Index-based Most Similar Trajectory Search

Elias Frentzos, Kostas Gratsias, Yannis Theodoridis

Laboratory of Information Systems

Department of Informatics

University of Piraeus

Hellas

Technical Report Series

UNIPI-ISL-TR-2006-01

November 2006

Index-based Most Similar Trajectory Search

Elias Frentzos, Kostas Gratsias, Yannis Theodoridis
Department of Informatics, University of Piraeus, Greece
{efrentzo, gratsias, ytheod}@unipi.gr

Abstract

The problem of trajectory similarity in moving object databases is a relatively new topic in the spatial and spatiotemporal database literature. Existing work focuses on the spatial notion of similarity ignoring the temporal dimension of trajectories and disregarding the presence of a general-purpose spatiotemporal index. In this work, we address the issue of spatiotemporal trajectory similarity search by defining a similarity metric, proposing an efficient approximation method to reduce its calculation cost, and developing novel metrics and heuristics to support k-most-similar-trajectory search in spatiotemporal databases exploiting on existing R-tree-like structures that are already found there to support more traditional queries. Our experimental study, based on real and synthetic datasets, verifies that the proposed similarity metric efficiently retrieves spatiotemporally similar trajectories in cases where related work fails, while at the same time the proposed algorithm is shown to be efficient and highly scalable.

1. Introduction

With the rapid growth of wireless communications and positioning technologies, the concept of Moving Object Databases (MOD) has been in the core of the spatial and spatiotemporal database research. An interesting type of query that is useful in MOD search is the so-called *trajectory similarity problem*, which aims to find 'similar' trajectories of moving objects.

To illustrate the problem, consider the following example. Suppose that the metro network of a city has been recently extended, initiating a new transportation line, in view of providing transport services to a major part of the residents of the city suburbs. This metro network extension requires the re-designing of the existing transportation network (buses, tram, trolley-buses, etc.). Experts in the field would be assisted if they could pose queries about the similarity between the trajectories of the existing transport means and the new metro line. As such, they would be able, for example, to change the timetable of a bus line, if it matches in a certain day with the timetable of the new metro line, or even abort it. To handle such queries efficiently, MOD systems should include methods for answering the so-called *Most-Similar-Trajectory* (MST) search also discussed in [14].

Trajectory similarity search is a relatively new topic in the literature; the majority of the methods proposed so far are based on either the context of time series analysis or the Longest Common SubSequence (LCSS) model [20] and the recently proposed Edit Distance on Real Sequence (EDR) [5]. However, all these methods have the main drawback that they either ignore the time dimension of the movement, therefore calculating the spatial (and not the spatiotemporal) similarity between the trajectories, or assume that the trajectories are of the same length and have the same sampling rate. To exemplify the problem derived when different sampling rates are present, consider Figure 1 presenting two trajectories T and Q with their position being sampled in different rates.

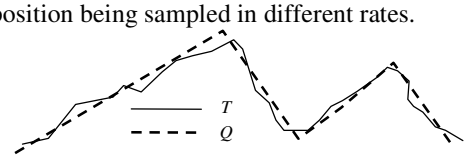

Figure 1. Trajectories with different sampling rates

While Q and T sample their position 4 and 32 times respectively, they have approximately the same length traversing through the same area. Though the two trajectories are obviously similar, methods based on the LCSS or the EDR model cannot detect this kind of similarity since they try to match trajectory sampled positions one by one, which clearly does not happen in the above (real world) example. Moreover, the majority of the proposed approaches exploit specialized index structures in order to prune the search space and retrieve the most similar to a query trajectory.

The challenge accepted in this paper, is to efficiently support the k-MST search in MODs storing historical trajectory information, exploiting existing R-tree-like structures which can also be used to support other types of queries as discussed in [12] and [6]. Our main contributions are outlined as follows:

- We define a dissimilarity metric (*DISSIM*) for the measurement of the spatiotemporal dissimilarity between two trajectories and we propose an efficient approximation method to overcome its costly calculation.
- Adopting the *MINDIST* calculation between a trajectory and an index node proposed in [6], we

present a set of novel metrics, and provide several lemmas, to be used for pruning.

- Using the above metrics, we propose a best-first query processing algorithm to perform *k*-MST search on R-tree-like structures.
- We conduct a comprehensive set of experiments over synthetic and real datasets demonstrating that the proposed similarity metric efficiently retrieves spatiotemporally similar trajectories in cases where related work fail, and the proposed MST search algorithm is highly scalable and efficient, in terms of execution time and pruned space.

We have to point out that this algorithm does not require any dedicated index structure and can be directly applied to any member of the R-tree family used to index trajectories. To the best of our knowledge, this is the first work providing techniques for a spatiotemporal index to support both classical range, topological and similarity based queries. The rest of the paper is structured as follows. Related work is discussed in Section 2, while Section 3 introduces the dissimilarity metric, as well as the set of metrics that support our pruning strategies. Section 4 describes in detail the best-first query processing algorithm to perform MST search over historical trajectory information. Section 5 presents the results of our experimental study and Section 6 concludes the paper giving hints for future work.

2. Related Work

Similarity search has been well studied in the time series analysis domain. As a measure of approximate matching, Agrawal et al. [1] proposed the utilization of the Discrete Fourier Transformation (DFT). An alternative time series matching technique through dimension reduction was proposed by Chan and Fu [3], using the Discrete Wavelet Transformation (DWT). In order to compare sequences with different lengths, Berndt and Clifford [2] used the Dynamic Time Warping (DTW) technique that allowed sequences to be stretched along the time axis to minimize the distance between sequences. Although DTW incurred a heavy computation cost, it was more robust against noise.

In [21] an indexing method for processing shape-based similarity queries for trajectory databases was presented. The proposed method was based on Euclidean Distance. However it could be applied only on trajectories with same lengths being valid during the same time interval. Cai and Ng [4] proposed the utilization of Chebyshev polynomials for approximating and indexing trajectories for similarity matching purposes. Still, this method suffered from the requirement that the trajectories should be of the same length (in terms of the number of spatiotemporal points that are composed of).

Vlachos et al. [19] presented a distance measure that allowed to find similar trajectories under translation, scaling and rotational transformations. The first step of their method was the mapping of each trajectory to a trajectory in a rotation invariant space. For the calculation of the distance between two trajectories in the new rotation invariant space, the DTW technique was utilized.

Sakurai et al. [13] proposed an improved version of DTW, the Fast search method for Dynamic Time Warping (FTW), based on a new lower bounding measure for the approximation of the time warping distance. They proved that FTW could prune a significant portion of the search space, leading to a significant reduction of the search cost.

Recently, Lin and Su [10] have studied the time independent similarity search problem of moving object trajectories. The "one way distance" (OWD) function is introduced for comparing the spatial shapes of trajectories along with appropriate algorithms for computing OWD. Their experimental study shows that the adoption of OWD function outperforms DTW algorithm in terms of precision and performance.

Several approaches are based on the Longest Common Sub Sequence (LCSS) similarity measure. LCSS measure matches two sequences by allowing them to stretch, without rearranging, the sequence of the elements, but allowing some elements to be unmatched (which is the main advantage of the LCSS measure compared with Euclidean Distance and DTW). Therefore, LCSS can efficiently handle outliers and different scaling factors. Vlachos et al. [20] adopted the utilization of the LCSS method. Introducing two similarity measures allowing time stretching and translations respectively, the authors proposed non-metric similarity functions, which were very robust to the presence of noise and provided an intuitive notion of similarity between trajectories by giving more weight to the similar portions of the trajectories. Moreover, an efficient index structure (based on hierarchical clustering) for similarity queries was presented. However, as will be shown in the experimental study, the proposed method suffers when trajectories have different sampling rates.

In [5] a distance function, called Edit Distance on Real Sequences (EDR), was introduced. This distance function, based on edit distance, was shown to be more robust than DTW and LCSS over trajectories with noise. The efficiency of this distance function was improved by the application of three pruning strategies, which reduced the respective computational cost in terms of computations between the query and data trajectories without introducing false dismissals. On the other hand, same as LCSS, EDR determines spatial similarity only, ignoring time, while trajectories with different sampling rates cannot be handled efficiently, as it will be shown in the experimental study. Moreover, both [20] and [5] propose the employment of dedicated indexes to prune the search space so as to efficiently support *k*-MST search.

Recently, Keogh et al. [9] presented an algorithm (based on the LB_Keogh function introduced in [8]), which dramatically reduced the time complexity of the

calculation of the Euclidean Distance measure. This speed up was further achieved by allowing indexing. However, the above algorithm, which was generalized to other distance measures, such as DTW and LCSS, could be applied only to 2D shapes.

Acknowledging the contributions of the above proposals, in the sequel we propose novel metrics and algorithms for trajectory similarity search on R-tree-like structures.

Table 1. Table of notations

Notation	Description
T, Q	an indexed and a query trajectory
T_k, Q_k	the k^{th} line segment of T or Q
t_k	a timestamp
$D_{Q,T}(t)$	function of distance in time between Q and T
a, b, c	factors of $D_{Q,T}(t)$ trinomial
$E_{Q,T}$	calculation error of the dissimilarity between trajectories
D	distance between trajectories
V	relative speed between moving objects
N	R-tree node
$MINDIST(Q,N)$	minimum distance between Q and N
V_{max}	the sum of the maximum speed of indexed trajectories plus the maximum speed of the query trajectory
S_R	the set of line segments already retrieved from the index
S_C	the set of trajectories with line segments already retrieved from the index but not yet fully completed inside the given time period.

3. Metrics for k-MST Search

In this section we will define the notion of spatiotemporal dissimilarity used in the rest of the paper followed by a series of metrics and heuristics used in our algorithms for MST Search (Table 1 presents the notations used in the rest of the section). As already mentioned, existing work in the domain of trajectory similarity search, either ignores the time dimension of the movement, as such calculating the spatial similarity between trajectories or assumes that trajectories have the same lengths (in terms of the number of spatiotemporal points that are composed of) and the same sampling rate. From a different perspective, extending the well known Euclidean Distance metric also used in [20] and [5], we define the notion of spatiotemporal *dissimilarity* between two trajectories T and Q both being valid during a definite time interval $[t_1, t_n]$, by integrating their Euclidean distance in time.

Definition 1: *The Dissimilarity DISSIM(Q,T) between trajectories Q and T being valid during the period $[t_1, t_n]$ is defined as the definite integral of the function of time of the Euclidean distance between the two trajectories during the same period:*

$$DISSIM(Q,T) = \int_{t_1}^{t_n} D_{Q,T}(t)dt ,$$

where $D_{Q,T}(t)$ is the function of the Euclidean distance between trajectories Q and T with time. However, since each trajectory is represented by a collection of discrete points where linear interpolation is applied in between, the definition of dissimilarity is transformed to:

$$DISSIM(Q,T) = \sum_{k=1}^{n-1} \int_{t_k}^{t_{k+1}} D_{Q,T}(t)dt ,$$

where t_k are the timestamps that objects T and Q recorded their position. Obviously, in real world applications, the sampling rates of trajectories may vary, resulting in trajectories with positions sampled at different timestamps; however, considering two trajectories with this characteristic, the position of the first object at the time instance when the second recorded its position can be approximated by applying linear interpolation.

The Euclidean distance between two points moving with linear functions of time between consecutive timestamps, was defined in [6]:

$$D_{Q,T}(t) = \sqrt{at^2 + bt + c} ,$$

where a, b, c are the factors of this trinomial (real numbers, $a \geq 0$).

In order to calculate the integral of $D_{Q,T}(t)$, we distinguish between the following two cases for the value of the non-negative factor a:

- $a = 0$. As shown in [6], it implies that $b = 0$. Hence,

$$\int_{t_k}^{t_{k+1}} D_{Q,T}(t)dt = \frac{\sqrt{c}}{t_{k+1} - t_k}$$

- $a > 0$. According to Meratnia and By [11]:

$$\int_{t_k}^{t_{k+1}} D_{Q,T}(t)dt = \left| \frac{2at+b}{4a}\sqrt{at^2+bt+c} - \frac{b^2-4ac}{8a\sqrt{a}} \operatorname{arcsinh}\left(\frac{2at+b}{\sqrt{4ac-b^2}} \right) \right|_{t_i}^{t_{i+1}}$$

In order to avoid such a computationally heavy operation, we adopt the utilization of the Trapezoid Rule for the computation of the integral, resulting in the following Lemma.

Lemma 1: *The dissimilarity value between two points moving linearly with time can be approximated by the following expression:*

$$DISSIM(Q,T) \approx \frac{1}{2}\sum_{k=1}^{n-1}\left((D_{Q,T}(t_k) + D_{Q,T}(t_{k+1})) \cdot (t_{k+1} - t_k) \right)$$

with the error of the approximation, which depends on t_k, t_{k+1} values, being bounded by:

$$E_{Q,T} \leq \sum_{k=1}^{n-1} \begin{cases} \frac{(t_{k+1}-t_k)^3}{12}\left| D_{Q,T}^{(2)}(-b/2a) \right| & ,if\ t_k \leq -b/2a \leq t_{k+1} \\ \frac{(t_{k+1}-t_k)^3}{12}\left| D_{Q,T}^{(2)}(t_{k+1}) \right| & ,if\ t_k < t_{k+1} < -b/2a \\ \frac{(t_{k+1}-t_k)^3}{12}\left| D_{Q,T}^{(2)}(t_k) \right| & ,if\ -b/2a < t_k < t_{k+1} \end{cases}$$

Proof: in Appendix. ■

Figure 2 demonstrates the trapezoid approximation illustrating the approximation error E in the three above cases: the value of $-b/2a$ is the flex of $D_{Q,T}^{(2)}$; E_k is calculated based on the value of $D_{Q,T}^{(2)}(t_{k+1})$ (case b), E_{k+1}

is calculated based on the value of $D^{(2)}_{Q,T}(-b/2a)$ (case a) and E_{k+2} is calculated based on $D^{(2)}_{Q,T}(t_{k+2})$ (case c).

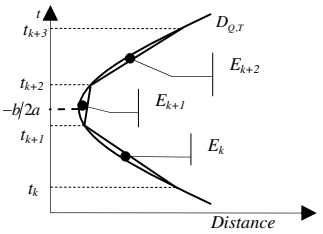

Figure 2. Trapezoid approximation

So far we have defined the dissimilarity between two trajectories (Definition 1) and have approximated this measure with a less expensive computation and a bounded error. As already mentioned, the location of non-recorded timestamps is approximated by linear interpolation between consecutive recorder points. (Support of non-linear e.g. arc, movement is left as a task for future work.) In the sequel, we will provide a series of metrics that will be used in our MST search algorithm.

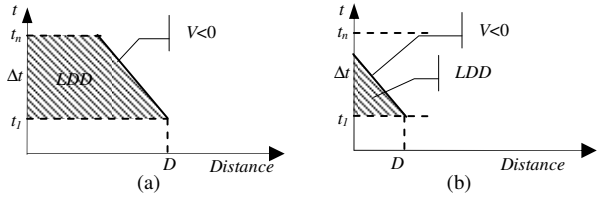

Figure 3. *LDD* definition

3.1 Speed-Dependent Metrics

In this section we define two metrics, namely *OPTDISSIM* and *PESDISSIM*, and provide several lemmas to be used for pruning purposes during MST Search. Before proceeding into the core of the section, we define the *Linearly Depended Dissimilarity* (*LDD*) which is used in the definition of our metrics:

Definition 2: *The Linearly Depended Dissimilarity (LDD) between two moving objects with initial distance D moving collinearly with relative speed V during the period* $\Delta t = [t_1, t_n]$*, is given by the following expression:*

$$LDD(D,V,\Delta t) = \begin{cases} \Delta t \cdot (D + V \cdot \Delta t/2) & ,if\ D + V \cdot \Delta t \geq 0 \\ D^2 / (2|V|) & ,otherwise \end{cases}$$

The relative speed *V* is a negative (positive) number when the distance between the two objects decreases (increases, respectively). To illustrate this definition, consider Figure 3 where *LDD* is described as the shaded area encompassed by the inclined line representing a distance function between two objects moving towards each other with relative speed *V*, with the horizontal lines t_1 and t_n defining Δt . The two cases of *LDD* definition are illustrated in Figure 3(a) and Figure 3(b), respectively.

Any algorithm used for MST search will have to calculate the dissimilarity between a query trajectory and several (indexed or not) trajectories; obviously, at any time instance such an algorithm will have retrieved several parts of candidate MSTs.

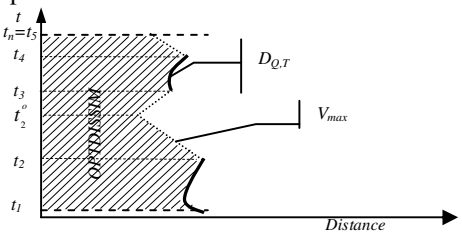

Figure 4. *OPTDISSIM* definition

Although we cannot calculate the exact *DISSIM* of these partially retrieved trajectories from the query trajectory, we can safely estimate a lower bound for it, called *OPTDISSIM*. Consider, for example, Figure 4 that illustrates *OPTDISSIM* of a partially retrieved candidate trajectory *T* from the query trajectory *Q*. *OPTDISSIM* partially consists of the dissimilarity of the entries already retrieved from the index (the shaded area during the time intervals $[t_1,t_2]$ and $[t_3,t_4]$). Regarding the period $[t_4,t_5]$, the smallest possible dissimilarity is given assuming that the moving object started from its position at t_4 approaching the query object with the maximum possible speed (the inclined line between t_4 and t_5). Finally, when dealing with intermediate time intervals such as $[t_2,t_3]$, one has to calculate the time instance t_2^o in which the object stopped its movement towards the query trajectory (the inclined line between t_2 and t_2^o) and then returned to its known position at the time instance t_3 (the inclined line between t_2^o and t_3). Now we can proceed with the formal definition of *OPTDISSIM*:

Definition 3: *The most optimistic DISSIM (OPTDISSIM) between a query trajectory Q and an indexed trajectory T with line segments partially retrieved from the index, during a period* $[t_1, t_n]$*, is defined as:*

$$OPTDISSIM(Q,T,t_1,t_n) = \sum_{k=1}^{n-1} \begin{cases} DISSIM(Q_k,T_k) & ,if\ T_k \in S_R; \\ LDD\left(D_{Q,T}(t_{k+1}), -V_{\max}, (t_{k+1}-t_k)\right) & ,if\ T_k \notin S_R, k=1; \\ LDD\left(D_{Q,T}(t_k), -V_{\max}, (t_{k+1}-t_k)\right) & ,if\ T_k \notin S_R, k=n-1; \\ LDD\left(D_{Q,T}(t_k), -V_{\max}, (t_k^o - t_k)\right) + & \\ LDD\left(D_{Q,T}(t_{k+1}), V_{\max}, (t_{k+1}-t_k^o)\right) & ,otherwise \end{cases}$$

where $D_{Q,T}$ *is the function of distance with time between trajectories Q and T,* S_R *is the set of line segments already retrieved from the index,* V_{max} *is the sum of the maximum speed of indexed trajectories plus the maximum speed of the query trajectory, and* t_k^o *is given by the following expression:*

$$t_k^o = \left(t_k + t_{k+1} + \left(D_{Q,T}(t_{k+1}) - D_{Q,T}(t_k)\right) / V_{\max}\right) / 2$$

Recalling Figure 4, the value of t_k^o is straightforward utilizing the fact that the slope of the two inclined lines between $[t_2, t_2^o]$ and $[t_2^o, t_3]$ is the same and equal to V_{max}. Having defined *OPTDISSIM*, we can provide the following lemma, which will also turn out to be useful for pruning purposes:

Lemma 2: *A trajectory indexed by an R-tree-like structure with line segments partially retrieved from the index cannot have smaller DISSIM from a query trajectory Q during a period* $[t_1, t_n]$ *than its respective OPTDISSIM.*

Proof: According to Definition 3, *OPTDISSIM* is the sum of the *DISSIM* of the trajectory entries already retrieved from the index (belonging to the set S_R), a value which is fixed, plus the *DISSIM* of an object which approached the query trajectory with the maximum possible speed (V_{max}) during the time intervals not already retrieved from the index, with the constraint that the object has to be found at given positions at the start and/or the end of the interval. Therefore, since the two objects approach each other with the maximum possible speed during those periods, the distance between them is minimized; hence minimizing the corresponding integral and consequently their dissimilarity. ■

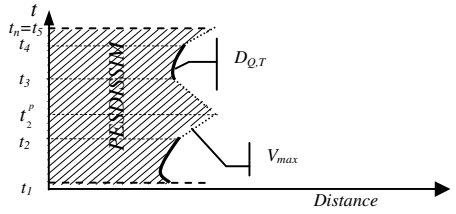

Figure 5. *PESDISSIM* definition

Likewise, by adopting the same scenario where an MST algorithm has only partially retrieved trajectories, one can estimate an upper bound, for the *DISSIM* between the query and a partially retrieved trajectory, named *PESDISSIM*. As illustrated in Figure 5, *PESDISSIM* works in a fashion similar to *OPTDISSIM* with the difference that during time intervals where the movement of the object is not known, the object is assumed to diverge (and not approach) the query trajectory with the maximum possible speed V_{max}. In the same way, we formally define *PESDISSIM*:

Definition 4: *The most pessimistic DISSIM (PESDISSIM) between a query trajectory Q and an indexed trajectory T with line segments partially retrieved from the index, during a period* $[t_1, t_n]$*, is defined as:*

$$PESDISSIM(Q,T,t_1,t_n) =$$
$$\sum_{k=1}^{n-1} \begin{cases} DISSIM(Q_k,T_k) & ,if\ T_k \in S_R; \\ LDD\left(D_{Q,T}(t_{k+1}), V_{\max}, (t_{k+1}-t_k)\right) & ,if\ T_k \notin S_R, k=1; \\ LDD\left(D_{Q,T}(t_k), V_{\max}, (t_{k+1}-t_k)\right) & ,if\ T_k \notin S_R, k=n-1; \\ LDD\left(D_{Q,T}(t_k), V_{\max}, (t_k^p-t_k)\right) + & \\ LDD\left(D_{Q,T}(t_{k+1}), -V_{\max}, (t_{k+1}-t_k^p)\right) & ,otherwise \end{cases}$$

where $D_{Q,T}$, S_R and V_{max} are as defined in previous definitions, and t_k^p is given by the following expression:

$$t_k^p = \left(t_k + t_{k+1} + \left(D_{Q,T}(t_k) - D_{Q,T}(t_{k+1})\right) / V_{\max}\right) / 2$$

The following lemma is directly derived by the definition of PESDISSIM.

Lemma 3: *A trajectory indexed by an R-tree-like structure with line segments partially retrieved from the index cannot have DISSIM from a query trajectory Q during a period* $[t_1, t_n]$ *greater than its respective PESDISSIM.*

Proof: According to Definition 4, *PESDISSIM* is the sum of the *DISSIM* of the trajectory entries already retrieved from the index (belonging to the set S_R), a value which is fixed, plus the *DISSIM* of an object which diverged the query trajectory with the maximum possible speed (V_{max}) during the time intervals not already retrieved from the index, with the constraint that the object has to be found in given positions at the start and/or the end of the interval. Therefore, the distance between the two trajectories during those periods is maximized, hence maximizing their dissimilarity. ■

3.2 Speed-Independent Metrics

The utilization of the previously defined metrics in an MST search algorithm can significantly enhance its performance by pruning several candidate trajectories. However, these metrics are relatively loose, since they are based on the maximum speed V_{max} which, theoretically speaking, could be orders of magnitude higher than the mean object speed. Therefore, we need to define other metrics not influenced by V_{max}, supporting our speed-independent MST search algorithms. These metrics can be developed when an MST algorithm reports index nodes in incremental order of their *MINDIST* from the query trajectory. Obviously, this is a reasonable assumption considering R-tree like structures where a best-first strategy like the one proposed in [7] can be utilized.

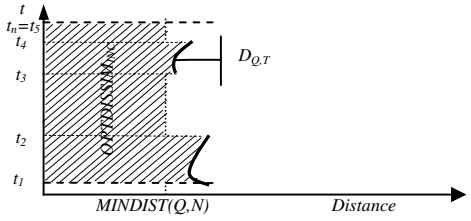

Figure 6. $OPTDISSIM_{INC}$ definition

Consider, for example, Figure 6 that describes the *DISSIM* of a partially retrieved candidate trajectory *T* from the query trajectory *Q*; According to our previous discussion, the *DISSIM* between $[t_1,t_2]$ and $[t_3,t_4]$ is accurately defined. In this case however, we can utilize the fact that index nodes are accessed in incremental order

of their *MINDIST* from the query trajectory. Consequently, any line segment not yet retrieved from the index, cannot be closer to Q than $MINDIST(Q,N)$ where N is the next index node in the queue, and the lower bound of *DISSIM* turns into the shaded area of Figure 6.

More formally, we define $OPTDISSIM_{INC}$ as follows:

Definition 5: *Assuming that index nodes are reported in incremental order of their MINDIST from the query trajectory, the most optimistic DISSIM between a query trajectory Q and an indexed trajectory T during a period* $[t_1,t_n]$ *having a line segment inside a tree node N, is given by the following expression:*

$$OPTDISSIM_{INC}(Q,T,N,t_1,t_n) = \sum_{k=1}^{n-1} \begin{cases} DISSIM(Q_k,T_k) & , \ if\ T_k \in S_R; \\ MINDIST(N,T)\cdot(t_{k+1}-t_k) & , \ otherwise \end{cases}$$

where S_R is the set of line segments already retrieved from the index.

Using the above definition of $OPTDISSIM_{INC}$, we can also define the minimum *DISSIM* of an index node N:

Definition 6: *Assuming that index nodes are reported in incremental order of their MINDIST from the query trajectory, the minimum DISSIM between a trajectory T, indexed by an R-tree-like structure having a line segment inside a node N, and a query trajectory Q during a period* $[t_1,t_n]$*, is defined as:*

$$MINDISSIM_{INC}(Q,N,t_1,t_n) = \min \begin{cases} MINDIST(Q,N)\cdot(t_n-t_1) \\ OPTDISSIM_{INC}(Q,T,N,t_1,t_n), \ \forall T \in S_C \end{cases}$$

where S_C, is the set of the trajectories with line segments already retrieved from the index but not yet fully completed inside the period $[t_1,t_n]$.

Lemma 4: *Assuming that index nodes are reported in incremental order of their MINDIST from a query trajectory Q, a trajectory that is partially stored inside a tree node N cannot have smaller DISSIM from Q during the time period* $[t_1,t_n]$ *than the node's respective* $MINDISSIM_{INC}$.

Proof: Any line segment inside N resides in a trajectory that either belongs to S_c or not. In the former case, considering that nodes are reported in incremental order, trajectory entries not yet retrieved cannot be closer to the query object than the *MINDIST* of the node in which they belong. So, the minimum dissimilarity of an object of S_c is the sum of the dissimilarity of its entries already retrieved from the index, plus the dissimilarity of an object being as close as *MINDIST* to the query trajectory during the rest of the query time period - a sum which corresponds to $OPTDISSIM_{INC}$ definition. In the latter case, where the trajectory does not belong to S_c, the line segment cannot belong to an object fully retrieved from the index because this would lead to duplicate line segments in the index. Hence the line segment belongs to a moving object with no segments retrieved from the previously accessed nodes and it cannot be closer to the query trajectory than *MINDIST*. Thus, in the best case, its distance from the query object during the query period is equal to *MINDIST* and its *DISSIM* is equal to $MINDIST(Q,N)\cdot\Delta t$. ■

3.3 Heuristics

The lemmas provided in previous sections support the following heuristics directly used in the MST Search algorithm that will be presented in Section 4.

- **Heuristic 1**: Every trajectory with *OPTDISSIM* greater than the current most similar (i.e. the one with the smallest calculated *DISSIM* - or *PESDISSIM* if there is not a fully calculated *DISSIM*) cannot be more similar to the query trajectory than the current most similar; as such, it can be pruned from the candidates list.
- **Heuristic 2**: When leaf and internal nodes are reported in incremental order of their *MINDIST* from the query trajectory, every trajectory line segment contained in a node with $MINDISSIM_{INC}$ greater than the current most similar belongs to a moving object, which cannot be more similar to the query trajectory, hence, the node can be pruned from the candidates list. Moreover, since any node reported after the one processed will have *MINDIST* greater or equal to *MINDIST* of the current node, according to Definition 6 the same will hold for the respective values of $MINDISSIM_{INC}$. As a result, all these nodes will have $MINDISSIM_{INC}$ greater than the current most similar, and the algorithm can be terminated since all the remaining nodes can be pruned.

4. A *k*-MST Search Algorithm

The proposed *BFMSTSearch* algorithm (illustrated in Figure 7) accesses the tree structure in a best-first mode, calculating the appropriate *MINDIST*s between the query trajectory and the tree nodes, thus reporting leaf and internal tree nodes in incremental order of their *MINDIST* from the query trajectory. At leaf level, the algorithm uses three hashed in-memory structures: One with the completed trajectories (*Completed*), one with the partially completed trajectories (*Valid*) and one with the partially completed nevertheless already rejected (*Rejected*) trajectories. Both *Completed* and *Valid* in-memory structures store *lists*. Each *list* contains the moving object's time intervals along with their starting and ending distances, its (partial) *DISSIM* the respective calculation error and the *OPTDISSIM* and *PESDISSIM* values. The *Rejected* in-memory structure contains only trajectory *id*s.

When an internal node is processed (lines 32-37) the algorithm calculates the *MINDIST* between the node and the part of the query trajectory Q being inside the temporal extend of the node and then is enqued. When a leaf entry is processed (lines 9-30), the algorithm checks whether it belongs to a *Rejected* moving object (by simply using its id) and rejects it if it does (line 12). In the sequel

it checks whether the entry belongs to a *Valid* moving object and if so retrieves its list *L*; otherwise it creates a new list and adds it to *Valid* (line 13). The algorithm uses a plane sweep method which scans leaf entries and trajectory segments in their temporal dimension in a single pass. This requires that the leaf entries are previously sorted according to their temporal order (line 10), unless the underlying tree structure (such as the TB-tree) stores them in temporal organization anyway.

When a leaf entry and a query trajectory segment overlap in the temporal dimension, the algorithm adds the period to the list *L* (line 17), calculating *DISSIM*, *OPTDISSIM* and *PESDISSIM,* together with the respective calculation error (line 18). If the list *L* is completed, it is removed from the *Valid* and added to the *Completed*, while its *DISSIM* is checked against the current most similar; if smaller, takes its position in *MSim* (lines 20-22). In the case where *L* is not yet completed, its *PESDISSIM* is checked against the current most similar and, if smaller, takes its position in *MSim* (lines 24-25); its *OPTDISSIM* is also compared with the current most similar and, if greater, the list is moved from *Valid* to *Rejected* applying heuristic 1 (lines 26-27).

```
   Algorithm BFMSTSearch (R-tree R, trajectory Q, time
      period Q_per)
 1. EnQueue Queue, R.RootNode, 0, Q
 2. DO WHILE Queue.Count > 0
 3.  Element = DeQueue(Queue)
 4.  N=Element.Node:Q=Element.QueryTrajectory
 5.  IF Completed.Count>0
 6.    IF MINDISSIM_INC(Q,N)>MSim.DISSIM
 7.      Return MSim
 8.  ELSE
 9.    IF N is leaf node
10.      Sort(N, T_S)
11.      FOR EACH leaf entry E in leaf node N
12.        IF Rejected not contains E.Id
13.          IF Valid contains E.Id retrieve list L
             ELSE create list L: Add L in Valid
14.          FIND next query entry QS in Q with
15.          QS.T_e<N.T_S: QE=QS
16.          DO UNTIL QE.T_S > E.T_e
17.            Interpolate to produce nE, nQE
               in period (T_1,T_2):Add (T_1,T_2) in L
18.            Calc DISSIM,PESDISSIM,
                    OPTDISSIM,ERR)
19.            IF L is completed
20.              Move L from Valid to Completed
21.              IF DISSIM<MSim.DISSIM
22.                Update Msim with nE,DISSIM
23.            ELSE
24.              IF PESDISSIM<MSim.DISSIM
25.                Update MSim with nE,PESDISSIM
26.              IF OPTDISSIM>MSim.DISSIM
27.                Move L from Valid to Rejected
28.          NEXT query entry QE
29.          Return QE in the query entry QS
30.      NEXT
31.    ELSE
32.      FOR EACH entry E in the node Element
33.        IF (Q.T_S,Q.T_E) Overlaps (E.T_S,E.T_E)
34.          Interpolate to produce nQE
             in period (T_1,T_2)
35.          Dist = MinDist(nQ, E)
36.          EnQueue Queue, E, Dist, nQ
37.      NEXT
38. LOOP
```

Figure 7. BFMST Search pseudo-code

In both cases where a node (leaf or internal) is processed, the algorithm first checks whether its $MINDISSIM_{INC}$ is greater than the current most similar and if so, the algorithm terminates applying heuristic 2, and returns the current most similar as the query reply (lines 5-7). Note that in order to avoid calculating all the $OPTDISSIM_{INC}$ values involving in the $MINDISSIM_{INC}$ definition (e.g. $T \in S_C$ in definition 6), we first check whether the $MINDIST(Q,N)\cdot(t_n - t_1)$ value of the node is less than the current most similar. In such a case, the calculation of the $OPTDISSIM_{INC}$ values is omitted, since the value of $MINDISSIM_{INC}$ will be less than the current most similar regardless of the $OPTDISSIM_{INC}$ values.

4.3 Extending to *k-MST* algorithms

In the same fashion as in [6], we generalize the above algorithm to support the *k*-most similar trajectory search by considering the following:

- using a buffer of at most *k* (current) most similar trajectories sorted by their actual dissimilarity from the query trajectory;
- terminating the algorithm execution when processing a node with $MINDISSIM_{INC}$ greater than the dissimilarity of the more dissimilar object in the buffer, when extending the *BFMSTSearch* algorithm.

4.4 Error Management

The above MST algorithm calculates dissimilarity between query and indexed trajectories using the approximation introduced in Lemma 1, computing at the same time the appropriate approximation error (denoted as *ERR* in Figure 7). However, apart from its computation, the usage of the error is fundamental in order to compute exact and correct results, a task not explicitly discussed in the description of the BFMST algorithm for sake of clarity. Actually, three modifications must be introduced in the algorithm so as to incorporate the role of the approximation error:

- A candidate most similar trajectory, not already completed, is compared against the current k^{th} most similar by using the value of *PESDISSIM-ERR*.
- A completed candidate most similar trajectory is compared against the current k^{th} most similar using the value *DISSIM-ERR*.
- Instead of using one k^{th} most similar, it is required to utilize a buffer of the candidate k^{th} most similar trajectories. These will be all the trajectories with *DISSIM* greater than the k^{th} most similar and *DISSIM-ERR* less than it.

Finally, a post processing step is required after the execution of the MST algorithm in order to determine the definite *k* MSTs by calculating the actual dissimilarity of each candidate trajectory against the query trajectory. Although, this is a computational heavy operation, it only

happens when the error buffer contains more than one trajectory, or when the order in which the trajectories are reported from the *k*-buffer can be affected by the calculation error of each trajectory's similarity. As an indication, during the entire experimental study, there was no experiment that this case appeared.

5. Experimental Study

The above illustrated algorithm can be implemented in any R-tree-like structure storing historical moving object information such as the 3D R-tree [18], the STR-tree [12] and the TB-tree [12]. Among them, we have chosen the 3D R-tree and the TB-tree that have excellent performance in specific traditional trajectory queries [12]. We used a page size of 4KB and a (variable size) buffer fitting the 10% of the index size, with a maximum capacity of 1000 pages. The experiments were performed in a PC running Microsoft Windows XP with AMD Athlon 64 3GHz processor, 512 MB RAM and several GB of disk space.

5.1 Datasets

Although existing work on trajectory similarity [20], [5] utilized real data, these datasets are not suitable for our objectives due to the fact that they are composed by 2D projections of trajectories without any information about the sampled timestamps; a reasonable fact, bearing in mind that the similarity measured in those papers only depends on the spatial and not the spatiotemporal trajectory similarity. On the other hand, several real datasets recently became available for experimentation purposes [15]; these datasets (representing the movement of a fleet of trucks) were used in our experiments to evaluate the quality of the proposed similarity measure (section 5.2). However, since they are relatively small (273 trajectories and 112203 line segments), they could not expose the actual performance of the algorithms; therefore, the performance study (section 5.3) was conducted using synthetic datasets generated by a custom generator based on the GSTD data generator [16].

Table 2. Summary dataset information

Dataset	*# Objects*	*# Entries (x1K)*	*Speed Distribution*			*Index Size (MB)*	
			Type	*μ*	*σ*	*3D R-tree*	*TB-tree*
Trucks	*273*	*112*	*Real data*			*3.2*	*1.8*
S_{0100}	*100*	*200*	*Lognormal*	*1*	*0.6*	*10.7*	*5.2*
S_{0250}	*250*	*500*	*Lognormal*	*1*	*0.6*	*25.8*	*13.1*
S_{0500}	*500*	*1000*	*Lognormal*	*1*	*0.6*	*51.0*	*26.2*
S_{1000}	*1000*	*2000*	*Lognormal*	*1*	*0.6*	*99.1*	*52.4*

In order to achieve scalability in the volumes of the datasets, we generated synthetic trajectories of 100, 250, 500 and 1000 moving objects resulting in datasets of 200K, 500K, 1000K, and 2000K entries, respectively (the position of each object was sampled approximately 2000 times), thus building indices of up to 100 MB size. Regarding the rest parameters of the generator, the initial distribution and the heading of objects in all cases was random, while their speed was ruled by a normal or lognormal distribution. Table 2 illustrates summary information about the real and the generated datasets and the corresponding indexes. Note that each synthetic dataset is denoted by its cardinality (e.g. the S_{0100} constitutes from 100 trajectories).

5.2 Experiments on the quality

In order to evaluate the quality of the proposed similarity measure we conducted an extensive set of experiments using the real *Trucks* dataset. All trajectories of the dataset were compressed using the TD-TR algorithm described in [11] producing thus artificial trajectories, which were similar (but not identical) to the ones of the original dataset. Then, we used each compressed trajectory to query the original dataset, expecting the algorithm to return the corresponding original trajectory as most similar. We run one set of queries setting k=1 and we counted the number of times the query failed to return the original trajectory as the most similar. We also scaled the value of the TD-TR parameter p from 0.1% to 10% of the length of each trajectory, in order to achieve different values of similarity since an increasing TD-TR parameter produces a compressed trajectory with fewer sampled points and greater dissimilarity regarding the original trajectory. As an example, Figure 8 illustrates (a) an original trajectory and the trajectories produced using the TD-TR algorithm with (b, c, d) different values of p. A major observation derived from Figure 8 is that while the general sketch of the trajectory remains unaffected with the evolution of p, the number of vertices outlining the trajectory decreases and the local details are vanished.

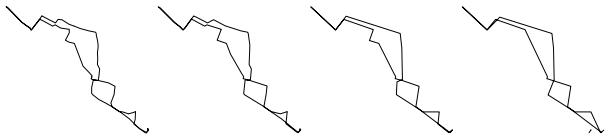

Figure 8. Different degree of compression on a trajectory

Among the related work we have chosen to run the same experiments using the LCSS [20] and EDR [5] similarity measures. We did not include DTW [2] in our experimental study, since both LCSS and EDR were shown to outperform it [20], [5]. We set the value of the parameter ε for these two measures to be a quarter of the maximum standard deviation of trajectories, which leads to the best clustering results, according to [5]. We also normalized the trajectory dataset as suggested in the same paper. Furthermore, for a fair comparison, we made an

obvious improvement over LCSS and EDR, by manually adding samples in the under-sampled (query) trajectory with linear interpolation at the timestamps the checked dataset trajectory was sampled. We called these improved versions LCSS-I and EDR-I respectively.

The results of the experiments evaluating the quality of the proposed similarity metric are illustrated in Figure 9. Clearly, the proposed dissimilarity measure (*DISSIM*) outperforms both its competitors in all settings, regarding also their improved versions. Actually, in the largest part of the experiments, *DISSIM* correctly identifies the original trajectory from which the query one has been produced. On the other hand, it produces false responses only when the value of p exceeds 5%, verifying that it is a very robust similarity metric. LCSS (and LCSS-I) also achieves good quality classifying correctly the query trajectory in the majority of the experimental settings; nevertheless, it is always less accurate than *DISSIM*. Regarding EDR and EDR-I, it turns out that for p values greater than 1% they completely fail to describe the similarity between trajectories, since the false responses exceed 60%.

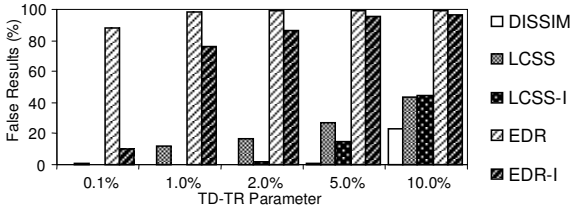

Figure 9. False results increasing the value the TD-TR parameter

The reason of the poor performance of EDR similarity measure demonstrated in these experiments can be explained considering its definition: EDR is the number of insert, delete, or replace operations that are needed to convert trajectory A into B [5]. Thus, supposing that n is the number of vertices in A and m is the number of vertices in (the compressed) A_c, $EDR(A, A_c) \geq n - m$ since at least *n-m* vertices are needed to be added into A_c so as to convert it to A. For an arbitrary dataset trajectory T with k vertices being spatially away from A, it can be easily shown that EDR between T and A_c is at most max(*m, k*). Therefore, if a dataset contains a trajectory T with k vertices and $\max(m,k) \leq n - m$, e.g. a trajectory composed by a small number of vertices, then it also holds that $EDR(T, A_c) \leq EDR(A, A_c)$.

5.3 Experiments on the performance

The proposed algorithm was evaluated with three sets of 500 queries according to the settings presented in Table 3. As such the effects of cardinality (Q1), query length (Q2) and k (Q3) were evaluated using both 3D R- and TB-trees. (Here, we have to note that although related work also uses index structures to prune the search space and support efficient k-MST search, they utilize dedicated indices not designed to support other types of queries. Due to this fact, they are not comparable with our proposal, hence they are not included in our performance study.)

Table 3. Query Settings

Query Set	*Datasets*	*Query Trajectory (as part of a random data trajectory)*	*k*
Q1	S_{0100} … S_{1000}	*5%*	*1*
Q2	S_{0500}	*1% … 100%*	*1*
Q3	S_{0500}	*5%*	*1..10*

Figure 10 illustrates the execution time and the achieved pruned space for the query sets Q1 (scaling with the dataset cardinality), Q2 (scaling with the query length) and Q3 (scaling with the number of k) evaluating the BFMST search algorithm. Clearly, the implementation of the proposed algorithm in both indices demonstrates high pruning power, pruning over 90% in all the experimental settings. Moreover, as also demonstrated in the same figures, the pruning power remains almost constant – or decreases at a low rate - regardless of the scaling factor.

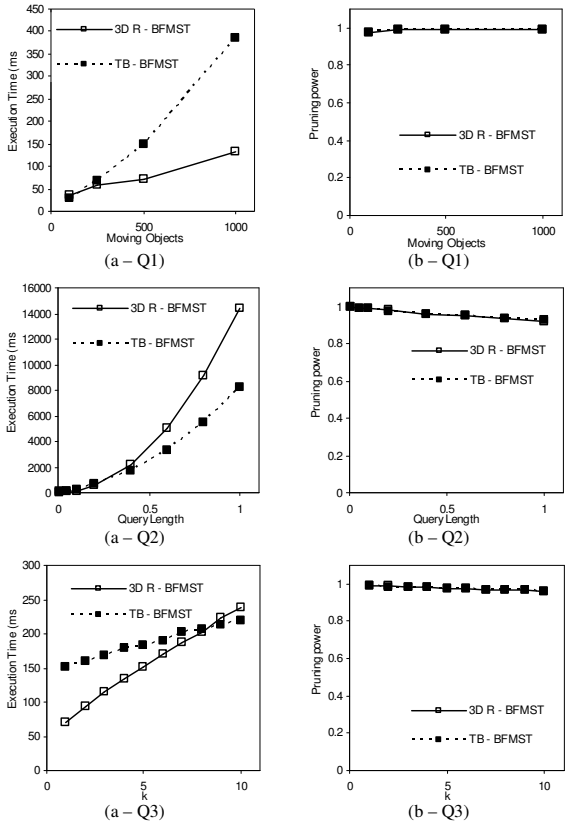

Figure 10. Scaling with the dataset cardinality (Q1) the query length (Q2) and the number of k (Q3)

Regarding execution time, both trees achieve good execution times, due to the fact that the algorithm prunes mainly by the $MINDISSIM_{INC}$ heuristic, which directly rejects all tree nodes not yet processed by the time it realizes. The execution time appears to be linear with the

number of moving objects, quadratic with the query length and sub-linear with *k*. Moreover, the TB-tree outperforms the 3D R-tree as the query length increases, while in the rest of the experimental settings, it is the opposite that is reported.

6. Conclusions and Future Work

Related work on similarity query processing either ignores time dimension of trajectories or considers trajectories with the same sampling rate. In this work, we relaxed these assumptions by defining a novel metric, called *DISSIM*, and then we presented a complete treatment of historical MST queries over moving object trajectories stored on R-tree like structures avoiding the drawbacks of the existing methods. Using our proposed metrics and heuristics for ordering and pruning purposes, we presented a best-first MST algorithm. Under various synthetic and real trajectory datasets, we illustrated the superiority of the proposed *DISSIM* metric against related work [20], [5], in terms of quality, while our algorithm was shown to have high pruning ability when processing MST queries, also verified in the case of *k*-MST queries.

Future work includes the development of algorithms to support *Time-Relaxed MST* queries over trajectories indexed by R-tree like structures using the proposed *DISSIM* metric. This type of query calculates the minimum dissimilarity between trajectories regardless of the time instance in which the query object starts. A second research direction includes the development of selectivity estimation formulae for query optimization purposes investing on the work presented in [17] for predictive spatiotemporal queries.

Acknowledgements

Research partially supported by FP6/IST Programme of the European Union under the GeoPKDD project (2005-08).

References

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

[2] Berndt, J. and Clifford, J., Finding patterns in time series: A dynamic programming approach, *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press.

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

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

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

[6] Frentzos, E., Gratsias, K., Pelekis, N., and Theodoridis, Y., Algorithms for Nearest Neighbor Search on Moving Object Trajectories. *Geoinformatica*, *to appear*

[7] Hjaltason, G., and Samet, H., Distance Browsing in Spatial Databases, *ACM Transactions in Database Systems*, vol. 24(2), pp. 265-318, 1999.

[8] Keogh, E., Exact indexing of dynamic time warping, *Proceedings of VLDB*, 2002

[9] Keogh, E., Wei, L., Xi, X., Lee, S.H., and Vlachos, M., LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures, *Proceedings of VLDB*, 2006.

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

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

[12] Pfoser D., Jensen C. S., and Theodoridis, Y., Novel Approaches to the Indexing of Moving Object Trajectories, *Proceedings of VLDB*, 2000.

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

[14] Theodoridis, Y., Ten Benchmark Queries for Location-based Services, *The Computer Journal*, vol. 46(6), pp. 713-725, 2003

[15] Theodoridis, Y., The R-tree Portal. URL: www.rtreeportal.org (accessed 15 March 2006).

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

[17] Tao, Y., Sun, J., and Papadias, D., Analysis of predictive spatio-temporal queries, *ACM Transactions on Database Systems* vol. 28(4), pp. 295-336, December 2003.

[18] Theodoridis, Y., Vazirgiannis, M., and Sellis, T., Spatio-temporal Indexing for Large Multimedia Applications, *Proceedings of ICMCS*, 1996.

[19] Vlachos, M., Gunopulos, D., and Das, G., Rotation Invariant Distance Measures for Trajectories, *Proceedings of SIGKDD*, 2004.

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

[21] Yanagisawa, Y., Akahani, J., and Satoh, T., Shape-Based Similarity Query for Trajectory of mobile Objects, *Proceedings of MDM*, 2003.

Appendix (Proof of Lemma 1)

The Trapezoid approximation $T_n(f)$ of $\int_{x_0}^{x_n} f(x)dx$ associated with the partition $x_0 < x_1 < ... < x_n$ is given by:

$$T_n(f) = \frac{1}{2}(x_n - x_0)[f(x_0) + 2f(x_1) + ... + 2f(x_{n-1}) + f(x_n)]$$

If $f^{(2)}(x)$ is continuous in $[x_0, x_n]$, then the error $E_n(f)$ in the trapezoid rule is bounded as follows:

$$E_n(f) \le \frac{(x_n - x_0)^3}{12n^2}\left|f^{(2)}(M)\right| \text{, where } \left|f^{(2)}(M)\right| \ge \left|f^{(2)}(x)\right| \forall\, x \in [x_0, x_n]$$

In our case, by setting $n = 1$, we finally calculate:

$$\int_{t_k}^{t_{k+1}} D_{Q,T}(t)dt \approx \frac{1}{2}(D_{Q,T}(t_k) + D_{Q,T}(t_{k+1})) \cdot (t_{k+1} - t_k)$$

with the error of our approximation being bounded by:

$$E_{Q_k,T_k} \leq \frac{(t_{k+1} - t_k)^3}{12} \left| D_{Q,T}^{(2)}(M) \right|$$

Therefore, we determine the maximum value of $D_{Q,T}^{(2)}(t) = \left(4ac - b^2\right) \Big/ \left(4(at^2 + bt + c)^{3/2}\right)$ in $[t_k, t_{k+1}]$. Since the first derivative of $D_{Q,T}^{(2)}(t)$, $D_{Q,T}^{(3)}(t)$ zeroes at $t = -b/2a$ and $D_{Q,T}^{(4)}(-b/2a) = \left(-3a\left(4a\right)^{5/2}\right) \Big/ \left(4(4ac - b^2)^{3/2}\right) \leq 0$ (since $a \geq 0$), the largest value of $D_{Q,T}^{(2)}(t)$ in $\mathbb{R}$ is $D_{Q,T}^{(2)}(-b/2a)$. Finally, we distinguish between three cases:

(a) $t_k \leq -b/2a \leq t_{k+1}$. In this case, $D_{Q,T}^{(2)}(M) = D_{Q,T}^{(2)}(-b/2a)$ and the error is $E_{Q_k,T_k} \leq (t_{k+1} - t_k)^3 \left| D_{Q,T}^{(2)}(-b/2a) \right| \Big/ 12$;

(b) $t_k < t_{k+1} < -b/2a$. In this case, $D_{Q,T}^{(2)}(M) = D_{Q,T}^{(2)}(t_{k+1})$ and the error is $E_{Q_k,T_k} \leq (t_{k+1} - t_k)^3 \left| D_{Q,T}^{(2)}(t_{k+1}) \right| \Big/ 12$;

(c) $-b/2a < t_k < t_{k+1}$. In this case, $D_{Q,T}^{(2)}(M) = D_{Q,T}^{(2)}(t_k)$ and the error is $E_{Q_k,T_k} \leq (t_{k+1} - t_k)^3 \left| D_{Q,T}^{(2)}(t_k) \right| \Big/ 12$.

Summing the n-1 equations of the dissimilarity error calculation by sides, it implies that the approximation error $E_{Q,T}$ is computed as presented in Lemma 1. ■