with generated and evenly distributed points. We performed an interpolation based on the previous k and next k points of both endpoints of the missing segment.
Let P be the list of points on a trajectory, where P_a and P_b are the endpoints of the missing segment. Given the distance between P_a and P_b, we can estimate the number of subsegments N, that is necessary for even distribution of points across the whole missing segment:

$$N = \left\lceil \frac{2kDist(P_a, P_b)}{\sum_{j=a-k}^{a-1} Dist(P_j, P_{j+1}) + \sum_{j=b}^{b+k-1} Dist(P_j, P_{j+1})} \right\rceil$$

where $Dist(P_a, P_b)$ is the Euclidean distance between P_a and P_b.
Then, the distance between two consecutive generated points P_i and P_{i+1} (where $a < i < b$) to fill in a missing segment is defined as:

$$Dist(P_i, P_{i+1}) = \frac{Dist(P_a, P_b)}{N}$$

In essence, we take the average distance between k consecutive previous (next) points of both endpoints of the missing segment.
Next, we create $N-1$ points starting from P_a towards P_b according to calculated distance $Dist(P_i, P_{i+1})$. After all necessary points are generated, we add a timestamp for each point based on the number of generated segments and on the time difference between two endpoints of the missing segment. This is done for clustering algorithms that consider dimension for their grouping criterion.
Note that this interpolation algorithm will also fill in missing segments that are resulted after stop removal.

3.3 Inaccuracy removal

In this work, the inaccuracy removal step is based on erroneous points that exist in many datasets, including Nokia MDC dataset. For unknown reasons, coordinates of such points in those datasets have low accuracy. For instance, we encountered gatherings of points that have exactly the same coordinates, but separated in time. Even though each of these points have their own timestamp, such a low accuracy would surely affect one's distance function. In addition, the stop detection algorithm that we used is unable to detect such gatherings if *minTime* parameter threshold is satisfied. Therefore, we have identified sets of points that had the same coordinates and removed them.
Second, some trajectories had points with null speed that were randomly changing their location with time. We assumed that this is due to GPS receiver inaccuracy (sometimes GPS location can be identified incorrectly when an object does not move) and eliminated these points too.

Lastly, after breaking down the raw dataset into separate trajectories, some trajectories needed to be disqualified from participation to the clustering step, because these trajectories would not add any meaningful knowledge to the final clustering. Thus, we eliminated trajectories that did not meet our threshold on the minimum number of points per trajectory (in our experiments we set this threshold to 10 points).

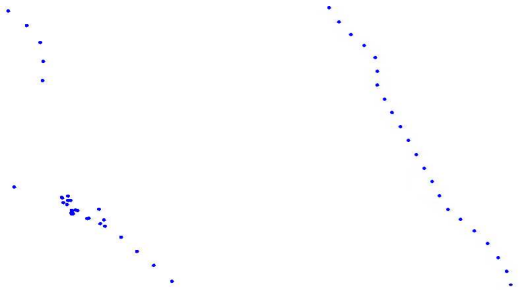

Figure 3: A trajectory before and after preprocessing

4. EXPERIMENTAL EVALUATION

To evaluate our framework, we implemented our proposal in Java programming language. First, we extracted subtrajectories from Nokia MDC Dataset by dividing each user's trajectory history as discussed in Section 3.2. Then, those subtrajectories were preprocessed according to our framework, which resulted in two datasets: initial raw dataset and a new, cleaned dataset. An example of applying such cleaning is illustrated in Figure 3.
In the next step, for both datasets we computed similarity matrices according to pairwise DTW distance between the trajectories. Then, both similarity matrices were given as an input to DBSCAN density-based clustering algorithm. Finally, we compared clusters resulted from raw dataset and clusters obtained from already cleaned trajectory data.
Since we do not have a "ground truth", it is not trivial to show how one clustering is better or more accurate than another. To our knowledge there is no widely recognized framework that would compare one clustering to another and accurately evaluate produced clusters. We evaluated the obtained clusters using the quality measure used in [10], which is called *QMeasure*. In essence, *QMeasure* attempts to minimize the sum of squared pairwise distances between elements that belong to one cluster (*Total SSE*), while penalizing for incorrectly identified noise points (*Noise Penalty*):

$$QMeasure = TotalSSE + NoisePenalty =$$
$$= \sum_{i=1}^{|C|} \left(\frac{1}{2|C_i| \sum_{x \in C_i} \sum_{y \in C_i} dtwDist(x,y)^2} \right) +$$
$$+ \frac{1}{2|F| \sum_{w \in F} \sum_{z \in F} dtwDist(w,z)^2}$$

where C is a set of clusters C_i, F is a set that contains noise trajectories and $dtwDist(x, y)$ is the DTW distance between trajectories x and y. Note that smaller *QMeasure* values imply more accurate clustering [10].
We compared raw and cleaned datasets for several users to see whether our trajectory cleaning framework actually improves clustering. As shown in Figure 4, clusterings resulted from cleaned dataset consistently had much smaller values for *QMeasure*. Noteworthy, the cleaned clusterings had flatter *QMeasure* curves, which suggests that the preprocessed

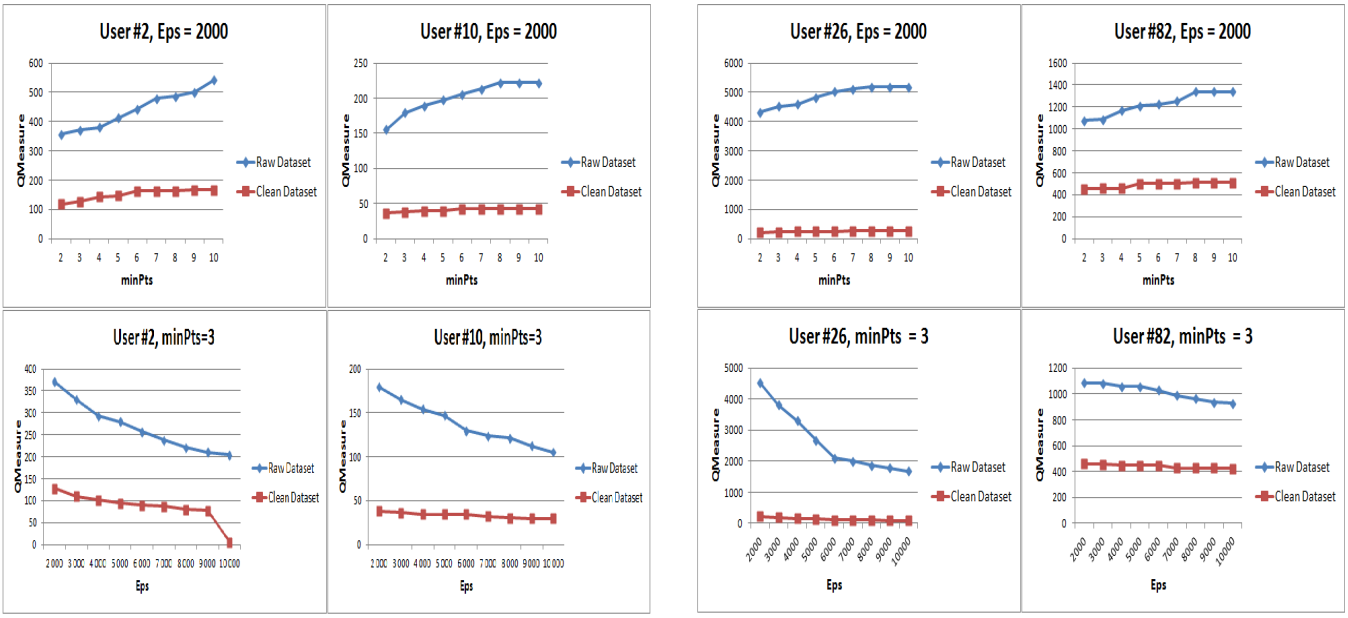

Figure 4: QMeasure for users # 2, # 10, # 26 and # 82

dataset is less sensitive to the choice of parameters for clustering algorithm.

Even though preliminary, our results indicate that, with respect to an objective quality measure, clusters obtained after our proposed cleaning process are of higher quality.

5. CONCLUSIONS AND FUTURE WORK

In this work we defined a trajectory cleaning framework for means of trajectory clustering. The chosen clustering quality measure shows that preliminarly processed dataset has a potential of generating more accurate clusters.

In future, we would like to develop more accurate and efficient methods of trajectory cleaning for each of the mentioned steps.

For stop detection, we will be working on the full automatization of this process. Current algorithm still relies on a user, who needs to specify parameters based on data, in particular *minTime* parameter. Determining this parameter based on given trajectory data would fully automate the process of trajectory preprocessing, because a user would not need to know any infomation about the dataset.

In terms of missing segment interpolation, we will seek for alternative, more sophisticated interpolation techniques. One of the objectives will be to produce "smoother" segments that take into account shapes of the previous and next segments.

One way to extend our current inaccuracy removal step would be to investigate the effects of *outlier detection and removal*. There are cases when certain points on a trajectory do not match with the general behaviour of the trajectory. For instance, one such point can be detected at location that is impossible to reach from the previous location in the given period of time. We believe, that it would be interesting to see how already existing outlier detection methods will complement our framework.

Finally, we would like to perform more thorough experiments that would compare the clusterings of raw and cleaned datasets.

6. ACKNOWLEDGEMENTS

Research partially supported by NSERC's DIVA Strategic Network.

7. REFERENCES

[1] L. O. Alvares, G. Oliveira, C. A. Heuser, and V. Bogorny. A framework for trajectory data preprocessing for data mining. In *Int. Conf. on Software Engineering and Knowledge Engineering*, pages 698 – 702, 2009.

[2] G. Andrienko, N. Andrienko, S. Rinzivillo, M. Nanni, D. Pedreschi, and F. Giannotti. Interactive visual clustering of large collections of trajectories. In *Visual Analytics Science and Technology*, pages 3 –10, 2009.

[3] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: ordering points to identify the clustering structure. In *Proceedings of SIGMOD Int. Conf. on Management of Data*, pages 49 – 60, 1999.

[4] D. J. Berndt and J. Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. In *Proceedings of KDD-94: AAAI Workshop on Knowledge Discovery in Databases*, pages 359 – 370, 1994.

[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Second International Conference on KDD*, pages 226–231, 1996.

[6] S. Gaffney and P. Smyth. Trajectory clustering with mixtures of regression models. In *Proceedings of the fifth ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, pages 63 – 72, 1999.

[7] F. Giannotti and D. Pedreschi, editors. *Mobility, Data Mining and Privacy - Geographic Knowledge*

Discovery. 2008.

[8] S. Kisilevich, F. Mansmann, M. Nanni, and S. Rinzivillo. Spatio-temporal clustering. In *Data Mining and Knowledge Discovery Handbook*. 2010.

[9] J. K. Laurila, D. Gatica-Perez, I. Aad, J. Blom, O. Bornet, T.-M.-T. Do, O. Dousse, J. Eberle, and M. Miettinen. The mobile data challenge: Big data for mobile computing research. In *Mobile Data Challenge by Nokia Workshop, in conjunction with Int. Conf. on Pervasive Computing*, 2012.

[10] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pages 593–604, 2007.

[11] A. T. Palma, V. Bogorny, B. Kuijpers, and L. O. Alvares. A clustering-based approach for discovering interesting places in trajectories. In *Proceedings of the ACM symposium on Applied computing*, pages 863–868, 2008.

[12] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Proceedings of the Int. Conf. on Data Engineering*, pages 673 –684, 2002.