Effectively Grouping Trajectory Streams

Gianni Costa, Giuseppe Manco, and Elio Masciari

ICAR-CNR
{costa,manco,masciari}@icar.cnr.it

Abstract. Trajectory data streams are huge amounts of data pertaining to time and position of moving objects. They are continuously generated by different sources exploiting a wide variety of technologies (e.g., RFID tags, GPS, GSM networks). Mining such amount of data is a challenging problem, since the possibility to extract useful information from this peculiar kind of data is crucial in many application scenarios such as vehicle traffic management, hand-off in cellular networks, supply chain management. Moreover, spatial data streams pose interesting challenges for their proper representation, thus making the mining process harder than for classical point data. In this paper, we address the problem of trajectory data streams clustering, that revealed really intriguing as we deal with a kind of data (trajectories) for which the order of elements is relevant. We propose a complete framework starting from data preparation task that allows us to make the mining step quite effective. Since the validation of data mining approaches has to be experimental we performed several tests on real world datasets that confirmed the efficiency and effectiveness of the proposed technique.

1 Introduction

The trajectory streams clustering challenge. Data Clustering is one of the most important mining techniques exploited in the knowledge discovery process[4]. Clustering huge amounts of data is a difficult task since the goal is to find a suitable partition in a unsupervised way (i.e. without any prior knowledge) trying to maximize the similarity of objects belonging to the same cluster and minimizing the similarity among objects in different clusters. Many different clustering techniques have been defined in order to solve the problem from different perspective, i.e. partition based clustering (e.g. *K-means*[7]), density based clustering (e.g. *DBScan*[2]), hierarchical methods (e.g. *BIRCH*[13]) and grid-based methods (e.g. *STING* [12]). Moreover, clustering methods have been exploited in a wide variety of application scenarios ranging from transactional data, text documents, XML data, etc. The main problem when clustering data is the high degree of uncertainty both in the data selection phase and in the definition of clusters, moreover due to complexity matter some algorithms do not scale-up very well when the size of the dataset becomes really huge.

Trajectory data streams are intrinsically quite difficult to be analyzed due to their *ordering* that makes the clustering task quite complex.

This work follows our first proposal for trajectory clustering shown in [8, 9]. Although our previous works gave some contribution to the trajectory clustering goal the techniques previously implemented are quite different since: 1) they are not designed for dealing with data streams; 2) we exploited an approach based on trajectory partitioning. We point out that, in this paper we tackle the clustering problem from a different point of view w.r.t. existing approaches (by ourselves and the other approaches in literature) both in the clustering definition (we work on the original trajectories) and the acquisition paradigm (we work on trajectory streams). The features of the presented approach guarantee more flexibility and better performances as will be shown in the experimental section.

2 Background

In this paper we tackle the problem of clustering large corpus of trajectory data streams. While for transactional data a tuple is a collection of features, a trajectory is an ordered set (i.e., a sequence) of timestamped points. Trajectory data are usually recorded in a variety of different formats, and they can be drawn from a continuous domain. We assume a standard format for input trajectories, as defined next.

Definition 1 (Trajectory). *Let P and T denote the set of all possible (spatial) positions and all timestamps, respectively. A trajectory is defined as a finite sequence $s_1, \cdots, s_N$, where $N \geq 1$ and each s_i is a pair (p_i, t_i) where $p_i \in P$ and $t_i \in T$.*

2.1 Data Pre-processing

Trajectory data are usually two dimensional, as mentioned above when a high accuracy is required we cannot disregard any point. Therefore, there is a need for a multi-resolution analysis to take into account both the spatial dimensions in the pre-processing step. We first give some preliminary definition on multi-resolution analysis.

Definition 2. *Let $\{S_m\}$ be a set of subspace of $L^2(\mathcal{R})$ and $m \in \mathcal{Z}$ such that the following hold: 1) $S_m \in S_{m+1}$; 2) $\bigcup_{j \in \mathcal{Z}} S_m = L^2(\mathcal{R})$; 3) $\bigcap_{j \in \mathcal{Z}} S_m = \emptyset$; 4) $x(t) \in S_m \Leftrightarrow x(\frac{t}{2} \in S_{m-1})$ then $\{S_m\}$ is a multi-resolution system.*

The choice of $\{S_m\}$ defines the analysis being performed in particular if we choose orthogonal subspace we have *orthogonal multi-resolution* analysis. We exploit here L space since it has a proper norm.

Trajectory data multi-resolution analysis aims at representing each trajectory being analyzed (and then elaborated using a mathematical transform) using a reliable set of coefficient. In order to perform this step allowing a *perfect* (i.e. lossless) reconstruction of trajectories we adopt the so called *Lifting Scheme* approach.

2.2 Lifting Schemes

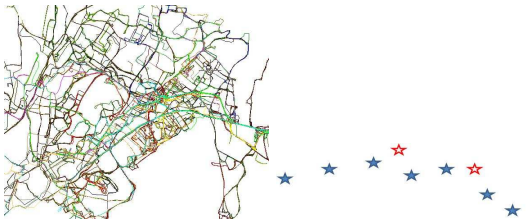
An effective approach for multi-resolution analysis is the *lifting scheme*. It was originally introduced for filtering signal and due to its intuitive features and more important because of its ability to exactly reconstruct the original input sequences it has been widely used also as a support for image compression in wavelet based systems. Moreover it is well suited as a preprocessing step for non separable transforms. In order to perform the proper lifting we need to define a filtering function. A filtering function $\mathcal{F}$, is a function that transform an input sequence I into an output sequence O according to an optimization function. Most widely used filters are Least Mean Square, Regression or Kalman filtering. In our implementation we exploited the Least Mean Square filter since it works by minimizing the least mean squares of the error signal, i.e. the difference between the computed and the actual signal. This choice offered best performances in our experimental setting as will be shown in the experimental section.

Performing lifting steps. Given a filtering function $\mathcal{F}$ and a input trajectory Tr_x, the trajectory can be split in two subsequences Tr_{x_o} and Tr_{x_e} that are respectively the sequence of odd and even indices. The trajectory lifting is performed by iteratively updating the subsequences with their predicted version in order to obtain two shorter sequences that are representative of the *original* sequence (say it Tr'_x) and the *trend* sequence (say it Tr_h). Obviously, when performing this step some errors could arise (say it Tr_e). More formally, let $\mathcal{P}$ and $\mathcal{U}$ two filtering function exploited for prediction and update, a generic lifting step works as follows:

- $Tr_e(k) = Tr_{x_o}(k) - \mathcal{P}(Tr_{x'}(k))$;
- $Tr_h(k) = Tr_{x_e}(k) - \mathcal{U}(Tr_{x_e}(k))$.

By iterating the above steps, we obtain a succinct representation of the original trajectory with no loss of information. The proposed lifting scheme will be used for implementing the non separable transform based clustering described in next sections, since when the trajectory size becomes unpractical we will reduce it by a lifting step without decreasing the information quality. Furthermore this step allow us to make trajectories equal length so the successive transforms will better compare them without needing any alignment or interpolation operation. To better clarify the proposed pre-elaboration steps we provide a toy example. Consider the trajectories depicted in Fig. 1(a) regarding buses movements in the Athens metropolitan area. Applying the lifting scheme defined above on a sample trajectory will produce the sequence in Fig. 1(b) where the solid (blue) stars represent the *trend* sequence, while the empty (red) stars represent the error sequence. It is easy to see that the number of points taken into account after lifting is really smaller than the original trajectory and this feature is particularly useful when considering (real life) longer trajectories having more complex shapes. Moreover, lifted trajectories can be updated as new data points arrive: indeed, incremental computation is a key requirement for streaming algorithms. Fur-

thermore, we can make the lifted trajectories equally sized in order to enhance the clustering performances.



(a) Original Data (b) A lifted trajectory

Fig. 1. Trajectory Lifting Example

3 Exploiting Fourier Transforms for Spatial Quincunx Lattices based Clustering

In this section we exploit non separable transforms in order to effectively manage two dimensional trajectories. Non separable transforms allow us to consider the whole trajectory taking into account both dimensions in the computation thus avoiding any approximation due to mono-dimensional transform composition. We will exploit them in a suitable way in order to catch similarity among trajectories. The first step to be performed for any mathematical transform is to define the basis function and the features of the search space where data reside. After a deep investigation of several trajectory datasets we found that the best representation for the trajectory search space is a *Quincunx Lattice*[1]. Indeed, optimal sampling scheme in the two-dimensional space is the hexagonal lattice. Unfortunately, hexagonal lattice is quite unwieldy in terms of hardware and software implementations. An appealing compromise is the quincunx lattice that is a sublattice of the square lattice. The quincunx lattice has a diamond tessellation which is closer to optimal hexagon tessellation than square lattice, and it can be easily generated by down sampling conventional digital images without any hardware change. Indeed, for this reason, quincunx lattice is widely adopted in many application dealing with spatial images[14]. Based on the above mentioned consideration exploiting a quincunx lattice will allow us to represent compactly and preserving accuracy even trajectories laying close to the edges of the search space. The first step is the construction of the algebra (details are not relevant for this paper, more details in [10]) that induces the spatial signal spectrum defined below:

[1] It is always possible to find a basis that allows this representation for the search space

- $u_k = \cos(\frac{k+1/2}{n/2} \cdot \pi), 0 < k < n/2$
- $v_l = \cos(\frac{l+1/2}{n/2} \cdot \pi), 0 < l < n/2$
- $w_{k,l,\pm} = \pm\frac{1}{2}\sqrt{(1+u_k)\cdot(1+v_l)}$

Once obtained the spatial signal spectrum we can easily define the distance between two trajectories by considering their spectra. We recall that we assume that each trajectory point is an impulse in the overall signal associated to the trajectory. In particular, given two trajectories Tr_1 and Tr_2 and their spectra $Q(Tr_1)$,$Q(Tr_2)$, we compute their angular distances as $d_{\widehat{\alpha\beta}} = \arccos(u_{1k}) - \arccos(u_{2k})$ and $d_{\widehat{\gamma\delta}} = \arccos(v_{1l}) - \arccos(v_{2l})$. For each pair of $w_{k,l,\pm}$ we compute the modulo distance as $d_w = \sqrt{w^2_{1k,l,\pm} - w^2_{2k,l,\pm}}$. Finally, we define the overall *Quincunx* based distance as:

$$dist_Q(Tr_1, Tr_2) = \sqrt{\sum_{k=1}^{n^2} d_w(k)^2} \cdot \cos(\mu(d_{\widehat{\alpha\beta}}) - \mu(d_{\widehat{\gamma\delta}}))$$

where $\mu(d_{\widehat{\alpha\beta}})$ is the average angle distance between each pair of u_k and $\mu(d_{\widehat{\gamma\delta}})$ is the mean between each pair of v_l. The distance so far defined is able to catch dissimilarity between trajectories since it considers the difference in angular distances between the two trajectories (the cos argument) while taking into account the overall extension of the spatial signal (the modulo part). This is a crucial feature since when using trajectory simplification techniques defined in literature we could lose relevant information about trajectories.

Clustering trajectory streams. We briefly recall the notion of trajectory cluster.

Definition 3 (Trajectory Cluster). *Given a set of trajectories T, a cluster is a subset $C \subseteq T$ such that the distance between each pair of trajectories in C is minimum, and the distance between each pair of trajectories $Tr_i \subseteq C$ and $Tr_j \nsubseteq C$ is maximized w.r.t. the chosen metric.*

Our algorithm described below is tailored for clustering evolving streams of trajectories, thus the dataset W being mined is defined as a sliding window over the continuous stream. W moves forward by a certain amount. Each window W either contains the same number of trajectories (*count-based* or physical window), or contains all trajectories arrived in the same period of time (*time-based* or logical window). The window is maintained by adding the new slide (δ^+) and dropping the expired one (δ^-). Therefore, the successive instances of W are referred as $W_1, W_2, \cdot$. The number of trajectories that are added to (and removed from) each window is called its slide size. In this paper, for the purpose of simplicity, we assume that all slides have the same size, and also each window consists of the same number of slides. Thus, $n = |W| \div |S|$ is the number of slides (a.k.a. panes) in each window, where $|W|$ denotes the window size and $|S|$

Method: IncrementalClusteringMaintenance
Input:
A trajectory stream T.
Output:
A trajectory clustering C_T.
Vars:
A new slide S_i of the input trajectories;
An expiring slide S_{exp} of the input trajectories;
A set of lifted trajectories T_L
0: $C_T = \emptyset$
1: **For Each** New Slide S_i
2: $lift(T_L, S_i)$;
3: $C_{Ti} = mineNewCluster(T_L)$;
4: $mergeClusters(C_T, C_{Ti}$
5: **For Each** trajectory $t \in T_L$
6: $annotate(t)$;
7: **For Each** Expiring Slide S_{exp}
8: $deleteOldestTrajectories(C_T)$;
9: $compactClusters(C_T)$;

Fig. 2. The incremental clustering algorithm

denotes the size of the slides. The incremental clustering algorithm is reported in Fig. 2.

As a new window slide S_i is loaded we compute the lifting of trajectories contained in S_i. We compute the clustering of the lifted trajectories ($mineNewCluster$) by running k-means++, a stable k-means improvement that has been proposed in [1], using a distance-based probabilistic algorithm $O(log(k))$ that makes it competitive with optimal clustering and avoid the initial cluster assignment problem. To keep fresh clusters and to avoid concept drift we merge the existing clusters with the new computed one ($mergeClusters$) by minimizing $dist_Q$ between each pair of trajectories to be merged. More in detail, given the centers of the candidate clusters we assign the center of the new cluster as the trajectory that minimize the $dist_Q$ w.r.t previous centers, then we eventually discard trajectories that are now closer to a different existing cluster than the one being created. The result of this step is a cluster assignment that is continuously updated, thus it results impervious to the eventual concept drift. After clustering computation we annotate the trajectories (by their timestamps) in the current window slide to allow successive delta maintenance (function *annotate*). Finally, as a window slide expires we discard the oldest trajectories and re-compact the clusters that could results loosely compact after trajectory deletions (we perform $compactClusters$ by recomputing cluster centers w.r.t. $dist_Q$ minimization as in $mereclusters$). In this respect, we recompute the clustering including the new trajectories and assigning new cluster centers (that are carefully chosen by the clustering algorithm we exploit), this will allow us to take into account the eventual concept drift. Note that no threshold information are provided by the

user since the provided distance is a metric thus it preserves the k-means++ features.

4 Experimental Results

In this section, we present the experiments we performed to assess the effectiveness of the proposed approach in clustering trajectories. To this purpose, a collection of tests is performed, and in each test some relevant groups of homogeneous trajectories (*trajectory classes*) are considered. The direct result of each test is a similarity matrix representing the degree of similarity for each pair of trajectories in the data set. The evaluation of the results relies on some *a priori* knowledge about the trajectory classes being used that was obtained by domain experts or available from the datasets providers.

We set up two classes of experiments: 1) we tested our algorithm (we refer to it in the following as $Fourier_{2D}$) in a *static* context, i.e. we considered datasets that can fit in a single window. The comparison in this case is made against TRACLUS [5] and since TRACLUS is a parametric algorithm we report here the best parameters assignment we obtained after an accurate tuning and suggested by the authors in [6]. This set of tests is intended to evaluate the accuracy of the clustering, and we choose TRACLUS since it is (at the best of our knowledge) the most accurate system available for static trajectory clustering and 2) we tested the *dynamic* streaming performances by tuning the windows size and measuring the effectiveness and the efficiency against TCMM [6] that is the most accurate proposal available for clustering trajectory data streams at the best of our knowledge.

Static Performance Evaluation. We performed several experiments on a wide variety of real datasets. More in detail we analyzed the following data: 1) *School Bus*: it is a dataset consisting of 145 trajectories of 2 school buses collecting (and delivering) students around Athens metropolitan area in Greece for 108 distinct days; 2) *Animals*, it is a dataset containing the major habitat variables derived for radio-telemetry studies of elk, mule deer, and cattle at the Starkey Experimental Forest and Range in northeastern Oregon[2].

In order to perform a simple quantitative analysis we produce for each test a similarity matrix, aimed at evaluating the resulting intra-cluster similarities (i.e., the average of the values computed for trajectories belonging to the same cluster), and to compare them with the inter-cluster similarities (i.e., the similarity computed by considering only trajectories belonging to different classes). To this purpose, values inside the matrix can be aggregated according to the cluster of membership of the related elements: given a set of trajectories belonging to n prior classes, a similarity matrix S about these trajectories can be summarized by a $n \times n$ matrix CS, where the generic element $CS(i, j)$ represents the average similarity between cluster i and cluster j.

[2] http://www.fs.fed.us/pnw/starkey/data/tables/index.shtml

$$CS(i,j) = \begin{cases} \frac{\sum_{x,y \in C_i, x \neq y} DIST(x,y)}{|C_i| \times (|C_i|-1)} & \text{iff } i = j \\ \frac{\sum_{x \in C_i, y \in C_j} DIST(x,y)}{|C_i| \times |C_j|} & \text{otherwise} \end{cases}$$

where $DIST(x, y)$ is the chosen distance metric ($TRACLUS$ metric or the $dist_Q$).

The above definition is significant since we normalize the different metrics pertaining to the different approaches, this will allow us to compare performance in the ideal setting for both approaches.

The higher are the values on the diagonal of the corresponding CS matrix w.r.t. those outside the diagonal, the higher is the ability of the similarity measure to separate different classes. In the following we report a similarity matrix for each dataset being considered, as it will be clear the reported results show that our technique is quite effective for clustering the datasets being considered and outperforms $TRACLUS$.

Measuring Effectiveness for School Bus. For this dataset our prior knowledge is the set of trajectories related to the two school buses that define the two clusters in the dataset. Our algorithm is able to exactly detect the two clusters. We present the results using the two classes but we point out that our technique is able to (eventually) further refine the cluster assignment identifying the micro-clusters represented by common sub-trajectories. As it is easy to see in Fig. 3(a) and (b) $Fourier_{2D}$ outperforms $TRACLUS$ by allowing a perfect assignment to the proper class to each trajectory.

$TRACLUS$	Bus 1	Bus 2
Bus 1	0.9790	0.8528
Bus 2	0.8528	0.9915

(a)

$Fourier_{2D}$	Bus 1	Bus 2
Bus 1	1	0.6250
Bus 2	0.6250	1

(b)

Fig. 3. $TRACLUS$ and $Fourier_{2D}$ similarity matrices for Bus dataset

The presence of several turns made by the buses makes the feature of $Fourier_{2D}$ well suited for this dataset since it takes into account all the angular values of the trajectory.

Measuring Effectiveness for Animals. In this case we considered as a class assignment the different trajectories traversed by elk, mule deer, and cattle. Also in this case our approach correctly identify the three cluster present in the dataset. We point out that it is worth studying animal data because the trajectories are in unrestricted space rather than on well known road network. In this case there were 3 main classes as it is shown in Fig. 4(a) and (b). Also in this case $Fourier_{2D}$ outperforms $TRACLUS$. As we can see, differences among the various classes are marked with higher precision by $Fourier_{2D}$. This is mainly due to the fact that our approach is quite discriminative since it takes into account both angular and modulo distances in the spectrum of a trajectory.

$Fourier_{2D}$	elk	mule deer	cattle
elk	0.9986	0.7759	0.7055
mule deer	0.7759	0.9889	0.7566
cattle	0.7055	0.7566	0.9920

(a)

$TRACLUS$	elk	mule deer	cattle
elk	0.9885	0.7439	0.7108
mule deer	0.7439	0.9899	0.7223
cattle	0.7108	0.7223	0.9874

(b)

Fig. 4. $TRACLUS$ and $Fourier_{2D}$ similarity matrices for $Animals$ dataset

Quality Measures Evaluation. Distance minimization is a natural and widely used norm of similarity, but a devil's advocate can point out that other clustering algorithms might not measure their effectiveness in terms of this metric or even the compactness and homogeneity of each cluster around its centroid. Thus, in this section we will attempt to measure the quality of the clusters produced by $Fourier_{2D}$ using very different criteria inspired by the nearest subclass classifiers that were previously used in a similar role in [11] and [3]. A first relevant evaluation measure is the error rate of a k-Nearest Neighbor classifier defined on the basis of the similarity measure. For a given trajectory, we can check whether the dominant class of the k most similar elements allows to correctly predict the actual class of membership. Thus, the total number of trajectories correctly predicted can be considered as a measure for evaluating the effectiveness of the similarity at hand. Formally, the error $e_k(S)$ of a kNN classifier exploiting a similarity matrix S can be defined as

$$e_k(S) = \frac{1}{N} \sum_{i=1}^{N} \gamma_k(i)$$

where N is the total number of trajectories, and $\gamma_k(i)$ is 0 if the predicted class of the i-th trajectory coincides with its actual class, and 1 otherwise. This value will measure the errors made in the cluster computation. Low values of the $e_k(S)$ index correspond to good results.

The above measure can be refined by evaluating the average number of elements, in a range of k elements, having the same class of the trajectory under consideration. Practically, we define q_k as the average percentage of trajectories in the k-neighborhood of a generic trajectory belonging to the same class of that trajectory. Formally:

$$q_k(S) = \frac{1}{N} \sum_{i=1}^{N} \frac{|\mathcal{N}_k(i) \cap Cl(i)|}{min(k, n_i)}$$

where $Cl(i)$ represents the actual class associated with the i-th trajectory in the dataset, $n_i = |Cl(i)|$, and $\mathcal{N}_k(i)$ is the set of k trajectories having the lowest distances from Tr_i, according to the similarity measure at hand. In principle, a Nearest Neighbor classifier exhibits a good performance when q_k is high, since high values of q_k indicate a great *purity* of the clustering. Furthermore, q_k provides a measure of the stability of a Nearest-Neighbor: high values of q_k make a

kNN classifier less sensitive to increasing values k of neighbors considered. The sensitivity of the similarity measure can also be measured by considering, for a given group of trajectories x, y, z, the probability that x and y belong to the same class and z belongs to a different class, but z is more similar to x than y is. We denote this probability by $\varepsilon(S)$, which is estimated as

$$\varepsilon(S) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{(n_i-1)(N-n_i)}\sum_{Cl(j)=Cl(i),j\neq i}\ \sum_{Cl(k)\neq Cl(i)}\delta_S(i,j,k)\right)$$

where δ_S is 1 if $S(i,j) < S(i,k)$, and 0 otherwise. Once again, low values of $\epsilon(S)$ denote a good performance of the similarity measure under consideration, since they indicates a low *ambiguity* in the clustering.

The quality values obtained by the various techniques are reported in the following Tables. As in the previous section we first show the Table for Bus dataset. For measures e_k and q_k we considered neighborhoods of size 70, i.e. the actual size of each class in the data set. As it is easy to see the results shown in Fig. 5 confirm the ones obtained by similarity matrix, thus assessing that for Bus dataset $Fourier_{2D}$ outperforms $TRACLUS$.

Fig. 5. Quality indices for our datasets

Bus

	ε	$e_{k=70}$	$q_{k=70}$
$Fourier_{2D}$	0.0113	0.0235	0.9965
$TRACLUS$	0.1811	0.0997	0.9658

Animals

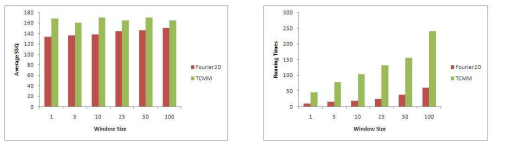
	ε	$e_{k=50}$	$q_{k=50}$
$Fourier_{2D}$	0.0792	0.1225	0.8945
$TRACLUS$	0.1801	0.1887	0.6257

For $Animals$ dataset, measures e_k and q_k are evaluated considering neighborhoods of size 50, i.e. the actual size of each class in the data set. Again the results shown in Fig. 5 confirm the ones obtained by similarity matrix, thus assessing that for $Animals$ dataset $Fourier_{2D}$ still outperforms $TRACLUS$.

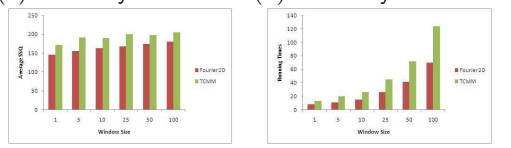
Measuring Performance on Streams. In previous section we assessed the accuracy of the approach for detecting clusters. In this section we will measure the performances of our algorithm when dealing with stressing trajectory streams. We report the results obtained for two huge real life datasets: 1) *Hurricanes* trajectory dataset. It is a dataset containing data about Atlantic hurricanes since 1851, it includes position in latitude and longitude, maximum sustained winds in knots, and central pressure in millibars [3]; 2) *GPS* trajectory dataset collected in (Microsoft Research Asia) *GeoLife* project [15] by 165 users in a period of over two years (from April 2007 to August 2009). This dataset recoded a broad range of users outdoor movements, including not only life routines like go home and go to work but also some entertainments and sports activities, such as shopping, sightseeing, dining, hiking, and cycling. Therefore, the dataset allows a severe test for our clustering approach.

[3] available at http://weather.unisys.com/hurricane/atlantic/

We ran several tests varying the window size ($|W|$) thus using the *count-based* approach for comparison purposes w.r.t. TCMM. Moreover, in order to perform a better comparison against TCMM we computed the average SSQ as described in [6]. In particular, we find for each cluster its centroid and then compute the SSQ using our $dist_Q$ in the well known SSQ formula. In Fig. 6(a) and (b) we report the results for *Hurricanes* dataset w.r.t. varying window size expressed in MBytes while times are expressed as seconds, while in Fig. 6(c) and (d) we report the results obtained for *GPS* dataset. $Fourier_{2D}$ (the left red bar) outperforms $TCMM$ (the right green bar) both in terms of execution times and accuracy. The faster execution is due to the incremental computation of the lifted trajectories that allows us to save execution time as new point are added, while the accuracy result is due to the peculiar features of $dist_Q$ that takes into account all possible differences between trajectories as explained above even when incrementally maintained.



(a) Accuracy Hurricanes (b) Efficiency Hurricanes



(c) Accuracy GPS (d) Efficiency GPS

Fig. 6. Performance Comparison w.r.t. TCMM

Quality Measures Evaluation. The quality values obtained by the $Fourier_{2D}$ and $TCMM$ are reported in the following Tables. We first show the Table for $Hurricanes$ dataset. For measures e_k and q_k we considered neighborhoods of size 100, i.e. the actual size of each class in the data set. As it is easy to see the results shown in Fig. 7 assess that for $Hurricanes$ dataset $Fourier_{2D}$ outperforms $TCMM$.

Fig. 7. Quality indices for our datasets

Hurricanes

	ε	$e_{k=100}$	$q_{k=100}$
$Fourier_{2D}$	0.0097	0.0354	0.9877
$TCMM$	0.1433	0.1025	0.9551

GPS

	ε	$e_{k=75}$	$q_{k=75}$
$Fourier_{2D}$	0.1633	0.2895	0.7781
$TCMM$	0.2534	0.4033	0.6021

Concerning GPS dataset for measures e_k and q_k we considered neighborhoods of size 75, i.e. the actual size of each class in the data set. Again the results shown in Table 7 confirm that also for GPS dataset $Fourier_{2D}$ outperforms $TCMM$.

5 Conclusion

In this paper we addressed the problem of detecting clusters in trajectory data. The technique we have proposed is mainly based on the idea of representing a trajectory with its lifted version. Thereby, the similarity between two trajectories can be computed by analyzing their Fourier transforms in the two-dimensional case. Experimental results showed the effectiveness of the approach in detecting common clusters for trajectories and robustness to the eventual concept drift.

References

1. D. Arthur and S. Vassilvitskii. k-means++ the advantages of careful seeding. In *SODA*, pages 1027–1035, 2007.
2. M. Ester, H.P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996.
3. S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese. Fast detection of xml structural similarity. *IEEE TKDE*, 17(2):160–175, 2005.
4. J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.
5. J.G. Lee, J. Han, and K.Y. Whang. Trajectory clustering: a partition-and-group framework. In *SIGMOD*, 2007.
6. Z. Li, J.G. Lee, X. Li, and J. Han. Incremental clustering for trajectories. In *DASFAA (2)*, pages 32–46, 2010.
7. S. Lloyd. Least squares quantization in pcm. *IEEE TOIT*, 28, 1982.
8. E. Masciari. A complete framework for clustering trajectories. In *ICTAI*, pages 9–16, 2009.
9. E. Masciari. Trajectory clustering via effective partitioning. In *FQAS*, pages 358–370, 2009.
10. M. Puschel and M. Rotteler. Fourier transform for the directed quincunx lattice. In *I.C.A.S.S.P.*, 2005.
11. C. J. Veenman and M. J. T. Reinders. The nearest subclass classifier: A compromise between the nearest mean and nearest neighbor classifier. *IEEE PAMI*, 27(9):1417–1429, 2005.
12. W. Wang, J. Yang, and R. R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *VLDB*, pages 186–195, 1997.
13. T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *SIGMOD*, pages 103–114, 1996.
14. X. Zhang, X. Wu, and F. Wu. Image coding on quincunx lattice with adaptive lifting and interpolation. In *Data Compression Conf.*, pages 193–202, 2007.
15. Y. Zheng, L. Zhang, X. Xie, and W. Y. Ma. Mining interesting locations and travel sequences from gps trajectories. In *WWW*, pages 791–800, 2009.