Mining Trajectory Databases via a Suite of Distance Operators

Nikos Pelekis[1] *Ioannis Kopanakis*[2] *Irene Ntoutsi*[1] *Gerasimos Marketos*[1] *Yannis Theodoridis*[1,3]
[1]*Dept. of Informatics, Univ. of Piraeus, Greece*
{npelekis, ntoutsi, marketos, ytheod}@unipi.gr
[2]*Technological Educational Institute of Crete, Greece*
i.kopanakis@emark.teicrete.gr
[3]*Research Academic Computer Technology Institute*
ytheod@cti.gr

Abstract

With the rapid progress of mobile devices and positioning technologies, Trajectory Databases (TD) have been in the core of database research during the last decade. Analysis and knowledge discovery in TD is an emerging field which has recently gained great interest. Extracting knowledge from TD using certain types of mining techniques, such as clustering and classification, impose that there is a mean to quantify the distance between two trajectories. Having as a main objective the support of effective similarity query processing, existing approaches utilize generic distance metrics that ignore the peculiarities of the trajectories as complex spatio-temporal data types. In this paper, we define a novel set of trajectory distance operators based on primitive (space and time) as well as derived parameters of trajectories (speed and direction). Aiming at providing a powerful toolkit for analysts who require producing distance matrices with different semantics as input to mining tasks, we develop algorithms for each of the proposed operators. The efficiency of our approach is evaluated through an experimental study on classification and clustering tasks using synthetic and real trajectory datasets.

1. Introduction

A Trajectory Database (TD) consists of objects whose location changes over time (e.g. moving humans or vehicles). With the integration of wireless communications and positioning technologies, the concept of TD has become increasingly important, and has posed great challenges to the data mining community [14]. In this work, we initially study the problem of trajectory similarity search in TD, where, given the trajectories of two moving objects (i.e., the sequence of their locations with respect to time), we detect and quantify their (dis-)similarity, hence their distance. Having in hand a powerful set of distance operators, each of them describing semantically different interrelation properties between trajectories, we investigate their utilization in clustering and classification tasks, which fundamentally rely on the notion of distance among the data under analysis.

The support of efficient trajectory similarity techniques is indisputably very important for the quality of data analysis tasks in TD. This justifies the fact that during the last decade there has been a lot of work in the literature regarding trajectory similarity search. Most of the existing approaches so far are mainly inspired by the time series analysis domain and propose generic similarity metrics for 1D data [2], [17], [6]. Other approaches deal with some basic trajectory features [28], [29], [33], [9], [10], such as the different sampling rates on trajectories, the different speeds of moving objects, the possible outliers that might be introduced due to an anomaly in data collection procedure, the different scaling factors, the different trajectory lengths, etc. The common characteristic of previous works is that they are interested in the movement shape of the trajectories, which are usually considered as 2D or 3D time series data. In other words, what is important in measuring the similarity between two trajectories is just the sequences of the sampled positions. This means that the temporal dimension is ignored, leaving the time recordings out of the knowledge discovery process. In real world applications though, trajectories are represented by finite sequences of time-referenced locations. What is more, such sequences may result from *time-based* (e.g. every 30 seconds), *change-based* (e.g. when the location of an entity deviates from the previous one by a given threshold), *location-based* (e.g. when a moving object is close to a sensor), *event-based* recording (e.g. when a user requests for localization), or even various combinations of these basic approaches [1]. A different perspective is required therefore, capable of coping with

real world application scenarios. In addition to the above, TD introduce issues related with derived parameters of motion, such as speed and direction.

To the best of our knowledge, there is no work on classifying the different similarity types that can be defined based on these underlying features of the trajectories. In this paper, we provide such a classification and propose novel distance operators, which can be exploited by mining procedures intrinsically requiring distance information. Our approach takes under consideration all the factors that characterize a trajectory (locality, temporality, directionality, rate of change) and formulates a flexible framework for the comparison of trajectories based on the above factors. Furthermore, the incremental and partial nature of the defined distance operators makes them capable of comparing trajectories partially, identifying similarities even in portions of their route. This is in contrast to other approaches that only define global metrics for the whole lifespan of the trajectories. As a motivating example, consider a location-based data management system, which monitors the movement of GPS-equipped moving objects; public transport means, animals under supervision or humans (walkers or drivers) could be real world examples. Experts in the field would be advantaged if they could run analysis tasks, such as:

- Task 1 – *spatiotemporal similarity*: Find clusters of objects that follow similar *routes* (i.e., projections of trajectories on 2D plane) during the same time interval (e.g. co-location and co-existence from 3pm to 6 pm) and
- Task 2 – *(time-relaxed) spatial-only similarity*: Find clusters of moving objects taking only their *route* into consideration (i.e., irrespective of time, direction and sampling rate).

and variations of the above, such as:

- Task 3 – *speed-pattern based spatial similarity*: Find clusters of objects that follow similar routes and, additionally, move with a similar speed pattern, and
- Task 4 – *directional similarity*: Find clusters of objects that follow a given direction pattern (e.g. NE during the first half of the route and subsequently W).

Since our framework is based on query processing operators, the novelty of our approach is augmented by the following two inter-related facts: (1) the combination of the tasks (using AND/OR clauses) provides analysis functionality unmatched so far (e.g. “find trajectories that moved closely in space but following opposite directions and with very dissimilar speed”); (2) the output of each of the supported operators defines similarity patterns that can be utilized to reveal local similarity features (e.g. “find the most similar portions between two dissimilar trajectories”).

The major contributions of this paper are the following:

- We define two main types of trajectory similarity search, spatiotemporal and (temporally-relaxed) spatial similarity, as well as two variations, speed-pattern based spatial and directional similarity.
- For each type of similarity query we introduce distance operators, and we propose respective query processing algorithms.
- We demonstrate how these algorithms implicitly define similarity patterns to expose stratified commonalities, wherein portions of trajectories that are similar to each other may be detected.
- We conduct a comprehensive set of experiments over synthetic and real trajectory datasets, in order to thoroughly evaluate our approach supporting classification and clustering tasks.

The rest of the paper is structured as follows: Problem statement and related work are discussed in Section 2. In Sections 3, 4 and 5, we describe in detail the different types of similarity search (spatial, spatiotemporal and variations, respectively) and respective search algorithms. In Section 6, we present the results of our experimental study. Section 7 concludes the paper and provides ideas for future work.

2. Problem Statement and Related Work

Before we define the different types of similarity search addressed in this paper, we first present the notations utilized hereafter. Let D be a database of N moving objects with object ids $\{o_1, o_2, \ldots, o_N\}$. Assuming linear interpolation between sampled locations, the trajectory T_i of a moving object o_i consists of a set of 3D Line Segments (3DLS), where each 3DLS represents the continuous development of the moving object during sampled locations. In other words, the movement of an object from a starting position (x_s, y_s) to an ending position (x_e, y_e) during a time period $[t_s, t_e]$ is described by a linear function of time $f_i(t)$. Projecting T_i on the spatial 2D plane (temporal 1D line), we get the *route* r_i (the *lifespan* l_i) of o_i. Moreover, additional motion parameters can be derived by $f_i(t)$, including speed s, direction angle φ, etc. Obviously, no assumptions of equal distanced time intervals between the sampled points are posed.

Definition 1: Let D be a database of trajectories T_i and Q be a (reference) trajectory consisting of a set of 3DLS. The *Most-Similar-Trajectory* (MST) S in D with respect to Q is the one that minimizes a distance measure $Dist(Q,T_i)$.

The distance measure $Dist(Q,T_i)$ is application-driven and may involve any combination of trajectory features, such as spatial projection (route), temporal projection (lifespan), speed and direction. Taking both route and lifespan into consideration, $Dist(Q,T_i)$ addresses Task 1 presented in Section 1; considering only route Task 2 is addressed, and so on.

Route, lifespan, speed and direction of a moving object trajectory are classified as *motion dependant* parameters. There also exist *data dependant* parameters that affect similarity search, such as length, scale, shift, sampling rate and outliers' existence, which have been the main research issues in related work. In this paper, we focus on the former class of parameters, as we treat the problem from a TD perspective.

Most of related work in trajectory similarity search is inspired by the time series analysis domain, based on mapping a trajectory into a vector in a feature space and using an L_p-norm distance function to define the similarity measure. The advantage of this approach is that it exploits on a dimensionality reduction technique, which allows the similarity between the trajectory vectors in the time domain to be approximately equal to the similarity between the two points in the feature domain.

Agrawal et al. [2] adopt the Discrete Fourier Transformation (DFT) to be the feature extraction technique since DFT preserves the Euclidean distance and, furthermore, only the first few frequencies are important. Rafiei and Mendelzon [24] use Fourier descriptors to represent shapes boundaries, compute a fingerprint for each shape and build a multidimensional index (R-tree) on fingerprints. The distance between two shapes is approximated by the distance of the corresponding fingerprints. This distance is not affected by variations in location, size, rotation and starting point.

Although Euclidean measures are easy to compute, they do not allow for different baselines or different scales. The main drawback of these methods is that their performance degrades in the presence of noise and outliers since all elements should be matched. To address the disadvantages of the L_p-norm, Goldin and Kanellakis [15] use normalization transformations, i.e., they normalize sequences and compute the similarity between normalized sequences. Although this method solves some problems like the different baselines, it is still sensitive to phase shifts in time and does not allow for acceleration along the time axis. Lee et al. [18] compute the distance between two multidimensional sequences by finding the distance between minimum bounding rectangles. Two approaches, which also use Euclidean distances, include the lower bound techniques [5] and the shape-based similarity query [30]. However, both approaches can be applied only on trajectories with same lengths.

Another approach is based on Dynamic Time Warping (DTW) technique that allows stretching in time in order to get a better distance [3]. DTW has been adopted in order to measure distances between two trajectories that have been represented as path and speed curves [19] or as sequences of angle and arc-length pairs [27]. Sakurai et al. [31] present the Fast search method for Dynamic Time Warping, utilizing a new lower bounding distance measure that approximates the time warping distance. Lin and Su [32] introduce the "One Way Distance" (OWD) function and they prove that their approach outperforms DTW algorithm as far as precision and performance is concerned. The DTW similarity measure suffers from shortcomings such as sensitivity to different baselines and scales of the sequences that can be reduced by first normalizing the sequences.

Several approaches use the Longest Common Sub Sequence (LCSS) similarity measure [4]. The basic idea is to match some sequences by allowing some elements to be unmatched. The advantage of the LCSS method is that it allows outliers, different scaling functions, and variable sampling rates. Vlachos et al. [29] use the LCSS model to define similarity measures for trajectories. The LCSS model is extended by considering a set of translations and finding the translation that yields the optimal solution to the LCSS problem. Based on the LCSS definition for trajectories, the authors propose two measures, which allow time stretching and translations, respectively. An index structure is also presented, which is based on hierarchical clustering. The use of LCSS in trajectory similarity proposed in [29] has the drawback that it penalizes the points that were marginally outside the matching region by assigning to them a similarity value of zero. To deal with this problem authors propose a non-metric distance function based on the LCSS in conjunction with a sigmoid matching function [28].

In order to overcome the inefficiency of the previously described methods in the presence of noise or obstacles, Chen et al. [10] recently proposed a new distance function called Edit Distance on Real sequences (EDR), based on the Edit Distance (ED) widely used in bio-informatics and speech recognition to measure the similarity between two strings. Their extensive experimental evaluation shows that the proposed distance function is more robust than the Euclidean distance, the DTW and ERP [9] and more accurate than LCSS, especially when dealing with trajectories having Gaussian noise.

Recently, Frentzos et al. [13] proposed a similarity metric (and an approximation method to reduce its calculation cost) in order to support MST search by utilizing R-tree-based trajectory indexing structures.

As previously mentioned, in this paper we follow a different approach by providing a classification of various types of distance operators, each one of them extracting valuable similarity properties based on motion dependant parameters, as such, forming a powerful framework for TD analysis tasks.

3. (Time-relaxed) Spatial Trajectory Similarity Search

In this section and before we define the spatiotemporal similarity between trajectories, we tackle the problem of measuring the spatial similarity of two moving objects as an intermediate step towards the general case. To this effect, we propose a novel distance operator, called *Locality In-between Polylines* (LIP). Intuitively, two moving objects are considered spatially similar when they move close (i.e. their routes approximate each other) at the same place irrespective of time and direction. As such, LIP defines a distance function upon the (projected on the Cartesian plane) routes of the trajectories. The idea is to calculate the area of the shape formed by two 2D polylines, which are the outcome of the above projection (see Figure 1). Formally:

Definition 2: The *Locality In-between Polylines* (LIP) distance between two trajectories Q and S is defined as follows:

$$LIP(Q,S) = \sum_{\forall\, polygon_i} Area_i \cdot w_i \qquad (1)$$

where $polygon_i$ is a member of the set of polygons formulated between intersection points (I_1, I_2, … I_n) created by the overlay of Q and S in 2D plane.

An example of the routes of two trajectories Q and S and the respective areas that contribute in $LIP(Q,S)$ is illustrated in Figure 1.

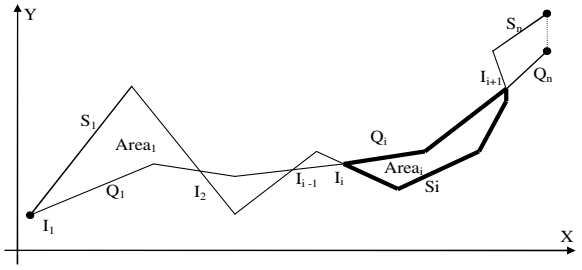


Figure 1: Locality In-between 2D Polylines

Each polygon area is biased by a contribution (weight), which is defined as follows:

Definition 3: Let $Length_Q(I_i, I_{i+1})$ and $Length_S(I_i, I_{i+1})$ be the lengths of the portions of Q and S that participate in the construction of $polygon_i$. The contribution $w_i \in [0,1]$ of $polygon_i$ in $LIP(Q,S)$ is:

$$w_i = \frac{Length_Q(I_i, I_{i+1}) + Length_S(I_i, I_{i+1})}{Length_Q + Length_S} \qquad (2)$$

Note that the numerator of the above equation is the perimeter of the polygon in question, while the denominator is the sum of the total length of the routes.

It is easily implied that LIP is a distance operator that may be intuitively utilized by a user (e.g. posing a query of the form "find similar trajectories that passed three building blocks (<100m^2) away from my route"). A powerful feature of the LIP operator is that it does not only provide a global measure for the similarity of two trajectories. LIP further quantifies the distance among portions of the trajectories. These portions are not statically predefined. For example, the implicit output of the LIP operator is a distance list of the form $LIPgram = \{LIPgram_1, \ldots, LIPgram_i, LIPgram_{i+1}, \ldots, LIPgram_n\}$, where $LIPgram_i = (I_i, I_{i+1}, LIP_i)$, is a triplet that implies the distance LIP_i between points I_i and I_{i+1}. It is a trivial task therefore to perform queries upon the resulted $LIPgram$ so that to find parts of trajectories that diverge or converge and, as such, to cluster subsets of the above trajectories.

The LIP distance function described so far works correctly with pairs of trajectories whose spatial deployment follows, on the average, a stable trend with no dramatic rotations (e.g. like the ones illustrated in Figure 1), or if they are rotating, they do so by turning similarly during their motion (Figure 2a). In the case, for example, where S starts rotating clockwise and Q counterclockwise then the resulting polygons of the LIP algorithm will be self-intersected (non-simple polygons) and, as an outcome the distance measure may be meaningless or even indefinite (Figure 2b). To deal with this issue we provide the following mechanism that makes LIP operator universal in utilization: During traversing polylines Q and S and constructing polygons $polygon_i$, we identify the so-called *bad* segments that violate the *simplicity* property of the under-construction polygons, as they exhibit diverging rotation in contrast to their coupling trajectory. When the algorithm detects these segments, it closes the current polygon by connecting the initial points of the *bad* segments. Then, it invokes the LIP function for the already investigated portions of the polylines that for sure result to simple polygons, and recursively restarts itself with the remaining portions of the initial polylines as parameters.

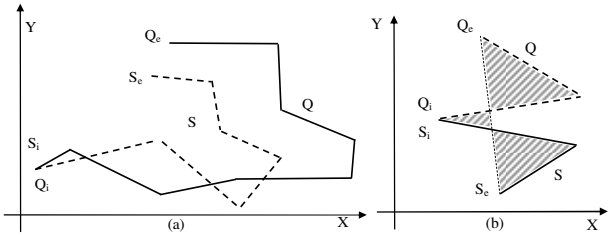


Figure 2: Exceptional cases for LIP operator

Let us now define a simple yet effective criterion that categorizes a pair of segments ($S_i \in S$, $Q_i \in Q$) as either *good* or *bad*. This condition requires that the segment implied between the ending points of the currently

investigated segments does not cross any of the previous segments of Q and S. This is a strict requirement that may lead searching backwards till the beginning of the polylines. Actually, this is not necessary as the effect of the *bad* segments is local and can be treated by testing just a small number of segments for intersection. To this line, the algorithm *GenLIP*, illustrated in Figure 3, searches for *bad* segments and acts as a driver for the invocation of the LIP distance operator.

```
Algorithm GenLIP(Q polyline, S polyline, p int)
1.  WHILE q < Q.LAST AND s < S.LAST
2.    IF intersect(Qq, Ss) THEN
3.      Mark Qq,Ss as 'good' & add them to Q',S'
4.    ELSIF NOT Bad(Q', S', Qq, Ss) THEN
5.      Mark Qq,Ss as 'good' & add them to Q',S'
6.    ELSE
7.      Qq, Ss are marked as 'bad'
8.      FOR k=1 to p
9.        Give p chances to repair LIP criterion
10.     NEXT
11.     IF repairing attempt succeeded THEN
12.       GOTO line 1 with policy-dependant q,s
13.     ELSE
14.       Recover from attempt and GOTO line 17
15.     END IF
16.   NEXT
17.   result = result + LIP(Q', S')
18.   Q = Q - Q'
19.   S = S - S'
20.   RETURN result + GenLIP(Q, S, p)
```

Figure 3: GenLIP Algorithm

More specifically, the algorithm begins traversing the segments of Q and S until the last segment of either Q or S is reached (line 1). At each step it checks whether the two segments are intersected (line 2). Since intersection is an indicator of proximity, the two segments are marked as *good* and are added as tails to two polylines Q' and S', representing the certified portions of Q and S, respectively, that fulfill the aforementioned requirement. The same happens when the test whether the inclusion of the currently investigated segments to Q' and S' characterizes them as *bad*, is negative. In the positive case, these segments are marked as *bad* and subsequently, a chance is given to the next p segments to repair the failure of the LIP criterion. If we succeed to find *good* segments in the following p segments then these are marked and added (as all of the intermediate *bad* segments are) to Q' and S' and the procedure continues in the same way (lines 11-12). If we fail then the procedure first recovers from the previous repairing attempt, and secondly stops the procedure, invokes the LIP operator for Q' and S', and recursively calls the *GenLIP* for the rest of the initial polylines.

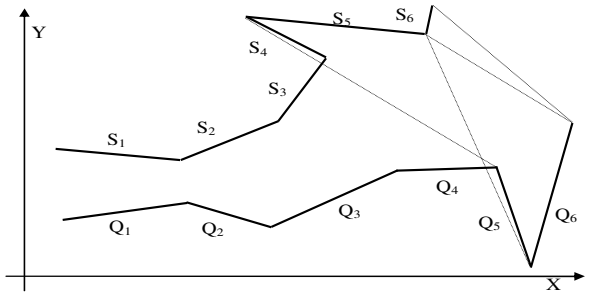


Figure 4: Demonstration of *good* and *bad* segments

To demonstrate the previous discussion, Figure 4 illustrates the routes of two trajectories Q and S for which the *GenLIP* procedure identifies (S_4, Q_4) as a pair of *bad* segments. If no repairing attempt is performed then the LIP distance function will be initially applied for Q' = {Q_1, Q_2, Q_3} and S' = {S_1, S_2, S_3}. On the other hand, if a look-forward repairing attempt is applied (e.g. p=2) then LIP will be applied for Q' = {Q_1, Q_2, Q_3, Q_4, Q_5, Q_6} and S' = {S_1, S_2, S_3, S_4, S_5, S_6}.

4. (Time-aware) Spatiotemporal Trajectory Similarity Search

The LIP distance operator does not include the notion of time as it just compares the projections of moving objects to the Cartesian plane. In this section, we propose a novel distance operator, called *Spatiotemporal LIP* (STLIP), to tackle the problem of measuring the spatio-temporal similarity between two trajectories. Intuitively, two moving objects are considered similar in both space and time when they move close concurrently at the same place. The idea is to utilize LIP and define STLIP as a multiple of LIP. In other words, STLIP is calculated by multiplying LIP by a factor having values greater than 1, implying the temporal distance of the corresponding LIP.

Definition 4: The *Spatiotemporal LIP distance* (STLIP) between two trajectories Q and S is defined as follows:

$$STLIP(Q,S,k,\delta) = \sum_{\forall\, polygon_i} STLIP_i \tag{3}$$

where $STLIP_i$ for $polygon_i$ is defined as:

$$STLIP_i = LIP_i \cdot (1 + k \cdot TLIP_i), \text{ where } k \geq 0\,. \tag{4}$$

Temporal LIP (*TLIP*) is a measure modeling the local temporal distance and participates to the *STLIP* measure by a penalty factor k which represents user's assigned importance to the time-factor.

In order to define the local temporal distance $TLIP_i$ and associate it with the corresponding LIP_i implicitly introduced by the polygons we need to find the timepoints

when Q and S cross each other. Let Qt_I_i be the timepoint when Q passes from intersection point I_i and Qt_I_{i+1} be the timepoint when Q reaches the next intersection point I_{i+1}. Respectively, St_I_i and St_I_{i+1} are the corresponding timepoints for S. These pairs of timepoints define the periods that each point needs to traverse its route from one intersection to the other. Let $Qp_i = [Qt\_I_i, Qt\_I_{i+1})$ and $Sp_i = [St\_I_i, St\_I_{i+1})$ be these periods. Having this in mind, we define *MDI* as the *maximum duration* of the temporal element (the list of time periods, each one representing the lifespan of a particular section of the motion) that is the outcome of the *intersection* between Qp_i and one of the following three alternatives: a) S's temporal projection (Sp_i), b) Sp_i stretched towards future by a temporal interval δ ($Sp_i+\delta$), c) S's temporal projection stretched to the past by δ ($Sp_i-\delta$). To this end, we define the temporal similarity for this LIP_i as formulated in Eq.5.

$$TLIP_i = \left\| 1 - 2\frac{MDI_i(\delta)}{Duration_Q + Duration_S} \right\| \quad (5)$$

MDI has been incorporated in order to support *almost concurrent* movements. This happens by introducing the parameter δ, a time window (tolerance in the past as well as in the future) in which two trajectories, though not moving concurrently, are considered temporally close.

In terms of implementation, STLIP works in the same fashion as LIP does. First, Q and S are projected to the temporal domain to get their lifespans. Subsequently, the intersection of the two lifespans is found and, then, the original trajectories are restricted according to this common temporal element and projected on the spatial 2D plane so as to find the polylines upon which GenLIP will be applied.

5. Variations

Expanding our framework, in this section we describe two variations of the previously presented operators. These variations enhance our distance functions by taking into consideration the rate of change (i.e. speed, acceleration) and directional (i.e. turn) characteristics of the trajectories.

5.1. Speed-pattern based Similarity Search

In order to support the first variation, we introduce a new distance operator, called *Speed-Pattern STLIP* (*SPSTLIP*), to measure the distance between two trajectories that move with similar speed pattern relaxing either space or time features. Two interesting scenarios, which also find realistic applications, are the following:

- *1st scenario*: We are not interested in time dimension; what we know about Q and S is their speed v at different segments *seg* in their route. In this case, the problem is to find the similarity between $Q = \{(seg_{Q,1}, v_{Q,1}), \ldots(seg_{Q,n}, v_{Q,n})\}$ and $S = \{(seg_{S,1}, v_{S,1}), \ldots(seg_{S,n}, v_{S,n})\}$.
- *2nd scenario*: We are not interested in space dimension; what we know about Q and S is their speed v at different periodes of time *per*. The problem here is to find the similarity between $Q = \{(per_{Q,1}, v_{Q,1}), \ldots(per_{Q,n}, v_{Q,n})\}$ and $S = \{(per_{S,1}, v_{S,1}), \ldots(per_{S,n}, v_{S,n})\}$.

These two scenarios are special cases of finding the similarity between two 1D time series and there are well-known methods to perform such tasks. What is more, the above cases imply that the points are moving with constant speed, which is the case of synthetic motions. Our approach is to find similarities of points moving with fluctuated speed and/or acceleration and may be randomly sampled which is the realistic case.

We define *SPSTLIP* by multiplying *STLIP* with a factor greater than 1, which is an indicator of the corresponding Speed-Pattern distance (*SPLIP*). As such, there is a need to define the local *SPLIP* and associate it with the corresponding *LIP*, as well as the respective *TLIP*. We define the local *SPSTLIP* for $polygon_i$ as:

$$SPSTLIP_i = LIP_i \cdot (1 + k \cdot TLIP_i) \cdot (1 + l \cdot SPLIP_i) \quad (6)$$

where $k, l \geq 0$ and

$$SPLIP_i = \frac{\left\| LQ_{Qp_i} - LS_{Qp_i} \right\|}{LQ_{Qp_i}} \quad (7)$$

where LQ_{Qp_i} measures the distance traversed by Q between two intersection points I_i and I_{i+1}, while LS_{Qp_i} is the length of the section of S trajectory restricted at the periods Qp_i during which Q moves from I_i to I_{i+1}. Note that if we omit the $(1 + k \cdot TLIP_i)$ factor in equation 7, the operator becomes time-relaxed as it estimates the distance among trajectories taking into consideration the spatial and speed parameters, irrelevantly to their lifespans.

In order for *SPLIP* to vary between 0 and 1, LS_{Qt_I} should not be greater than $2 \cdot LQ_{Qt\_I}$. In order to enforce it, we partition the TD into subsets according to speed, so that the fastest moving object within each subset is at most α = 2 times the slowest. As such, for objects moving with speed difference higher than α we consider them as non-similar. The overall SPSTLIP between two trajectories Q and S is defined as:

Definition 5: The *Speed-Pattern Spatiotemporal LIP* distance (SPSTLIP) between two trajectories Q and S is defined as follows:

$$SPSTLIP(Q, S, k, l, \delta) = \sum_{\forall\ polygon_i} SPSTLIP_i \quad (8)$$

In contrast to STLIP, the implementation of SPSTLIP further needs to delimit the 3D development of S inside Qp_i and subsequently project the resulting moving point on the spatial plane.

5.2. Directional Similarity Search

A second variation supports similarity of the form: "Find similar trajectories according to their turns". Based on such kind of similarity we could further cluster trajectories that move i.e. initially NW (during period A in place B) and then NE (during period C in place D). We define *Direction Similarity* (DSIM) between Q and S as follows:

Definition 6: The *Directional Similarity* (DSIM) between two trajectories Q and S is defined as follows:

$$DSIM(Q,S) = \sum_{\forall \phi_i} DSIM_{\phi_i} \tag{9}$$

where

$$DSIM_{\phi_i} = \left(\frac{180 - \phi_i}{180}\right) \cdot w_i \tag{10}$$

is the similarity defined between two segments of Q and S characterized by the angle φ_i they form.

This angle ranges from 0° (full similarity, $DSIM = 1$) to 180° (full dissimilarity, $DSIM = 0$) and is reduced by a weight w_i corresponding to the percentage of Q and S trajectory that participate in the similarity. So, if Q_{ϕ_i} and S_{ϕ_i} are the parts of Q and S, respectively, where φ_i is the angle between them, w_i is defined as:

$$w_i = \frac{length(Q_{\phi_i}) + length(S_{\phi_i})}{length(Q) + length(S)} \tag{11}$$

Figure 5 depicts that the angle φ_i formed between two segments of Q and S change to φ_{i+1} whenever either Q or S changes direction. This means that a change occurs at the ending points of each sub-movement of Q and S.

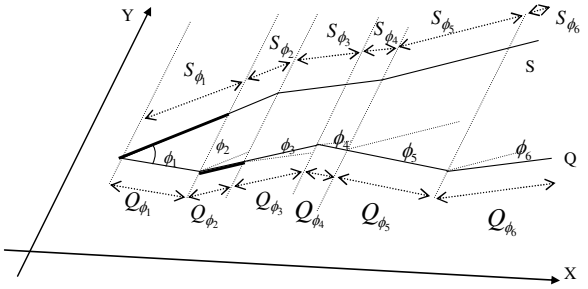


Figure 5: Direction similarity on projected trajectories

Let us now examine how the angle between the directions of two moving objects is calculated. First of all, the direction of a moving object during a period where its motion vector remains constant is the angle formed between the *xx'* axis and the line segment from the initial $i(x_i, y_i)$ to the ending point $e(x_e, y_e)$, measured in degrees ($0° \le direction < 360°$).

In order to calculate the angle between two directed segments and scale this angle between 0° and 180° there is a need for a small examination according to which quadrant Q and S are moving towards. We identify sixteen different cases. Below we examine four of them, where Q is moving towards the 1st quadrant ($0° \le dir(Q) < 90°$) and S is moving towards the 1st, the 2nd, the 3rd, and the 4th quadrant, respectively.

$$\hat{\phi} = angle(Q,S) = \begin{cases} |dir(Q) - dir(S)|, & if\ 0° \le dir(S) < 90° \\ dir(S) - dir(Q), & if\ 90° \le dir(S) < 180° \\ dir(S) - dir(Q), & if\ 180° \le dir(S) < 270° \\ & if\ \hat{\phi} > 180° \\ & \quad \hat{\alpha} = 270° - \hat{\phi} \\ & \quad \hat{\beta} = 90° - \hat{\phi} - \hat{\alpha} \\ & \quad \hat{\phi} = \hat{\phi} - 2 \cdot \hat{\beta} \\ (360° - dir(S)) + dir(Q), & if\ 270° \le dir(S) < 360° \end{cases}$$

The third case is when Q and S are moving towards anti-diametric quadrants. In this case, if the angle is greater than 180° the angle is adjusted to the symmetric angle formed by Q's extended line. The remaining twelve cases follow similar strategies.

The algorithm that implements DSIM starts traversing the coordinates' list of both Q and S projected polylines until it reaches the end of one of them. Two indexes control the access to the points' lists. Only one of the two indexes is advanced at each step depending on which polyline changes its direction. At each step, the algorithm calculates the angle formed between the segments starting from the points currently indexed in the lists (Q_s and S_s) and ending to their subsequent (Q_e and S_e). This occurs after the translation of the segment S_seg (S_s, S_e) towards segment Q_seg (Q_s, Q_e). The next step is to find which of the x-ordinates of the ending points is closer, in terms of Euclidean distance, to the point where the previous change of direction occurred. Finding which is the closest point, makes us aware of where the end of Q_{ϕ_i} or S_{ϕ_i} reside. So if the closest next x is that of Q_e then the end of Q_{ϕ_i} is Q's next point, while the end of S_{ϕ_i} is the projection of Q_e to the currently investigated segment of S. In such case we advance the index of Q coordinates' list, while we keep S_{ϕ_i} so as we clip S_seg by S_{ϕ_i} (bold portion of S_{ϕ_1} in Figure 5) before the calculation of the next angle φ_i. In the same way, if the closest next x is that of Se then the end of S_{ϕ_i} is S's next point, while the end point of Q_{ϕ_i} is the projection of S_e to the currently investigated segment of Q. Similarly, this time we advance the index of S's coordinates list and we clip Q by Q_{ϕ_i} (bold portion of Q_{ϕ_2} in Figure 5). To find the above projection points there are two cases in proportion of whether S_seg and Q_seg are directed towards the same or opposite half-planes defined

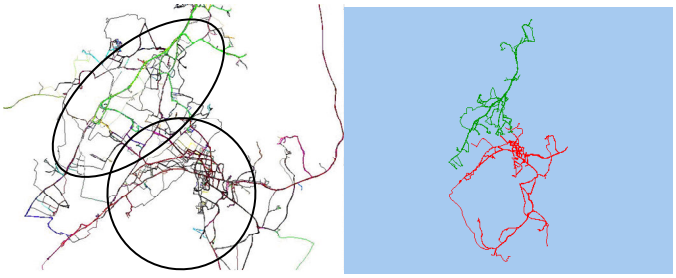
by the yy' axis. In the first case, the x ordinate of the projection point is the closest next x while in the second case the required x ordinate is that of the doubled x of the corresponding start point minus the x ordinate of end point (i.e. $2 \cdot S_s.x - S_e.x$ if closest next x is Q_e). Using interpolation we find the y ordinate of this point.

6. Experimental Evaluation

We implemented the proposed distance operators by incorporating them as query functionality to a TD engine [22], [23]. In the sequel, we describe the datasets and then we present the experimental results evaluating our techniques.

6.1. Datasets

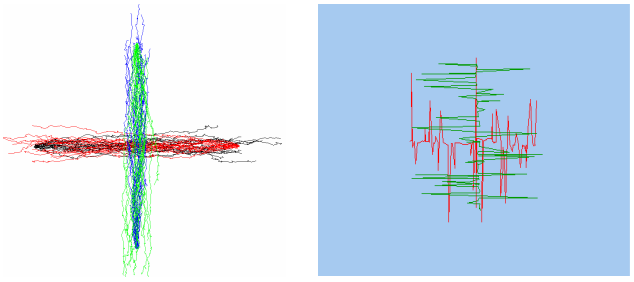
While several real spatial datasets are around for experimental purposes, this is not true for the TD domain. Nevertheless, in this paper, we have used a real-world dataset for experimentation purposes, namely a fleet of trucks available in [25]. The dataset, illustrated in Figure 6a, consists of 276 trajectories. From this original dataset we exported a subset of 20 trajectories belonging to two distinct clusters, with the first 10 following an E→N→W→S pattern and the rest following a N→E pattern (the lower and the upper set, respectively, illustrated in Figure 6b). The semi-automatic procedure that identified the two clusters separated the work space into nine equal square quadrants (NW, N, NE, W, ∗, E, SW, S, SE) and for each one, spatial range queries were performed to find the objects located in each region. Inter-relating the results of range queries, we identified the two clusters to be used in our experimentation.



(a) a fleet of trucks (b) Two identified clusters

Figure 6: Real world trajectory dataset

The experimental study is not limited to real data. We have also used synthetic datasets generated by the GSTD data generator [26], by applying a Gaussian initial distribution, a random movement function and a Gaussian sampling rate. In order to generate four datasets with each (consisting of 20 trajectories with 123 segments per trajectory) heading to a different direction (*N-S-W-E*), the objects' movement was restricted by applying different, non equal extents in each sampling (see Figure 7a).



(a) N-S-W-E datasets *(b) GSTD with noise*

Figure 7: GSTD synthetic trajectory datasets

Furthermore, we have produced a series of datasets by adding different percentages of noise at every sequence of the (*S*, *N*, *W*, *E*) dataset randomly at the 1/3 of its points. The noise was added using the formula $x_{noise} = x + randn * rangeValues$ in the S and N datasets, and $y_{noise} = y + randn * rangeValues$ in the E and W datasets, where *randn* produces a random number chosen from a normal distribution with mean 0 and variance (0.05 – 0.50), and *rangeValues* is the range of values on *X* or *Y* coordinates. Figure 7b illustrates two trajectories included in the W_{20} and S_{20} datasets (i.e., W and S datasets with 20% noise).

6.2. GenLIP Accuracy Experiments

In this section we present experimental results regarding the accuracy of our techniques in classification and clustering tasks. The first experiment was performed using the synthetic *N-S-W-E* datasets and tried to classify routes according to their LIP distance following a methodology initially introduced by Keogh et al. [16]. In this technique, each route is assigned a class label. Then the *leave-one-out* prediction mechanism is applied to each route in turn. That is, the class label of the chosen route is predicted to be the class label of its nearest neighbor, defined based on the given distance. If the prediction is correct, then it is a *hit*; otherwise, it is a *miss*. The *classification error rate* is defined as the ratio of the number of misses to the total number of routes. In this experiment, where we are only interested in spatial similarity, these four datasets form two clusters, one consisting of the *N*, *S* datasets (vertical movement) and the other of the *W* and *E* datasets (horizontal movement in Figure 7a). We apply the *leave-one-out* prediction mechanism based on the GenLIP distance operator for each one of the 80 routes and the accuracy of our approach was absolute. In the sequel, we repeat the same experiment several times but each time utilizing the datasets having noise N_i, S_i, W_i and E_i with i = 5, 10, …, 50. The general

idea is to try to confuse GenLIP by interleaving routes with noise that introduce larger polygons and more *bad* segments than the initial. The GenLIP distance function turns out to be robust enough since it present zero misses up to noise > 25%. Even with more noise the average classification error rate is less than 10 / 80 = 12.5%.

Observing the outcome distances in the previous experiment we noticed that as the noise increases the NN may not yield a misclassification but the *k*-NN have a few misses. So as to stress our evaluation and in order to assess the *inter-cluster* quality we followed the subsequent evaluation procedure. For each route in any of the two clusters we apply (k–1)-NN (k is the number of the routes in each cluster) queries, and we sum all the correct classifications inside the k–1 nearest neighbors. The overall accuracy is defined as the ratio of the number of correct classifications over the total number tests. We further weight the position of each miss in the range of the k–1 neighbors. Applying this procedure to the distances computed previously, we get the results illustrated in Figure 8.

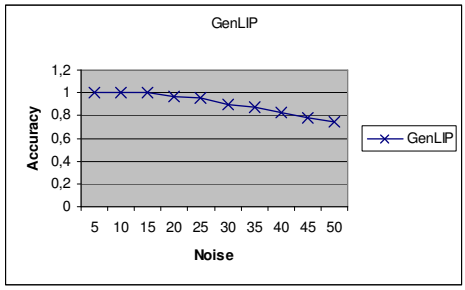


Figure 8: Accuracy of GenLIP against noise

The main conclusion that can be extracted from this chart is that accuracy lowers as the noise increases (which is straightforward) but even for high noise it remains at high levels (accuracy higher than 0.6 for noise up to 50%).

6.3. Experiments on Spatiotemporal Similarity

To demonstrate the STLIP distance operator we conducted experiments using the Trucks dataset. We chose randomly 10 trajectories from the dataset, which were compressed using the TD-TR algorithm described in [20] producing similar but not identical artificial trajectories. We applied the TD-TR compression technique with parameter values of p from 0.1% to 1% of the length of each trajectory (i.e. such parameters values correspond to hundred of meters in our application scenario). The aim was to achieve different values of distance, since increasing TD-TR p parameter produces compressed trajectories with fewer sampled points, while the general sketch of the trajectory remains unaffected. To evaluate this, we formed a 10 cluster data set with 3 trajectories per cluster. We got all possible pairs of clusters and we partitioned them into two clusters applying an agglomerative hierarchical clustering algorithm with the complete linkage criterion (using the CLUTO tool [11]. We sketched the dendrogram of each clustered result to evaluate whether it correctly partitions the trajectories. Figure 9 depicts a correct and a erroneous clustering. STLIP managed to perform very well as it clustered correctly 37 out of the 45 cluster pairs.

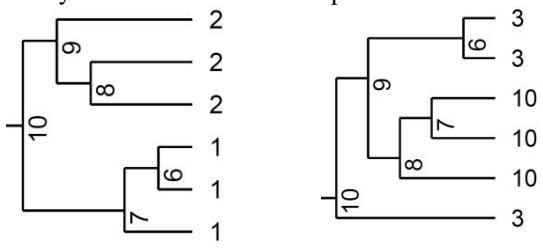


Figure 9: Examples of correct (left) and erroneous (right) clusterings

6.4. Experiments on Directional Similarity

The purpose of the experiment that follows is to use the synthetic *N-S-W-E* datasets and try to classify routes according to their directional similarity DSIM. In this case where direction of routes is of interest, each dataset corresponds to a distinct cluster. For each of the six different pairs of clusters we apply the inter-cluster evaluation procedure and the results are illustrated in Figure 10. Considering the fact that the datasets include routes with quite different lengths and that our approach penalizes moving points with diverging extents the results are satisfactory. What is more, we remind that one of the orientations of our techniques is not only to define global operators but to additionally expose local properties.

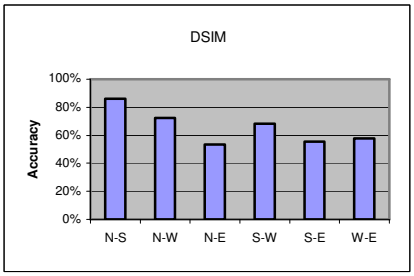


Figure 10: Demonstrating DSIM

7. Conclusions and Future Work

In this paper, we proposed novel distance operators, to address the trajectory similarity search problem that may straightforwardly support knowledge discovery in TD. Clear future work objectives arise from this work. More

specifically, we plan to devise appropriate indexing structures in order to improve overall performance of the operators, while further qualitative evaluation of the operators will be a parallel task. Finally, we plan to study thoroughly the quality of *LIPgrams* and utilize these similarity meta-data patterns so as to perform other mining tasks, like finding most frequent motion patterns.

8. Acknowledgements

Research partially supported by the FP6-14915 IST/FET Project GeoPKDD (Geographic Privacy-aware Knowledge Discovery and Delivery) funded by the European Union and the Pythagoras and Heracletos EPEAEK II Programmes of the Greek Ministry of National Education and Religious Affairs, co-funded by the European Union.

9. References

[1] N. Andrienko, G. Andrienko, N. Pelekis, and S. Spaccapietra, "Basic Concepts of Movement Data", chapter in F. Giannotti and D. Pedreschi (eds.) *Geography, mobility and privacy: A knowledge discovery vision*, Springer, 2007, to appear.

[2] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient Similarity Search in Sequence Databases", Proceedings of *FODO*, 1993.

[3] J. Berndt and J. Clifford, "Finding patterns in time series: A dynamic programming approach", *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT Press, Menlo Park, CA, 1996, 229-248.

[4] B. Bollobas, G. Das, D. Gunopulos, and H. Mannila, "Time-Series Similarity Problems and Well-Separated Geometric Sets", *Nordic Journal of Computing*, 2001.

[5] Y. Cai and R. Ng, "Indexing spatio-temporal trajectories with Chebyshev polynomials", Proceedings of *ACM SIGMOD*, 2004.

[6] K.P. Chan and A.W-C Fu, "Efficient time series matching by Wavelets", Proceedings of *ICDE*, 1999.

[7] T.M. Chan, "A Simple Trapezoid Sweep Algorithm for Reporting Red/Blue Segment Intersections", Proceedings of *CCCG*, 1994.

[8] B. Chazelle and H. Edelsbrunner, "An Optimal Algorithm for Intersecting Line Segments in the Plane", *Journal of the ACM*, 39, 1 (2002), 1-54.

[9] L. Chen and R. Ng, "On the marriage of edit distance and Lp norms", Proceedings of *VLDB*, 2004.

[10] L. Chen, M. Tamer Özsu, and V. Oria, "Robust and Fast Similarity Search for Moving Object Trajectories", Proceedings of *ACM SIGMOD*, 2005.

[11] CLUTO - Family of Data Clustering Software Tools, *http://glaros.dtc.umn.edu/gkhome/views/cluto* (URL valid on December 6, 2006).

[12] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast Subsequence Matching in Time-Series Databases", Proceedings of *ACM SIGMOD*, 1994.

[13] E. Frentzos, K. Gratsias, and Y. Theodoridis, "Indexed-based Most Similar Trajectory Search", Proceedings of *ICDE*, 2007.

[14] GeoPKDD Geographic Privacy-aware Knowledge Discovery, *www.geopkdd.eu* (URL valid on December 6, 2006).

[15] Q. Goldin and C. Kanellakis, "On Similarity Queries for Time-Series Data: Constraint Specification and Implementation", Proceedings of *CP*, 1995.

[16] E. Keogh and S. Kasetty "On the need for time series data mining benchmarks: a survey and empirical demonstration". Proceedings of *SIGKDD*, 2002.

[17] F. Korn, H. Jagadish, and C. Faloutsos, "Efficiently Supporting Ad hoc Queries in Large Datasets of Time Sequences", Proceedings of ACM *SIGMOD*, 1997.

[18] S.-L. Lee, S.-J. Chun, D.-H. Kim, J.-H. Lee, and C.-W. Chung, "Similarity Search for Multidimensional Data Sequences", Proceedings of *ICDE*, 2000.

[19] J. L. Little and Z. Gu, "Video retrieval by spatial and temporal structure of trajectories", Proceedings of *SPIE*, 2001.

[20] N. Meratnia and R.A. de By, "Spatiotemporal Compression Techniques for Moving Point Objects", Proceedings of *EDBT*, 2004.

[21] D. Papadopoulos, G. Kollios, D. Gunopulos, and V.J. Tsotras, "Indexing Mobile Objects on the Plane", Proceedings of *MDDS*, 2002.

[22] N. Pelekis and Y. Theodoridis, "Boosting Location-Based Services with a Moving Object Database Engine", Proceedings of *MobiDE*, 2006.

[23] N. Pelekis, Y. Theodoridis, S. Vosinakis, and T. Panayiotopoulos, "Hermes - A Framework for Location-Based Data Management", Proceedings of *EDBT*, 2006.

[24] D. Rafiei and A. Mendelzon, "Efficient Retrieval of Similar Shapes", *VLDB* Journal, 11, 1 (2002), 17-27.

[25] Y. Theodoridis, "R-tree Portal", *www.rtreeportal.org* (URL valid on December 6, 2006).

[26] Y. Theodoridis, J. R. O. Silva, and M. A. Nascimento, "On the Generation of Spatio-temporal Datasets", Proceedings of *SSD*, 1999.

[27] M. Vlachos, D. Gunopulos, and G. Das, "Rotation Invariant Distance Measures for Trajectories", Proceedings of *SIGKDD*, 2002.

[28] M. Vlachos, D. Gunopulos, and G. Kollios, "Robust Similarity Measures for Mobile Object Trajectories", Proceedings of *MDDS*, 2002.

[29] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering Similar Multidimensional Trajectories", Proceedings of *ICDE*, 2002.

[30] Y. Yanagisawa, J. Akahani, and T. Satoh, "Shape-Based Similarity Query for Trajectory of mobile Objects', Proceedings of *MDM*, 2003.

[31] Y. Sakurai, M. Yoshikawa, and C. Faloutsos, "FTW: Fast Similarity Search under the Time Warping Distance", Proceedings of *PODS*, 2005.

[32] B. Lin, and J. Su, "Shapes Based Trajectory Queries for Moving Objects", Proceedings of *ACM GIS*, 2005.

[33] B-K Yi, H. Jagadish, and C. Faloutsos, "Efficient Retrieval of Similar Time Sequences under Time Warping". Proceedings of *ICDE*, 1998.