Similarity Search in Trajectory Databases

Nikos Pelekis[1,†] *Ioannis Kopanakis*[2] *Gerasimos Marketos*[1] *Irene Ntoutsi*[1] *Gennady Andrienko*[3] *Yannis Theodoridis*[1]

[1]*Dept. of Informatics, Univ. of Piraeus, Greece*
{npelekis, ntoutsi, marketos, ytheod}@unipi.gr

[2]*Tech. Educational Institute of Crete, Greece*
i.kopanakis@emark.teicrete.gr

[3]*Fraunhofer Institute, Germany*
gennady.andrienko@iais.fraunhofer.de

Abstract

Trajectory Database (TD) management is a relatively new topic of database research, which has emerged due to the explosion of mobile devices and positioning technologies. Trajectory similarity search forms an important class of queries in TD with applications in trajectory data analysis and spatiotemporal knowledge discovery. In contrast to related works which make use of generic similarity metrics that virtually ignore the temporal dimension, in this paper we introduce a framework consisting of a set of distance operators based on primitive (space and time) as well as derived parameters of trajectories (speed and direction). The novelty of the approach is not only to provide qualitatively different means to query for similar trajectories, but also to support trajectory clustering and classification mining tasks, which definitely imply a way to quantify the distance between two trajectories. For each of the proposed distance operators we devise highly parametric algorithms, the efficiency of which is evaluated through an extensive experimental study 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 qualitatively different similarity search, as well as 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 [29], [30], [34], [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 temporal 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

† Contact author's address: 80 Karaoli-Dimitriou St., GR-18534 Piraeus, Greece. Tel: +30-2104142449, Fax: +30-2104142264.

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 spatio-temporal 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) or
- 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, or
- Task 4 – *directional similarity*: Find groups of objects that follow a given direction pattern (e.g. NE during the first half of the route and subsequently W), concurrently or not.

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 also 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, in general, 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. The complexity of the algorithms proposed is given and proved lower compared to existing approaches.
- 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 sequence 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.

As already mentioned, 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 [31]. 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 [28]. Sakurai et al. [32] 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 [33] 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], [30] 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 [30] has the drawback that it penalizes the points that were marginally outside the matching region by assigning to them zero similarity value. To deal with this problem, the authors propose a non-metric distance function based on the LCSS in conjunction with a sigmoid matching function [29].

In order to overcome the inefficiency of the previously described methods in the presence of noise or obstacles, Chen et al. [10] 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, 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. 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.

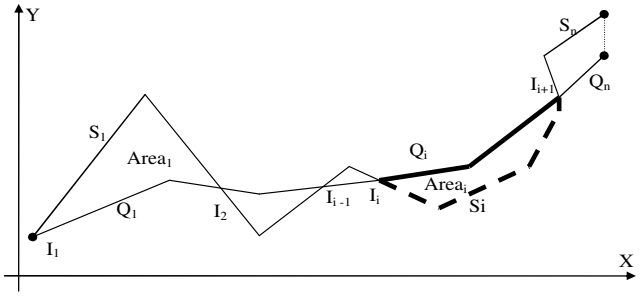


Figure 1: Locality In-between 2D Polylines

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

In summary, the procedure for computing LIP(Q,S) distance iteratively detects the above discussed simple polygons, whereas each polygon is constructed as a list of points. This list is separated into two sets. The first set consists of points that reside on Q (solid line in Figure 1), while the second set of points belongs to S (dashed line). The algorithm follows forwardly the solid line, adding points from Q to the current polygon, and when finding an intersection point, follows backwardly the dashed line, adding points from S to the current polygon until the previous intersection point is visited. Intersection points I_i between Q and S are computed and stored once in a priority queue PQ. Together with the priority queue, a positioning array (PA) is maintained keeping the line segments of both Q and S where upon the intersection points reside.

Regarding the complexity of LIP(Q,S) computation, we note the following: Assuming $N = |Q| + |S|$ line segments and K intersection points, finding intersection points I_i between Q and S (line 1) corresponds to the red-blue intersection problem [7], which can be solved in $O(N\log N+K)$ time using $O(N+K)$ space [8]. PQ stores K elements (equals to the number of the intersection points) and for each one, 2 positions are pointed out in PA. So $O(K)$ space is required for maintaining the two structures. Since the number of polygons defined over the K intersection points is K, $O(K)$ time is required to calculate the areas of polygons. In total, LIP(Q,S) computation is $O(N\log N)$ time and $O(N)$ space complexity, assuming that K and N are of the same order.

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 (say, <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; furthermore it 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 (e.g. like the ones illustrated in Figure 1) with no dramatic rotations, 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.

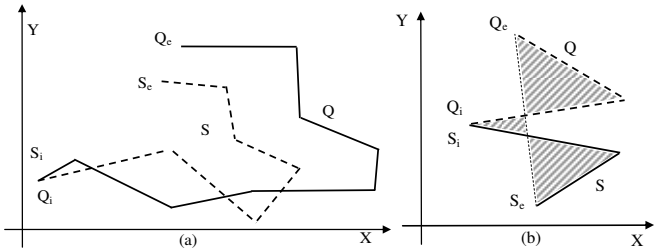


Figure 2: Exceptional cases for LIP operator

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*.

Definition 4: The *LIP criterion* imposes that the segment implied between the ending points of the currently investigated segments S_i and Q_i crosses none of the previous segments of Q and S.

This is a strict requirement that may lead searching backwards until 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 only 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.  q=s=1 // q, s are indices to Q's & S's segments
2.  WHILE q < Q.LAST AND s < S.LAST
3.     IF intersect(Qq, Ss) THEN
4.       Mark Qq,Ss as 'good' & add them to Q',S'
5.     ELSIF NOT Bad(Q', S', Qq, Ss) THEN
6.       Mark Qq,Ss as 'good' & add them to Q',S'
7.     ELSE
8.       Mark Qq,Ss as 'bad'
9.       FOR k=1 to p // look the next p segments
10.        Apply the LIP criterion to Qq + k, Ss + k
11.        IF NOT Bad(Q', S', Qq + k, Ss + k) THEN
12.          Continue at line 2 with Qq + k + 1,Ss + k + 1
13.        END IF
14.      NEXT
15.      IF all p segments failed the criterion THEN
16.        Remove them from Q' & S' and GOTO line 20
17.      END IF
18.    END IF
19.  NEXT
20.  result = result + LIP(Q', S')
21.  Q = Q - Q'
22.  S = S - S'
23.  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 2). At each step it checks whether the two segments are intersected (line 3). 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 (p is a parameter of GenLIP algorithm). 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 9-14). If we fail then the procedure ignores the previous processing by removing p segments from Q' and S' (lines 15-17), invokes the LIP operator for the latter (line 20), and recursively calls the *GenLIP* for the rest of the initial polylines (lines 21-23).

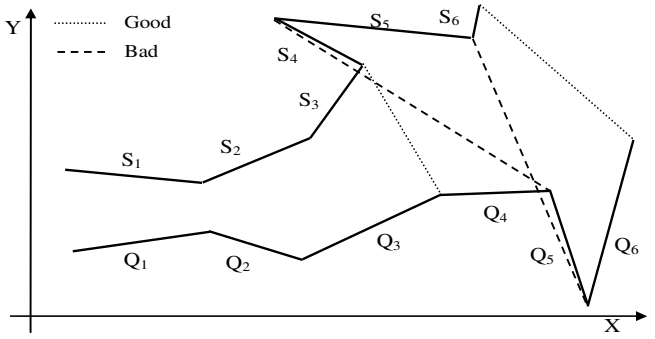


Figure 4: Demonstration of *good* and *bad* segments

To demonstrate the previous discussion, Figure 4 illustrates the routes of two trajectories Q and S for which the *GenLIP* procedure marks (S_i, Q_i) for $i \leq 3$ as pairs of *good* segments and subsequently 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 $S' = \{S_1, S_2, S_3, S_4, S_5, S_6\}$.

Regarding the complexity of GenLIP algorithm, $O(\max(|Q|,|S|))\cdot O(p^2)$ is the cost of the while loop (lines 1-16), where p is the maximum number of segments to be examined in order for GenLIP to decide whether the under examination pair of segments is either 'good' or 'bad', and $O(N\log N)$ is the cost of LIP function as discussed earlier. In total, GenLIP algorithm is $O(L\cdot N\log N)$ time complexity (assuming p is a very small number, i.e. $p << N$), where L is the depth of the recursion (line 23). In the worst case, L is order of N (all pairs of segments are marked as 'bad') but in this case the cost of LIP function is negligible. Overall, we can safely argue that GenLIP Algorithm is $O(N\log N)$ time complexity, which is also demonstrated in the experimental study (Section 6).

4. (Time-aware) Spatiotemporal Trajectory Similarity Search

The LIP distance operator does not take time into consideration since it compares the projections of moving objects to the Cartesian plane. In this section, we extend LIP and we propose a novel distance operator, called *Spatiotemporal LIP* (STLIP), that measures the spatio-temporal similarity between two trajectories. Intuitively, two moving objects are considered similar in both space and time when they move close at the same time.

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

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

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

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

In other words, we define STLIP as a multiple of LIP (by a factor greater than 1), implying the temporal distance of the corresponding LIP. *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) temporal projection of S (Sp_i), b) Sp_i stretched towards future by a temporal interval δ ($Sp_i+\delta$), c) Sp_i stretched to the past by δ ($Sp_i-\delta$). To this end, we define the temporal similarity as formulated in Eq.5.

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

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. Considering time complexity, the cost of calculating STLIP is again $O(N\log N)$ since LIP is the dominant factor.

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:

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

where $k, l \geq 0$ and

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

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}. Intuitively, $SPLIP_i$ is a local index of speed similarity as it quantifies how long the route traversed by S is, in comparison with that of Q, during the interval that Q needs to go from one intersection point to another.

Note that if we omit the $(1+k\cdot TLIP_i)$ factor in Eq. 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 $\alpha = 2$ times the slowest. Thus, for objects moving with speed difference higher than α we consider them as non-similar. The overall SPSTLIP between Q and S is defined as:

Definition 6: The *Speed-Pattern Spatiotemporal LIP* distance (SPSTLIP) between two trajectories Q and S is defined as follows:

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

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. Like STLIP, SPSTLIP presents $O(N\log N)$ time complexity.

5.2. Directional Similarity Search

A second variation supports similarity of the form: "Find similar trajectories according to their heading". Based on such kind of similarity we could further cluster trajectories that move e.g. initially NW (during period A in place B) and then NE (during period C in place D).

Definition 7: *Directional Distance* (DDIST) between two trajectories Q and S is defined as follows:

$$DDIST(Q,S) = \sum_{\forall \phi_i} DDIST_{\phi_i} \qquad (9)$$

where

$$DDIST_{\phi_i} = \frac{\phi_i}{\pi}\cdot w_{\phi_i} \qquad (10)$$

is the distance defined between two segments of Q and S characterized by the angle φ_i they form. This angle ranges from 0 (full similarity, $DDIST = 0$) to π (full dissimilarity, $DDIST = 1$). The weight w_{ϕ_i} corresponds to the percentage of Q and S trajectory that participate in the similarity. So, assuming that Q_{ϕ_i} and S_{ϕ_i} are the parts of Q and S, respectively, where φ_i is the angle between them, w_{ϕ_i} is defined as:

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

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 segment of Q and S.

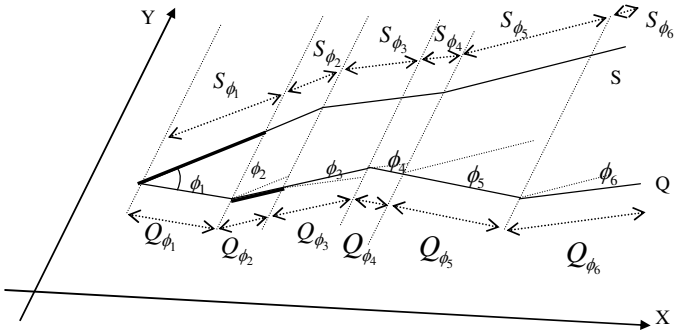


Figure 5: Direction similarity on projected trajectories

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 radians ($0 \le dir(Q), dir(S) < 2\pi$).

In order to calculate the angle between two directed segments and scale this angle between 0 and π 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 ($0 \le dir(Q) < \pi/2$) and S is moving towards the 1st, the 2nd, the 3rd, and the 4th quadrant, respectively.

$$\hat{\phi} = angle(Q,S) = \begin{cases} |dir(Q) - dir(S)|, & if\ 0 \le dir(S) < \pi/2 \\ dir(S) - dir(Q), & if\ \pi/2 \le dir(S) < \pi \\ dir(S) - dir(Q), & if\ \pi \le dir(S) < 3\pi/2 \\ & if\ dir(S) - dir(Q) > \pi \\ & \quad \hat{\alpha} = 3\pi/2 - dir(S) \\ & \quad \hat{\beta} = \pi/2 - dir(Q) - \hat{\alpha} \\ & \quad \hat{\phi} = dir(S) - dir(Q) - 2\cdot\hat{\beta} \\ (2\pi - dir(S)) + dir(Q), & if\ 3\pi/2 \le dir(S) < 2\pi \end{cases}$$

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

The procedure for computing DDIST 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 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) is calculated. This occurs

after the translation of the segment (S_s, S_e) towards segment (Q_s, Q_e), hereafter called *S_seg* and *Q_seg*, respectively. 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. We proceed similarly, in the case where the closest next x is that of S_e (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.

When the previous procedure reaches the end of either Q or S, which is the case where the compared trajectories have different lengths, the operator continues similarly with the remaining portion of the longer trajectory against the extension of the shorter one towards its average direction.

DDIST is time-relaxed in the sense that it does not consider the temporal information of trajectories. In the sequel, we study the time-aware case of directional similarity.

Definition 8: *Temporal Directional Distance* (TDDIST) between Q and S is defined as follows:

$$TDDIST(Q,S) = \frac{\sum_{\forall Q_i} DDIST_{\phi_i}(Q_i, S_{Q_i})}{\#i} \qquad (12)$$

where $DDIST_{\phi_i}(Q_i, S_{Q_i})$ is the DDIST between Q_i, the projection of each 3DLS of Q in the Cartesian plane, and S_{Q_i}, the corresponding projection of S sub-motion restricted at the period that Q needs to traverse Q_i. In other words, $DDIST_{\phi_i}(Q_i, S_{Q_i})$ is an indicator of how similarly S follows the direction of Q_i. The overall TDDIST is defined as the average DDIST introduced by each pair (Q_i, S_{Q_i}). To implement this function we simply restrict the lifespans of Q and S inside the temporal element where there is concurrent development. Subsequently, for each segment Q_i we project the S restricted to the time period of Q_i motion and invoke the DDIST operator for the two sections of Q and S trajectories.

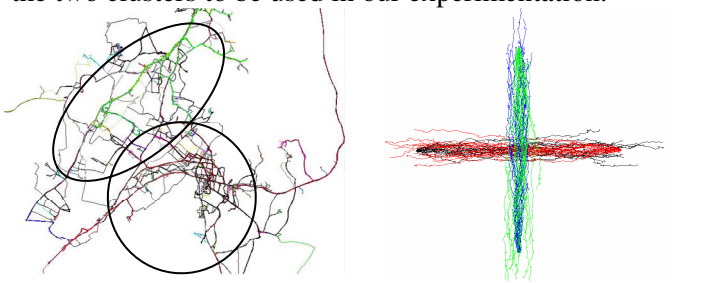
Considering time complexity, in order to calculate directional distance (either DDIST or TDDIST), a single pass over the segments of the two trajectories is required; hence the cost is $O(\max(|Q|,|S|))$.

6. Experimental Evaluation

We implemented the proposed distance operators by incorporating them as query functionality into a TD engine [22], [23]. In the sequel, we present the datasets and then the experimental results evaluating our approach. The experiments were run on a PC with Intel Core Duo at 2 GHz, 2 GB RAM and 100 GB hard disk.

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 [26]. 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 6a). 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 and two identified clusters *(b) N-S-W-E GSTD datasets*

Figure 6: Real vs. synthetic trajectory datasets

The experimental study is not limited to real data. We have also used synthetic datasets generated by the GSTD data generator [27], by applying a Gaussian initial distribution, a random movement function and a Gaussian sampling rate. In order to generate four datasets (each consisting of 20 trajectories with 123 segments per trajectory) heading to different directions (*N-S-W-E*, respectively), the objects' movement was restricted by applying different, non equal extents in each sampling (see Figure 6b).

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. For instance, W_{20} and S_{20} datasets are generated from *W* and *S* datasets with 20% noise.

We have also generated three variations of the *E* and *W* datasets (*E*2, *E*3, *E*4, *W*2, *W*3, *W*4) in terms of their sampling rate. More specifically, for each one of these trajectories we have produced their counterparts that have the same spatial extend and multiple number of segments. For instance, *E*2, *W*2 datasets have more or less twice the number of segments per pair of trajectories in comparison to *E* and *W* datasets.

6.2. GenLIP Processing Time

Here we present the results of an experimental study on the GenLIP distance function as it is the one with the higher computational cost. Initially, we utilize the *E* and *W* datasets and, for each route of the one, we compute the GenLIP distance with each route of the other (hence we measure the total time to perform 20x20 calculations). The experiment is performed four times (for different values of p = 0, 2, 4, 8). This set of experiments is repeated three times, using the (*E*2, *W*2) (*E*3, *W*3) (*E*4, *W*4) datasets, respectively. Figure 7 shows the average processing time (CPU plus I/O time) between a pair of routes.

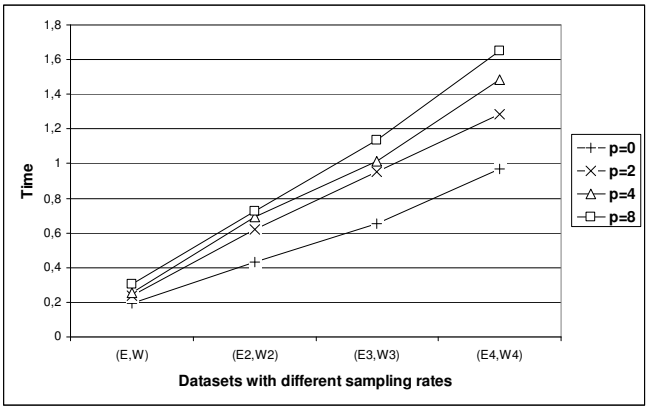


Figure 7: Average processing times for GenLIP function

As can be observed from the experimental results, the running times follow a curve that proves the complexity analysis of the GenLIP distance function. Intuitively this occurs as the number of intersections between the polylines is not minimal as the routes move towards the same direction. We also notice that the p value in the behavior of the GenLIP is almost negligible.

6.3. GenLIP Quality Experiments

In this section we present experimental results regarding the quality of GenLIP when used for 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, 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 consisting of the *W* and *E* datasets (horizontal movement). 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 (i.e. 80/80 hits). In the sequel, we repeat the same experiment several times using the 'noisy' 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 25% noise. Even adding more noise, the average classification error rate does not exceed 12.5% (i.e. 10 / 80 misses).

It is interesting to notice that as the noise increases the NN may not yield a misclassification though the *k*-NN may 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 of 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, where for each policy we present only the average accuracy for different values of p.

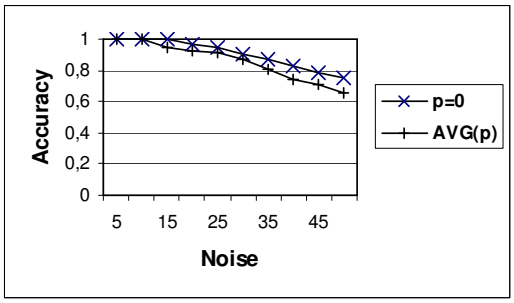


Figure 8: Accuracy of GenLIP against noise

The main conclusion that can be drawn from this chart is that accuracy lowers as the noise increases (which is a straightforward behavior) but even for high noise it remains at high levels (accuracy higher than 0.6 for noise up to 50%). Secondly, as expected, by increasing the number of p we do not succeed better accuracy. This is because parameter p is a simple way to produce bigger and more intuitive *LIPgrams*, or in other words, a mean to overlook jerky movements (or noise), which temporarily spoil the LIP criterion leading to unnecessary recursions.

6.4. Experiments on Spatiotemporal Similarity

We consider that the core advantage of the current approach relies on the integration of various types of novel distance operators under a flexible framework. Furthermore, although starting from different baselines, it is only STLIP among the proposed operators that can be compared with related work. To demonstrate its applicability and robustness we compare it with a state-of-the-art similarity measure, namely EDR [10]. (LCSS [30] and DTW [3] are not included in the experimental study, since EDR has been shown to outperform them in [10]).

We conducted experiments using the Trucks dataset. Following [10] and in order to get the best clustering results we normalized the dataset. Although not important in the experiment setting, we run STLIP giving penalty $k = 2$, while we set the temporal interval δ to 10 seconds.

We chose 10 trajectories randomly out of 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 in the set {0.02%, 0.05%, 0.1%, 0.15%, 0.2%, 1%, 2%, 5%, 10%} of the length of each trajectory. We should note that even for small TD-TR parameter values the effect of the compression is significant. For instance, for p=0.1% the compressed trajectory has almost half number of 3DLS in contrast to the original trajectory. In other words, increasing TD-TR p parameter produces compressed trajectories with fewer sampled points, while the general sketch of the trajectory remains unaffected. For a fair benchmark, we also improved EDR by interleaving samples in the compressed trajectory with linear interpolation at the timestamps the checked dataset trajectory was sampled.

We formed 10 datasets of 10 clusters each, one for each trajectory, where one dataset is different from the other only in the number of trajectories per cluster. For example, clusters of the first dataset (i.e. p=0.02%) consist of the original trajectory and the corresponding compressed with the lowest p value (i.e. p=0.02%). Clusters of the second dataset consist of the original trajectory and the two corresponding compressed with the two lowest p values, and so on. For each dataset, we got all possible pairs of clusters (i.e., 45 cluster pairs) 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 an erroneous clustering for the second dataset. A dendrogram is considered erroneous if there is even one node in the hierarchy, except the root, that joins items belonging to different classes (i.e. the right dendrogram is erroneous due to the ninth node that joins nodes from different classes).

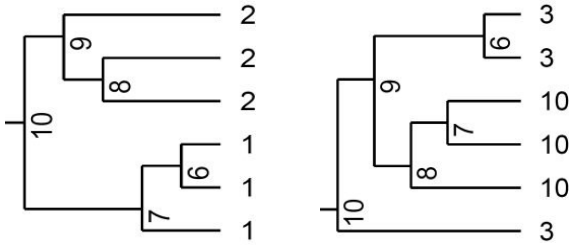


Figure 9: Examples of correct (left) and erroneous (right) clusterings

The results of the experiments are illustrated in Figure 10. Obviously, STLIP outperforms EDR. Actually, STLIP correctly partitions the dataset into two clusters even we add trajectories that were compressed with high values of p of their length. A straightforward conclusion that can be made by observing the dendrograms produced by EDR is that they are able to correctly identify the NN of the query trajectory (i.e. this is the case of node six in the right dendrogram of Figure 9); or even to temporarily/initially identify the correct cluster at the lower levels of the dendrogram (i.e. see node eight in the right dendrogram of Figure 9); however, at the end it fails in detecting similar trajectories of almost the same length which have been sampled differently. This is because it attempts to match the sampled positions one by one, which is not the usual case in real world case studies.

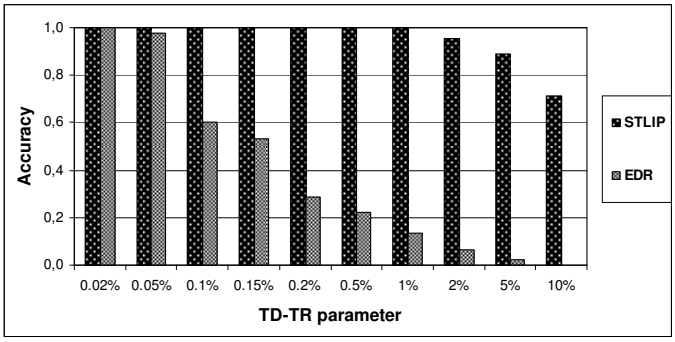


Figure 10: Accuracy against bigger clusters with more dissimilar trajectories

6.5. Experiments on Directional Similarity

The purpose of this section is to evaluate the operators focusing on the directional similarity between trajectories, namely the TDDIST and the time-relaxed version DDIST. We perform the same experiment as in the case of the previous subsection by selecting a subset of the produced datasets. Intuitively, compressed versions of a trajectory follow similar direction patterns, and as such they form a cluster that, both TDDIST and DDIST will be able to identify against a corresponding set of compressed versions of another trajectory. The results of the experiments depicted in Figure 11 prove experimentally our intuition, while further shows the efficiency of the operators. No conclusion can be made about the accuracy of the two operators against each other, since the TDDIST operator first restricts trajectories to their common lifespan, which means that in case of trajectories with different lengths, it does not compare the same portions on which DDIST operates.

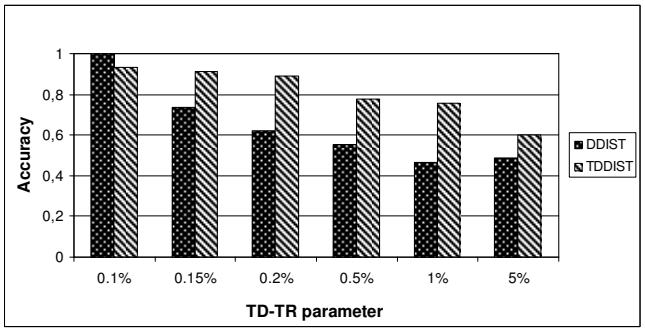


Figure 11: Demonstrating TDDIST and DDIST

7. Conclusions and Future Work

In this paper, we proposed novel distance operators, to address different versions of the so-called trajectory similarity search problem that may straightforwardly support knowledge discovery in TD. To the best of our knowledge, this is the first work that decomposes the problem into different types of similarity queries based on various motion parameters of the trajectories. The synthesis of the operators under a unified trajectory management framework provides functionality so far unmatched in the literature. The efficiency and robustness of the operators have been proved experimentally by performing clustering and classification tasks to both real and synthetic trajectory datasets.

Clear future work objectives arise from this work. More specifically, we plan to devise appropriate indexing structures in order to improve the overall performance of the operators, while further qualitative evaluation of the operators will be a parallel task. Additionally, we intend 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. Finally, we will investigate extending our techniques to address the problem of similariry search for trajectories restricted in spatial networks [25].

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 EPEAEK II Programm 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 February 12, 2007).

[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 and Delivery project, *www.geopkdd.eu* (URL valid on February 12, 2007).

[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] E. Tiakas, A. N. Papadopoulos, A. Nanopoulos, Y. Manolopoulos, D. Stojanovic and S. Djordjevic-Kajan, “Trajectory Similarity Search in Spatial Networks”, Proceedings of *IDEAS* 2006.
[26] Y. Theodoridis, “R-tree Portal”, *www.rtreeportal.org* (URL valid on February 12, 2007).
[27] Y. Theodoridis, J. R. O. Silva, and M. A. Nascimento, “On the Generation of Spatio-temporal Datasets”, Proceedings of *SSD*, 1999.
[28] M. Vlachos, D. Gunopulos, and G. Das, “Rotation Invariant Distance Measures for Trajectories”, Proceedings of *SIGKDD*, 2002.
[29] M. Vlachos, D. Gunopulos, and G. Kollios, “Robust Similarity Measures for Mobile Object Trajectories”, Proceedings of *MDDS*, 2002.
[30] M. Vlachos, G. Kollios, and D. Gunopulos, “Discovering Similar Multidimensional Trajectories”, Proceedings of *ICDE*, 2002.
[31] Y. Yanagisawa, J. Akahani, and T. Satoh, “Shape-Based Similarity Query for Trajectory of mobile Objects’, Proceedings of *MDM*, 2003.
[32] Y. Sakurai, M. Yoshikawa, and C. Faloutsos, “FTW: Fast Similarity Search under the Time Warping Distance”, Proceedings of *PODS*, 2005.
[33] B. Lin, and J. Su, “Shapes Based Trajectory Queries for Moving Objects”, Proceedings of *ACM GIS*, 2005.
[34] B-K Yi, H. Jagadish, and C. Faloutsos, “Efficient Retrieval of Similar Time Sequences under Time Warping”. Proceedings of *ICDE*, 1998.