On the Effect of Trajectory Compression in Spatiotemporal Querying

Elias Frentzos, Yannis Theodoridis

Department of Informatics, University of Piraeus,
80 Karaoli-Dimitriou St, GR-18534 Piraeus, Greece
{efrentzo, ytheod}@unipi.gr
and
Research Academic Computer Technology Institute,
10 Davaki St, GR-11526 Athens, Greece

Abstract. Existing work repeatedly addresses that the ubiquitous positioning devices will start to generate an unprecedented stream of time-stamped positions leading to storage and computation challenges. Hence the need for trajectory compression arises. The goal of this paper is to estimate the effect of compression in spatiotemporal querying; towards this goal, we present an analysis of this effect and provide a model to estimate it in terms of average false hits per query. Then, we propose a method to deal with the model's calculation, by incorporating it in the execution of the compression algorithm. Our experimental study shows that this proposal introduces a small overhead in the execution of trajectory compression algorithms, and also verifies the results of the analysis, confirming that our model can be used to provide a good estimation of the effect of trajectory compression in spatiotemporal querying.

Keywords: Moving Object Databases, Trajectory Compression, Error Estimation

1 Introduction

The recent advances in the fields of wireless communications and positioning technologies activated the concept of Moving Object Databases (MOD), which has become increasingly important and has posed a great challenge to the database management system (DBMS) technology. During the last decade the database community continuously contributes on developing novel indexing schemes [1, 6, 12, 17, 10] and dedicated query processing techniques [7], in order to handle the excessive amount of data produced by the ubiquitous location-aware devices. However, as addressed by [9], it is expected that all these positioning devices will eventually start to generate an unprecedented data stream of time-stamped positions. Sooner or later, such enormous volumes of data will lead to storage and computation challenges. Hence the need for trajectory compression techniques arises.

The objectives for trajectory compression are [9]: to obtain a lasting reduction in data size, to obtain a data series that still allows various computations at acceptable

(low) complexity, and finally, to obtain a data series with known, small margins of error, which are preferably parametrically adjustable. As a consequence, our interest is in lossy compression techniques which eliminate some redundant or unnecessary information under well-defined error bounds. However, existing work in this domain is relatively limited [3, 9, 13, 14], and mainly guided by advances in the field of line simplification, cartographic generalization and time series compression.

Especially on the subject of the error introduced on the produced data by such compression techniques, the single related work [9] provides a formula for estimating the mean error of the approximated trajectory in terms of distance from the original data stream. On the other hand, in this work, we argue that instead of providing a user of a MOD with the mean error in the position of each (compressed) object at each timestamp (which can be also seen as the data (im)precision), he/she would rather prefer to be informed about the mean error *introduced in query results* over compressed data. The challenge thus accepted in this paper is to provide a theoretical model that estimates the error due to compression in the results of spatiotemporal queries. To the best of our knowledge, this is the first analytical model on the effect of compression in query results over trajectory databases.

Outlining the major issues that will be addressed in this paper, our main contributions are as follows:

- We describe two types of errors (namely, *false negatives* and *false positives*) when executing timeslice queries over compressed trajectories, and we prove a lemma that estimates the average number of the above error types. It is proved that the average number of the false hits of both error types depends on the *Synchronous Euclidean Distance* [3, 9, 13] between the original and the compressed trajectory, and the perimeter (rather than the area) of the query window.
- We show how the cost of evaluating the developed formula can be reduced to a small overhead over the employed compression algorithm.
- Finally, we conduct a comprehensive set of experiments over synthetic and real trajectory datasets demonstrating the applicability and accuracy of our analysis.

The model described in this paper can be employed in MODs so as to estimate the average number of false hits in query results when trajectory data are compressed. For example, it could be utilized right after the compression of a trajectory dataset in order to provide the user with the average error introduced in the results of spatiotemporal queries of several sizes; it could be therefore exploited as an additional criterion for the user in order to decide whether compressed data are suitable for his/her needs, and possibly decide on different compression rates, and so on.

The rest of the paper is structured as follows. Related work is discussed in Section 2. Section 3 constitutes the core of the paper presenting our theoretical analysis. Section 4 presents the results of our experimental study, while Section 5 provides the conclusions of the paper and some interesting research directions.

2 Background

In this section we firstly deal with the techniques introduced for compressing trajectories during the last few years, while, we subsequently examine the related

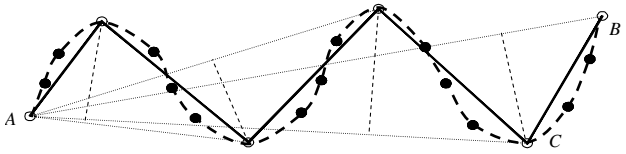


Fig. 1. Top-down Douglas-Peucker algorithm used for trajectory Compression. Original data points are represented by closed circles [9]

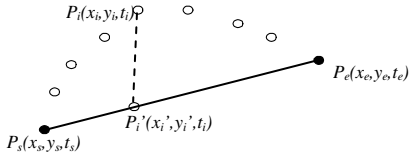


Fig. 2. The Synchronous Euclidean Distance (SED): The distance is calculated between the point under examination (P_i) and the point P_i' which is determined as the point on the line (P_s, P_e) the time instance t_i [9]

work in the field of estimating and handling the error introduced by such compression techniques.

2.1 Compressing Trajectories

As already mentioned, existing work in trajectory compression is mainly guided by related work in the field of line simplification and time series compression. Meratnia and By [9] exploit existing algorithms used in the line generalization field, presenting one top-down and one opening window algorithm, which can be directly applied to spatiotemporal trajectories. The *top-down* algorithm, named TD-TR, is based on the well known Douglas-Peucker [5] algorithm (Fig. 1) introduced by geographers in cartography. This algorithm calculates the perpendicular distance of each internal point from the line connecting the first and the last point of the polyline (line *AB* in Fig. 1) and finds the point with the greatest perpendicular distance (point *C*). Then, it creates lines *AC* and *CB* and, recursively, checks these new lines against the remaining points with the same method, and so on. When the distance of all remaining points from the currently examined line is less than a given threshold (e.g., all the points following *C* against line *BC* in Fig. 1) the algorithm stops and returns this line segment as part of the new - compressed - polyline. Being aware of the fact that trajectories are polylines evolving in time, the algorithm presented in [9] replaces the perpendicular distance used in the DP algorithm with the so-called *Synchronous Euclidean Distance* (*SED*), also discussed in [3, 13], which is the distance between the currently examined point (P_i in Fig. 2) and the point of the line (P_s, P_e) where the moving object would lie, supposed it was moving on this line, at time instance t_i determined by the point under examination (P_i' in Fig. 2).

The time complexity of the original Douglas-Peucker algorithm (on which the TD-TR algorithm is based) is $O(N^2)$, with N being the number of the original data points, while it can be reduced to $O(N\log N)$ by applying the proposal presented in [8].

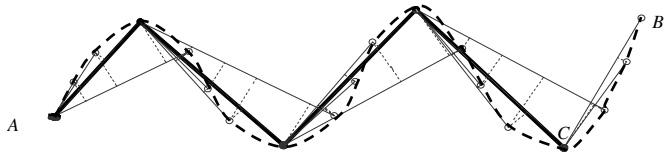


Fig. 3. Opening Window algorithm used for trajectory Compression. Original data points are represented by closed circles [9]

Although the experimental study presented in [9] shows that the TD-TR algorithm is significantly better than the opening window one (presented later in this section) in terms of both quality and compression (since it globally optimizes the compression process), the TD-TR algorithm has the disadvantage that it is not an on-line algorithm and, therefore, it is not applicable to newcoming trajectory portions as soon as they feed a MOD. On the contrary, it requires the a priori knowledge of the entire moving object trajectory.

On the other hand, under the previously described conditions of on-line operation, the *opening window* (OW) class of algorithms can be easily applied. These algorithms start by anchoring the first trajectory point, and attempt to approximate the subsequent data points with one gradually longer segment (Fig. 3). As long as all distances of the subsequent data points from the segment are below the distance threshold, an attempt is made to move the segment's end point one position up in the data series. When the threshold is going to exceed, two strategies can be applied: either the point causing the violation (*Normal Opening Window*, *NOPW*) or the point just before it (*Before Opening Window*, *BOPW*) becomes the end point of the current segment, as well as the anchor of the next segment. If the threshold is not exceeded, the float is moved one position up in the data series (i.e., the window opens further) and the algorithm continuous until the last point of the trajectory is found; then the whole trajectory is transformed into a linear approximation. While in the original OW class of algorithms each distance is calculated from the point perpendicularly to the segment under examination, in the OPW-TR algorithm presented in [9] the *SED* is evaluated. Although OW algorithms are computationally expensive - since their time complexity is $O(N^2)$ - they are very popular. This is because, they are online algorithms, and they can work reasonably well in presence of noise.

Recently, Potamias et al. [13] proposed several techniques based on uniform and spatiotemporal sampling to compress trajectory streams, under different memory availability settings: fixed memory, logarithmically or linearly increasing memory, or memory not known in advance. Their major contributions are two compression algorithms, namely, the *STTrace* and *Thresholds*. The *STTrace* algorithm, utilizes a constant for each trajectory amount of memory *M*. It starts by inserting in the allocated memory the first *M* recorded positions, along with each position's *SED* with respect to its predecessor and successor in the sample. As soon as the allocated memory gets exhausted and a new point is examined for possible insertion, the sample is searched for the item with the lowest *SED*, which represents the least possible loss of information in case it gets discarded. In the sequel, the algorithm checks whether the inserted point has *SED* larger than the minimum one found already in the sample and, if so, the currently processed point is inserted into the

sample at the expense of the point with the lowest *SED*. Finally, the *SED* attributes of the neighboring points of the removed one are recalculated, whereas a search is triggered in the sample for the new minimum *SED*. The proposed algorithm may be easily applied in the multiple trajectory case, by simply calculating a global minimum *SED* of all the trajectories stored inside the allocated memory.

It notably arises from the previous discussion that the vast majority of the proposed trajectory compression algorithms base their decision on whether keeping or discarding a point of the original trajectory on the value of *SED* between the original and the compressed trajectory at this particular timestamp. Consequently, a method for calculating the effect of compression in spatiotemporal querying based on the value of *SED* along the original trajectory data points, would not introduce a considerable overhead in the compression algorithm, since it would require only performing additional operations inside the same algorithm.

2.2 Related Work on Error Estimation

To the best of our knowledge, a theoretical study on modeling the error introduced in spatiotemporal query results due to the compression of trajectories is lacking; our work is the first on this topic covering the case of the spatiotemporal timeslice queries. Nevertheless, there are two related subjects: The first is the determination of the error introduced directly in each trajectory by the compression [9], being the average value of the *SED* between a trajectory p and its approximation q (also termed as synchronous error $E(q, p)$). [9] provide a method for calculating this average value as a function of the distance between p and q along each sampled point. The outcome of this analysis turns to a costly formula, which provides the average error (i.e., mean distance between p and q along their lifetime); however, there is no obvious way on how to use it in order to determine the error introduced in query results.

The second related subject is the work conducted on the context of trajectory uncertainty management, such as [4, 11, 16, 19]. This is due to the fact that the error introduced by compression can also be seen as uncertainty, and thus related techniques may be applied in the resulted dataset (e.g., probabilistic queries). However, such methodology cannot be directly used in the presence of compressed trajectory data, since the task of determining the statistical distribution of the location of the compressed trajectory using information from the original one, is by itself a complex task. Moreover, none of the proposed techniques actually deals with our essential proposal, i.e., the determination of the error introduced in query results using information about the compressed (or uncertain) data.

On the other hand, our approach is based only on the fact that the compression algorithm exploits the *SED* in each original trajectory data point and thus, introduces a very small overhead on the compression algorithm.

3 Analysis

The core of our analysis is a lemma that provides the formula used to estimate the average number of false hits per query when executed over a compressed trajectory

dataset. In this work, we focus on timeslice queries, which can be used to retrieve the positions of moving objects at a given time point in the past and can be seen as a special case of spatiotemporal range queries, with their temporal extent set to zero [18, 12]. This type of query can also be seen as the combination of a spatial (i.e., query window W) and a temporal (i.e., timestamp t) component. As it will be discussed in Section 5, the extension of our model to support range queries with non-zero temporal extent is by no means trivial and is left as future work.

It is important to mention that our model supports arbitrarily distributed trajectory data without concerning about their characteristics (e.g., sampling rate, velocity, heading, agility). Therefore, it can be directly employed in MODs without further modifications. The single assumption we make is that timeslice query windows are uniformly distributed inside the data space. Should this assumption be relaxed, one should mathematically model the query distribution using a probability distribution and modify the following analysis, accordingly. Table 1 summarizes the notations used in the rest of the paper.

Table 1. Table of notations

Notation	Description
$S, T^{\dagger}, T$	the unit space, a trajectory dataset and its compressed counterpart.
$T_i^{\dagger}$, T_i	an original trajectory and its compressed counterpart.
T_N, T_P	the set of false negatives and the set of false positives.
R, $R_{a\times b}$, W_j	the set of all timeslice queries over S, its subset with sides of length a and b along the x- and y- axes, and a timeslice query window.
n, m_i	the cardinality of dataset T and the number of sampled points inside $T_i^{\dagger}$.
$SED_i(t)$, $\delta x_i(t)$, $\delta y_i(t)$	the function of the Synchronous Euclidean Distance (SED) between $T_i^{\dagger}$ and its compressed counterpart T_i, and its projection along the x- and y- axes.
$t_{i,k}$, $SED_{i,k}$, $\delta x_{i,k}$, $\delta y_{i,k}$	the k^{th} timestamp on which trajectory $T_i^{\dagger}$ sampled its position, its Synchronous Euclidean Distance from its compressed counterpart T_i at the same timestamp, and its projection along the x- and y- axes.
$A_{i,j}$	the area inside which the lower-left corner of W_j has to be found at timestamp t_j in order for it to retrieve T_i as false negative (or false positive).
$AvgP_{i,N}(R_{a\times b})$, $AvgP_{i,P}(R_{a\times b})$	the average probability of all timeslice queries $W_j \in R_{a\times b}$, to retrieve T_i as false negative (or false positives).
$E_N(R_{a\times b})$, $E_P(R_{a\times b})$	the average number of false negatives (or false positives) in the results of a query $W_j \in R_{a\times b}$.

Let us consider the unit $3D$ (i.e., $2D$ spatial and $1D$ temporal) space S containing a set $T^{\dagger}$ of n trajectories $T_i^{\dagger}$ and a set T with their compressed counterparts T_i. Let also R be the uniformly distributed set of all timeslice queries posed against datasets $T^{\dagger}$ and T, and $R_{a\times b}$ be the subset of R containing all timeslice queries having sides of length a and b along the x- and y- axis respectively. Two types of errors are introduced when executing a timeslice query $W_j \in R$ over a dataset with the previously described settings:

- *false negatives* are the trajectories which originally qualified the query but their compressed counterparts were not retrieved; formally, the set of false negatives $T_N \subseteq T$ is defined as $T_N = \left\{ T_i \in T : T_i \notin W_j \mid T_i^{\dagger} \in W_j \right\}$;

- *false positives* are the compressed trajectories retrieved by the query while their original counterparts are not qualifying it; formally, the set of false positives $T_P \subseteq T$ is defined as $T_P = \left\{ T_i \in T : T_i \in W_j \mid T_i^{\dagger} \notin W_j \right\}$.

Consider for example Fig. 4 illustrating a set of n uncompressed trajectories $T_i^{\dagger}$, along with their compressed counterparts T_i. Each uncompressed trajectory $T_i^{\dagger}$ is composed by a set of m_i time-stamped points, applying linear interpolation in-between them. Fig. 4 also illustrates a timeslice query W; though W retrieves the compressed trajectory T_1, its original counterpart $T_1^{\dagger}$ does not intersect the query window, encountering a false positive. Conversely, though the original trajectory $T_2^{\dagger}$ intersects W, its compressed counterpart T_2 is not present in the query results, forming a false negative. Having described the framework of our work, we state the following lemma.

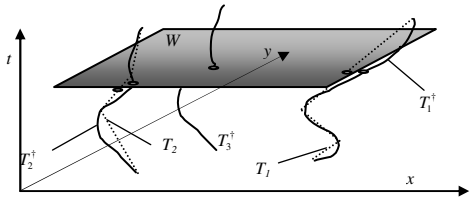


Fig. 4. Problem setting

Lemma 1. *The average number of false negatives $E_N(R_{a\times b})$ and false positives $E_P(R_{a\times b})$ in the results of timeslice queries $W_j \in R_{a\times b}$ uniformly distributed inside the unit space with sides of length a and b along the x- and y- axis respectively, over a compressed trajectory dataset is given by the following formula:*

$$E_N\left(R_{a\times b}\right) = E_P\left(R_{a\times b}\right) =$$

$$\sum_{i=1}^{n}\sum_{k=1}^{m_i-1} \frac{\left(t_{i,k+1}-t_{i,k}\right)}{(1+a)\cdot(1+b)} \cdot \left(\frac{b\left(\left|\delta x_{i,k}\right|+\left|\delta x_{i,k+1}\right|\right)}{2} + \frac{a\left(\left|\delta y_{i,k}\right|+\left|\delta y_{i,k+1}\right|\right)}{2} - \frac{e}{6} \right) \quad \textbf{(1)}$$

where $e = 2\left|\delta x_{i,k}\delta y_{i,k}\right| + 2\left|\delta x_{i,k+1}\delta y_{i,k+1}\right| + \left|\delta x_{i,k}\delta y_{i,k+1}\right| + \left|\delta x_{i,k+1}\delta y_{i,k}\right|$.

Eq.(1) formulates the fact that the average error in the results of timeslice queries over compressed trajectory data is directly related to the projection of the weighted average *SED* along the *x*- and *y*- axis (i.e., $\left(t_{i,k+1}-t_{i,k}\right)$ multiplied by $\left|\delta x_{i,k}\right|+\left|\delta x_{i,k+1}\right|$ or $\left|\delta y_{i,k}\right|+\left|\delta y_{i,k+1}\right|$) multiplied by the respective opposite query dimension (i.e., $b\left(\left|\delta x_{i,k}\right|+\left|\delta x_{i,k+1}\right|\right)$ and $a\left(\left|\delta y_{i,k}\right|+\left|\delta y_{i,k+1}\right|\right)$), while e is a sum of minor importance, since it is the sum of the products between $\left|\delta x_{i,k}\right|, \left|\delta x_{i,k+1}\right|, \left|\delta y_{i,k}\right|, \left|\delta y_{i,k+1}\right|$.

3.1 Proof of Lemma 1

The average number $E_N(R_{a\times b})$ of trajectories being false negatives in the results of a timeslice query $W_j \in R_{a\times b}$, can be obtained by summing up the probabilities $P\left(T_i \notin W_j \mid T_i^{\dagger} \in W_j\right)$ of all dataset trajectories T_i (i=1,…,n) to be false negative regarding an arbitrary timeslice query window $W_j \in R_{a\times b}$:

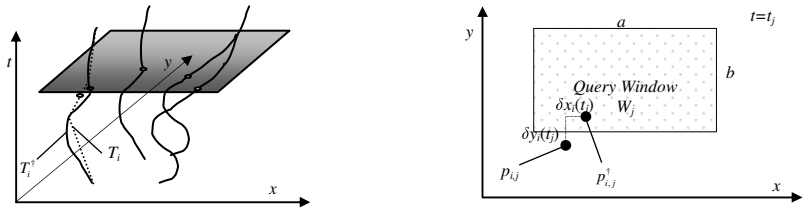


Fig. 5. The intersection of a trajectory $T_i^\dagger$ and its compressed counterpart T_i, with the plane of a timeslice query at timestamp t_j.

$$E_N\left(R_{a\times b}\right)=\sum_{i=1}^{n} AvgP_{i,N}\left(R_{a\times b}\right) \tag{2}$$

Similarly, the average number $E_P(R_{a\times b})$ of trajectories being false positives can be calculated by the following formula:

$$E_P\left(R_{a\times b}\right)=\sum_{i=1}^{n} AvgP_{i,P}\left(R_{a\times b}\right) \tag{3}$$

Hence, our target is to determine $AvgP_{i,N}(R_{a\times b})$ and $AvgP_{i,P}(R_{a\times b})$. Towards this goal, we formulate the probability of a random trajectory be false negative (or false positive), regarding an arbitrary timeslice query window $W_j \in R_{a\times b}$ invoked at timestamp t_j (i.e., $T_i \notin W_j \mid T_i^\dagger \in W_j$, and $T_i \in W_j \mid T_i^\dagger \notin W_j$, respectively). As also illustrated in Fig. 5(b), the intersection of trajectories T_i, $T_i^\dagger$ with the plane determined by the temporal component of W_j (i.e., timestamp t_j) will be demonstrated as two points (points $p_{i,j}$ and $p^\dagger_{i,j}$, respectively, in Fig. 5(b)) having in-between them, distance $\delta x_{i,j}$ and $\delta y_{i,j}$ along the x- and y- axis, respectively.

In order to calculate the quantity of timeslice query windows that would retrieve trajectory T_i as a false negative (false positive) at the timestamp t_j, we need to distinguish among four cases regarding the signs of $\delta x_{i,j}$ and $\delta y_{i,j}$ as demonstrated in Fig. 6 (Fig. 7, respectively). The shaded (with sided stripes) region in all four cases illustrate the area inside which the lower-left query window corner has to be found in order for it to retrieve trajectory T_i as false negative (or false positive, respectively).

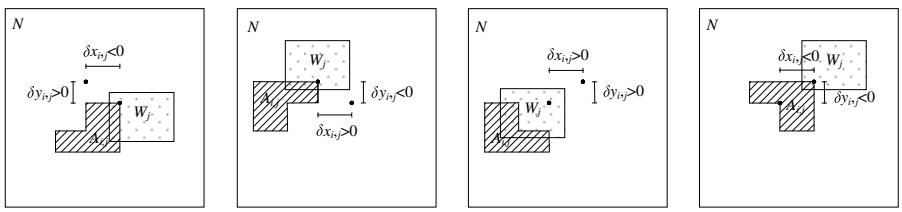
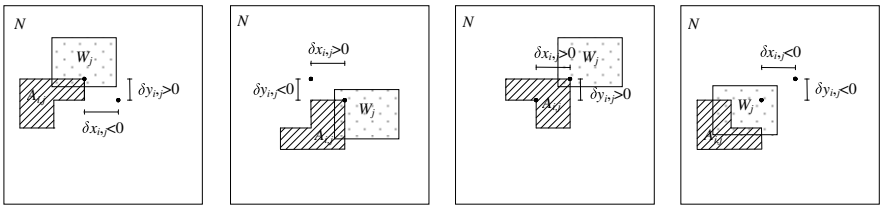


Fig. 6. Regions inside which the lower-left query window corner has to be found in order to retrieve trajectory T_i as false negative



(a) $\delta x_{i,j} < 0 \wedge \delta y_{i,j} > 0$ (b) $\delta x_{i,j} > 0 \wedge \delta y_{i,j} < 0$ (c) $\delta x_{i,j} > 0 \wedge \delta y_{i,j} > 0$ (d) $\delta x_{i,j} < 0 \wedge \delta y_{i,j} < 0$

Fig. 7. Regions inside which the lower-left query window corner has to be found in order to retrieve trajectory T_i as false positive

However, as can be easily derived from these figures, the area of the shaded region in all four cases, is equal for both false negatives and false positives, and can be calculated by the following equation:

$$A_{i,j} = a \cdot b - \left(a - \left|\delta x_{i,j}\right|\right) \cdot \left(b - \left|\delta y_{i,j}\right|\right) \tag{4}$$

Given that W_j is valid when it is (either partially or totally) found inside the unit space, the lower-left query window corner must be found inside a space region of area equal to $(1+a)\cdot(1+b)$. Then, since queries are uniformly distributed inside the unit space, the probability of trajectory T_i to be retrieved as a false negative or false positive at timestamp t_j is:

$$\begin{gathered} P\left(T_i \notin W_j \mid T_i^{\dagger} \in W_j\right) = P\left(T_i \in W_j \mid T_i^{\dagger} \notin W_j\right) = \\ \frac{1}{(1+a)\cdot(1+b)} \cdot A_{i,j} = \frac{1}{(1+a)\cdot(1+b)} \cdot \left(a \cdot b - \left(a - \left|\delta x_{i,j}\right|\right) \cdot \left(b - \left|\delta y_{i,j}\right|\right)\right) \end{gathered} \tag{5}$$

Given also our assumption regarding the distribution of query windows, the average probability of a trajectory T_i to be false negative regarding an arbitrary query window $W_j \in R_{a\times b}$ at any timestamp can be obtained by integrating Eq.(5) over all timestamps inside the unit space. As long as $P\left(T_i \notin W_j \mid T_i^{\dagger} \in W_j\right) = P\left(T_i \in W_j \mid T_i^{\dagger} \notin W_j\right)$, it follows that:

$$\begin{gathered} AvgP_{i,N}\left(R_{a\times b}\right) = AvgP_{i,P}\left(R_{a\times b}\right) = \\ \int_0^1 P\left(T_i \notin W_j \mid T_i^{\dagger} \in W_j\right) dt = \int_0^1 P\left(T_i \in W_j \mid T_i^{\dagger} \notin W_j\right) dt \end{gathered} \tag{6}$$

However, given that each original trajectory T_i is a set of m_i time-stamped points applying linear interpolation in between them, Eq.(6) is transformed as follows:

$$\begin{gathered} AvgP_{i,N}\left(R_{a\times b}\right) = AvgP_{i,P}\left(R_{a\times b}\right) = \\ \sum_{k=1}^{m_i-1} \frac{1}{t_{i,k+1} - t_{i,k}} \int_{t_k}^{t_{k+1}} P\left(T_i \notin W_j \mid T_i^{\dagger} \in W_j\right) dt = \sum_{k=1}^{m_i-1} \frac{1}{t_{i,k+1} - t_{i,k}} \int_{t_k}^{t_{k+1}} P\left(T_i \in W_j \mid T_i^{\dagger} \notin W_j\right) dt \end{gathered} \tag{7}$$

and $\delta x_{i,j}$ and $\delta y_{i,j}$ can be trivially formulated as single functions of t when $t_{i,k} \le t \le t_{i,k+1}$, between sampled points:

$$\delta x_i(t) = \delta x_{i,k} + \left(t - t_{i,k}\right) \cdot \frac{\delta x_{i,k+1} - \delta x_{i,k}}{t_{i,k+1} - t_{i,k}}, \qquad \text{and} \tag{8}$$

$$\delta y_i(t) = \delta y_{i,k} + \left(t - t_{i,k}\right) \cdot \frac{\delta y_{i,k+1} - \delta y_{i,k}}{t_{i,k+1} - t_{i,k}} \tag{9}$$

Substituting Eq.(8), Eq.(9) and Eq.(5) into Eq.(7) and performing the necessary calculations we result in the following formula:

$$AvgP_{i,N}\left(R_{a\times b}\right) = AvgP_{i,P}\left(R_{a\times b}\right) =$$
$$\sum_{k=1}^{m_i-1} \frac{\left(t_{i,k+1} - t_{i,k}\right)}{(1+a)\cdot(1+b)} \cdot \left(\frac{b\left(\left|\delta x_{i,k}\right| + \left|\delta x_{i,k+1}\right|\right)}{2} + \frac{a\left(\left|\delta y_{i,k}\right| + \left|\delta y_{i,k+1}\right|\right)}{2} - \frac{2\left|\delta x_{i,k}\delta y_{i,k}\right| + 2\left|\delta x_{i,k+1}\delta y_{i,k+1}\right| + \left|\delta x_{i,k}\delta y_{i,k+1}\right| + \left|\delta x_{i,k+1}\delta y_{i,k}\right|}{6} \right) \tag{10}$$

Finally, by substituting Eq.(10) into Eq.(2) and defining $e = 2\left|\delta x_{i,k}\delta y_{i,k}\right| + 2\left|\delta x_{i,k+1}\delta y_{i,k+1}\right| + \left|\delta x_{i,k}\delta y_{i,k+1}\right| + \left|\delta x_{i,k+1}\delta y_{i,k}\right|$ we haven proven Lemma 1. ■

3.2 Discussion on Lemma 1

Eq.(1), the main result of Lemma 1, can be straightforwardly used to estimate the average number of false negatives and false positives for timeslice query windows with known size along the *x*- and *y*- axes (*a* and *b*, respectively). It notably arises from this formula that the average number of false negatives in the results of a timeslice query is equal to the respective average number of false positives, while their values depend mainly on the perimeter of the query window (*a*+*b*), rather than its area ($a \cdot b$).However, it should be explicitly mentioned that Lemma 1 holds in the case of uniformly distributed query windows only; as such, the estimated average number of false negatives and false positives serves as a metric estimating data losses due to compression, rather than providing an accurate result regarding individual queries.

Obviously, the evaluation of Eq.(1) is a costly operation; given that it involves a double sum, its time complexity is $O(n \cdot m)$ where n is the number of trajectories and m is the (average) number of sampled points per trajectory. In other words, since Eq.(1) includes the calculation of $\delta x_{i,k}$, $\delta y_{i,k}$, between each tuple of the initial and compressed trajectories on each timestamp the trajectory was originally sampled, it requires to process the entire original dataset along with its compressed counterpart. On the other hand, as already stated in Section 2, the vast majority of the proposed trajectory compression algorithms, base their decision about the point of the original trajectory data to eliminate, on the value of the *SED*; however, since $SED_i(t) = \sqrt{\delta x_i(t)^2 + \delta y_i(t)^2}$, the respective algorithm should first evaluate $\delta x_i(t)$ and $\delta y_i(t)$ at timestamps $t_{i,k}$ producing thus, $\delta x_{i,k}$ and $\delta y_{i,k}$, respectively.

Consequently, any trajectory compression algorithm using the *SED* as the criterion to decide which trajectory points to eliminate, also calculates $\delta x_{i,k}$ and $\delta y_{i,k}$. As such, Eq.(1) can be calculated during the algorithm's execution, adding very small overhead in the original algorithm; the above observation is further confirmed in our experimental study presented in the next section.

Moreover, since Eq.(1) involves the query dimensions *a* and *b*, it follows that different values of *a* and *b* will lead to different calculations for the average error. However, such an approach (i.e., evaluating Eq.(1) from the beginning for every different query size), would lead to high computation cost since it would also require $O(n \cdot m)$ time. In order to overcome this drawback, Eq.(1) can be rewritten as follows:

$$E_N\left(R_{a\times b}\right) = E_P\left(R_{a\times b}\right) = \frac{A \cdot a + B \cdot b + C}{(1+a)\cdot(1+b)}, \quad \textbf{(11)}$$

where $A = \sum_{i=1}^{n}\sum_{k=1}^{m_i-1}\left(t_{i,k+1} - t_{i,k}\right)\cdot\frac{\delta y_{i,k} + \delta y_{i,k+1}}{2}$, $B = \sum_{i=1}^{n}\sum_{k=1}^{m_i-1}\left(t_{i,k+1} - t_{i,k}\right)\cdot\frac{\delta x_{i,k} + \delta x_{i,k+1}}{2}$ and $C = -\sum_{i=1}^{n}\sum_{k=1}^{m_i-1}\left(t_{i,k+1} - t_{i,k}\right)\cdot\frac{e}{6}$. Therefore, in the case where the average error need to be determined for a variety of query sizes (i.e., different sizes of *a* and *b*), rather than directly calculating Eq.(1) for each different query size, the three factors *A*, *B* and *C* could be calculated first, and be subsequently employed in Eq.(11); an approach which dramatically reduces the computation cost to $O(1)$ time.

4 Experiments

In this section, we present several sets of experiments using synthetic and real trajectory datasets. The goal of our experimental study is two-fold:

- first, to present the overhead introduced in the execution of a compression algorithm when calculating during the values of *A*, *B* and *C* factors introduced in Eq.(11), and,
- second, to present the accuracy of the estimation provided by our analytical model regarding the number of false negatives and false positives over synthetic and real trajectory datasets.

Regarding the datasets used, we have exploited on a real-world dataset of a fleet of trucks consisting of 276 trajectories and 112203 entries of trajectory segments [15]. We have also used synthetic datasets produced by a network-based data generator over the San Joaquin road network [2]. The synthetic trajectories generated correspond to 2000 moving objects, each one sampling its position 400 times. All the datasets where normalized in the [0,1] space. In order to test the accuracy of our model and produce compressed datasets, we implemented the TD-TR algorithm proposed by [9]. Then we executed it against all the (real and synthetic) datasets, varying its threshold between 0.001 and 0.02 of the total space, producing thus, the respective compressed datasets. Finally we used the original and compressed datasets and created several 3D R-trees [18] in order to accelerate the querying process used

when performing experiments on the quality. Table 2 illustrates summary information about the (original and compressed) datasets used. The experiments were performed in a PC running Microsoft Windows XP with AMD Athlon 64 3GHz processor, 1 GB RAM and several GB of disk size. All structures and algorithms were implemented in Visual Basic.

Table 2. Summary Dataset Information

	Original Datasets		*Compressed Datasets (# entries)*				
			TD-TR threshold value				
	# trajectories	*# entries*	*0.001*	*0.005*	*0.010*	*0.015*	*0.020*
Trucks	*273*	*112,203*	*62,067*	*20,935*	*12,636*	*9,274*	*7,571*
Synthetic	*2,000*	*800,000*	*229,167*	*120,437*	*88,565*	*74,638*	*65,410*

4.1. Experiments on the performance

In order to demonstrate the applicability of our proposal in trajectory data and estimate the overhead introduced in a trajectory compression algorithm when calculating the values of *A*, *B* and *C* factors introduced in Eq.(11), we run the TD-TR compression algorithm over the real data and measured the average execution time required for each trajectory, scaling also the threshold of the algorithm. We then modified the algorithm in order to calculate the model parameters (i.e., the values of *A*, *B* and *C* in Eq.(11)) within its execution and also run it against the same dataset with the same parameters. The respective results are illustrated in Fig. 8.

In particular, Fig. 8(a) and Fig. 8(b) illustrate the execution time of the TD-TR algorithm per compressed trajectory (in milliseconds), with and without the evaluation of the model parameters, against the trucks, and the synthetic datasets, respectively. A first conclusion is that the algorithm's execution time reduces as the value of the TD-TR threshold increases; this is an expected result, since typically, the number of the algorithm's iterations increase, as the value of the threshold decreases.

However, the main result gathered from Fig. 8 is that the overhead introduced in the algorithm's execution, is typically small (i.e., the difference between the two bars). In all cases, the overhead introduced in the algorithm is between 7% and 19% of the originally required execution time; furthermore, in absolute times, the overhead introduced never exceeds 0.2 milliseconds per trajectory. As a consequence,

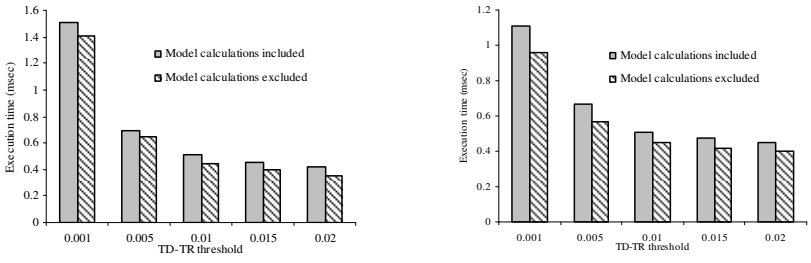


Fig. 8. Execution time for the TD-TR algorithm with and without the calculation of the model parameters over (a) the trucks, and, (b) the synthetic datasets, scaling the value of the TD-TR threshold.

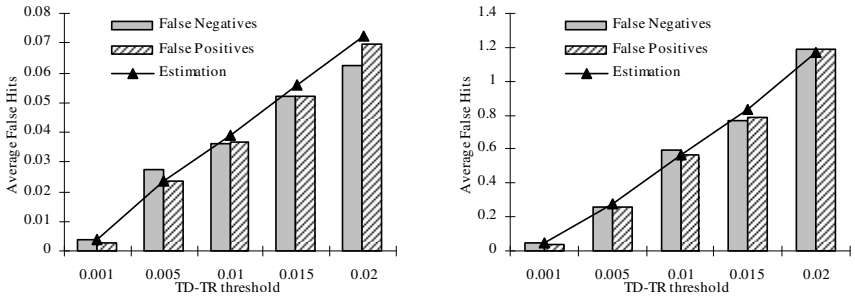


Fig. 9. Accuracy of the model scaling the value of the TD-TR threshold over (a) the trucks, and, (b) the synthetic datasets

the discussion presented in Section 3.2 is further confirmed, and our model can be evaluated as an extension of the compression algorithm's execution, introducing a small / perhaps negligible overhead.

4.2. Experiments on the quality

The statistical measure employed in order to demonstrate the quality of our estimation, are the reported *average number of false negatives and false positives*, $\overline{E_N}$ and $\overline{E_P}$, respectively. Formally, these measures are defined as:

$$\overline{E_N} = 1/n \sum_{i=1..n} E_{N,i} \ , \ \overline{E_P} = 1/n \sum_{i=1..n} E_{P,i} \qquad \textbf{(12)}$$

where, n is the number of executed queries and $E_{N,i}$ ($E_{P,i}$) the actual number of false negatives (false positives, respectively) in the i^{th} query. In the next experiments, n is set to 10000 timeslice queries.

Our first set of experiments was performed against both the real and the synthetic datasets. Specifically, we executed 10000 rectangular timeslice queries of 0.10×0.10 size (i.e., covering 1% of unit space) randomly distributed inside the unit space, over both the original and the compressed datasets (each one stored in separate 3D R-trees), and then, utilizing the results of each particular query over the two datasets, we counted *the actual number of false negatives and false positives*, $E_{N,i}$ and $E_{P,i}$, respectively and then calculated their average values over all the executed queries (termed as *average false hits* – negatives and positives - in all figures describing the experimental evaluation). Fig. 9 illustrates the results of this experiment scaling the value of the compression threshold over the trucks and the synthetic dataset. A first conclusion is that the average number of false hits (negatives and positives) is linear with the value of the TD-TR compression threshold.

Moreover, the estimations, $\overline{E_N}$ and $\overline{E_P}$, of our model are very close to the actual values of average false negatives and false positives reported by the experiments, regardless of the value of the compression threshold. In particular, the average error in the estimation (i.e., the difference between the bars describing the reported by the experiment average number of false negatives and positives, and our model

estimation drawn by a solid line) for the synthetic dataset is around 6%, varying between 0.2% and 14%; regarding the trucks dataset (i.e., Fig. 9(a)), the average error increases around 10.6%, mainly due to the error introduced in small values of TD-TR threshold.

In our second experiment we used the same experimental settings (i.e., datasets, number of queries), but we fixed the TD-TR threshold to 0.01 and scaled the size of the timeslice query window between 0.05×0.05 and 0.30×0.30 (resulting in 0.25% and 9% of unit space, respectively). The corresponding results are illustrated in Fig. 10(a) and Fig. 10(b) against the trucks and the synthetic datasets, respectively. Again, it is clear that our model is highly accurate, producing estimates $\overline{E_N}$ and $\overline{E_P}$ with errors (i.e., the difference between the average values reported by the experiment and our model estimation) for the synthetic dataset between 0.2% and 8.7% and the average error being around 2.9% (while the respective average error for the trucks dataset is 7.5%). Another notable conclusion is that the average number of false positives and false negatives are sub-linear with the query size; an expected result gathered directly from the way that Eq.(11) involves the lengths a and b of the query sides.

In the last experiment we verified the effect of using non-square timeslice queries (i.e., $a \neq b$) over the synthetic datasets (while the experiments with the trucks dataset

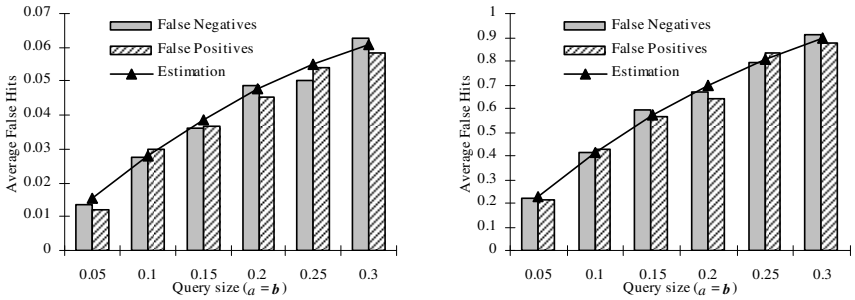


Fig. 10. Accuracy of the model scaling the square query size over (a) the trucks, and, (b) the synthetic datasets

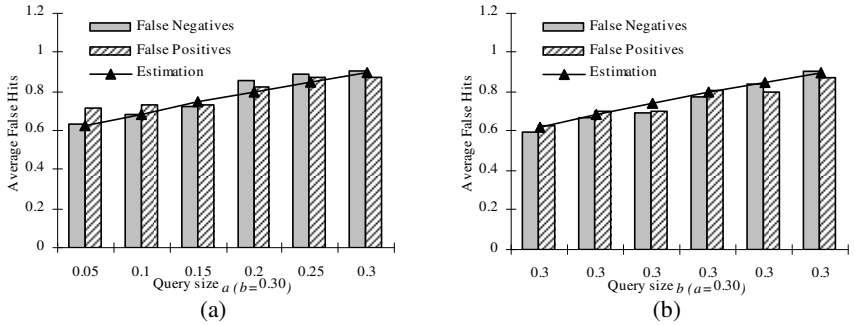


Fig. 11. Accuracy of the model scaling the non-square query size towards (a) the *x*- axis, and (b) the *y*- axis, against the synthetic datasets.

produced similar results). Specifically, we used timeslice query windows with sizes varying from 0.05×0.30 (where $a<<b$) to 0.30×0.30 (where $a=b$); we also scaled the query size towards the other direction (from 0.30×0.05 to 0.30×0.30). The results of this experiment, illustrated in Fig. 11(a) and (b) respectively, resulted in similar outcomes as the ones presented in the previous paragraph regarding square (i.e, $a=b$) timeslice queries. Specifically, our model is once again very accurate, producing estimates with error between 0.6% and 7.2%, while the average error is 3.5%.

5 Conclusions

Related work on the subject of trajectory compression has focused on the development of compression algorithms also emphasizing on the error introduced in the position of each object from the compression. In this work, acknowledging that users are more likely concerned about the error introduced by the compression in spatiotemporal *query results*, we presented the first theoretical model that estimates this error in the results of timeslice queries. We provided a closed formula of the average number of false hits (false negatives and false positives) covering the case of uniformly distributed query windows and arbitrarily distributed trajectory data with various speeds, headings etc. Under various synthetic and real trajectory datasets, we first illustrated the applicability of our model under real-life requirements – it turns out that the estimation of the model parameters introduce only a small overhead in the trajectory compression algorithm - and then presented the accuracy of our estimations, with an average error being around 6%.

There are numerous interesting research directions arising from this work, including the development of the model's counterparts for nearest neighbor queries, or even more, general spatiotemporal range queries (i.e., with temporal extent ≠ 0). More specifically, the extension of our approach towards the second direction, would require to determine the shape of the spatiotemporal space inside which the lower left range query corner (i.e., the minimum point of the range query) has to be found in order for the compressed trajectory to be retrieved as a false hit (negative of positive), in accordance with Fig. 6, Fig. 7, and subsequently to determine its volume in accordance with Eq.(4). Although this volume can be calculated when δx_i and δy_i are expressed as single functions (i.e., between consecutive timestamps), in the general case where δx_i and δy_i are expressed as multi-functions (i.e., different functions in different original trajectory line segments), the respective volume is very hard to be determined. Nevertheless, it is a great challenge for future work.

Finally, we plan to examine the application of our model to trajectory data warehouse environments, which manage aggregate data. Considering for example a trajectory data warehouse with population measurements (i.e., the number of trajectories located in each cell of the partitioned space), our model could be utilized in order to estimate the number of false hits introduced in the number of objects contained within each cell.

Acknowledgements

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

References

1. Almeida, V. T., and Guting, R. H., Indexing the Trajectories of Moving Objects in Networks. *GeoInformatica*, 9(1):33–60, 2005.
2. Brinkhoff, T.: A Framework for Generating Network-Based Moving Objects, *Geoinformatica*, 6 (2), 2002
3. Cao, H., Wolfson, O., and Trajcevski, G., Spatio-temporal Data Reduction with Deterministic Error Bounds. *Proceeding of DIALM–POMC*, 2003.
4. Cheng, R., Kalashnikov, D., and Prabhakar, S., Querying Imprecise Data in Moving Object Environments, *IEEE TKDE*, 16 (9), 2004.
5. Douglas, D. H., Peucker, T. K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer* 10 (1973) 112–122
6. Frentzos, E., Indexing Objects Moving on Fixed Networks. *Proceedings of SSTD*, 2003.
7. Frentzos, E., Gratsias, K., Pelekis, N., and Theodoridis, Y., Algorithms for Nearest Neighbor Search on Moving Object Trajectories, *Geoinformatica* 11 (2), 2007.
8. Hershberger, J., Snoeyink, J.: Speeding up the Douglas-Peucker line-simplification algorithm. *Proceeedings of SDH,* 1992.
9. Meratnia, N., By, R., Spatiotemporal Compression Techniques for Moving Point Objects, *Proceedings of EDBT*, 2004.
10. Ni, J., and Ravishankar, C., Indexing Spatiotemporal Trajectories with Efficient Polynomial Approximation", *IEEE TKDE*, 19(5), 2007
11. Pfoser, D., and Jensen, C. S., Capturing the uncertainty of moving-object representations. *Proceedings of SSD*, 1999.
12. Pfoser D., Jensen C. S., and Theodoridis, Y., Novel Approaches to the Indexing of Moving Object Trajectories, *Proceedings of VLDB*, 2000
13. Potamias, M., Patroumpas, K. and Sellis, T., Sampling Trajectory Streams with Spatiotemporal Criteria, *Proceedings of SSDBM*, 2006.
14. Potamias, M., Patroumpas, K. and Sellis, T., Amnesic online synopses for moving objects, *Proceedings of CIKM*, 2006.
15. Theodoridis, Y. (ed.), The R-tree Portal. URL: www.rtreeportal.org (accessed 5 March, 2007)
16. Trajcevski, G., Probabilistic Range Queries in Moving Objects Databases with Uncertainty. *Proceedings of MobiDE*, 2003.
17. Tao, Y., and Papadias, D., MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries, *Proceedings of VLDB*, 2001
18. Theodoridis, Y., Vazirgiannis, M., and Sellis, T., Spatio-temporal Indexing for Large Multimedia Applications. *Proceedings of ICMCS*, 1996
19. Trajcevski, G., Wolfson, O., Hinrichs, K., and Chamberlain, S., Managing uncertainty in moving objects databases, *ACM TODS* 29 (3), 2004.