The VLDB Journal (2006) 15(3): 211–228
DOI 10.1007/s00778-005-0163-7

REGULAR PAPER

Hu Cao · Ouri Wolfson · Goce Trajcevski

Spatio-temporal data reduction with deterministic error bounds

Received: 26 March 2004 / Revised: 22 January 2005 / Accepted: 20 March 2005 / Published online: 28 April 2006

Abstract A common way of storing spatio-temporal information about mobile devices is in the form of a 3D (2D geography + time) trajectory. We argue that when cellular phones and Personal Digital Assistants become location-aware, the size of the spatio-temporal information generated may prohibit efficient processing. We propose to adopt a technique studied in computer graphics, namely line-simplification, as an approximation technique to solve this problem. Line simplification will reduce the size of the trajectories. Line simplification uses a distance function in producing the trajectory approximation. We postulate the desiderata for such a distance-function: it should be sound, namely the error of the answers to spatio-temporal queries must be bounded. We analyze several distance functions, and prove that some are sound in this sense for some types of queries, while others are not. A distance function that is sound for all common spatio-temporal query types is introduced and analyzed. Then we propose an aging mechanism which gradually shrinks the size of the trajectories as time progresses. We also propose to adopt existing linguistic constructs to manage the uncertainty introduced by the trajectory approximation. Finally, we analyze experimentally the effectiveness of line-simplification in reducing the size of a trajectories database.

Keywords Data reduction · Line simplification · Uncertainty · Moving objects database

This research is supported by NSF Grants 0326284, 0330342, ITR-0086144, 0513736, 0209190, and partly supported by the NSF grant IIS-0325144/003 and the Northrop-Grumman Corp. grant PO 8200082518.

H. Cao (✉) · O. Wolfson
Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607, USA
E-mail: hcao2@cs.uic.edu

G. Trajcevski
Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208, USA

1 Introduction

In this paper we consider lossy compression of spatio-temporal information, particularly trajectories (i.e., motion traces) of moving objects databases. The compression is done by approximating each trajectory, but in contrast to traditional compression methods, we are interested in approximations that guarantee a bound on the error, i.e., in which the distance between the approximation and the original trajectory is bounded. We employ line-simplification for this type of compression. Next, we consider various types of queries on the approximations database, e.g., range and joins. It turns out that even though the error of each approximation is bounded, the error of the query may be unbounded. Whether or not this is so depends on the query type, and the distance function used in the approximation. We determine the pairs (query-type, distance function) for which the query error is bounded. We identify a distance function for which, if the error of the approximation is bounded, then the error of the common query types on the approximations database is also bounded.

Now consider the background of this paper. The management of transient location[1] information is an enabling technology for location-based service applications [1–3]. In this paper we address the problem of managing a set of spatio-temporal points of the form (x, y, t). Such a point indicates that a moving object m was or will be at the geographic location with coordinates (x, y) at time t. These spatio-temporal points may be generated, for example, by a GPS receiver on board m. We will call such point a GPS point, although it may be generated by other means (e.g., PCS network triangulation [4]). The GPS points will be transformed to a unique chosen (X, Y, T) coordinate system[2] [6].

[1] In telecom literature, the term location sometimes refers to the cell or base station associated with the position of a mobile user. In contrast, in this paper, the term location refers to the x, y coordinates.

[2] Global Positioning System (GPS) produces points of the form (x, y, z, t), where (x, y, z) is the 3D position with respect to the earth center. However, we assume, as is common in GIS applications, that an

Consider that a GPS receiver usually generates a new (x, y, t) point every second or two. Furthermore, delivery companies such as Fedex and large municipal transportation agencies own tens of thousands of vehicles that need to be tracked for performance monitoring (e.g., answering a query such as: how many times was bus number 5 late by more than 5 min at a stop) and auditing (e.g., whether the Fedex employee delivered the package at 10: a.m.); and large cellular service companies may soon have hundreds of millions of subscribers whose motion patterns may be subject to statistical analysis for capacity planning. Remember also that one is interested in the historical GPS points that may need to be mined for personal activity pattern analysis (see Offloading Your Memories, NY Times, December 14, 2003; and extend the idea to location tracking), transportation planning and auditing, and cellular network capacity planning.

Of course, storage of motion information may raise important privacy issues. These can be addressed differently in various scenarios. Employers can track their employees during working hours, and this can be, and is made part of existing employment contracts. For example shipments and delivery companies such as Fedex already track delivery employees regularly.[3] In many cases, the issue is access control. For example, a person may be interested in accessing her historical traces, and she may allow access to others. People may want to track their kids or their dementia-stricken parents, owners are allowed to track their pets (pet tracking companies have emerged). Cellular subscriber privacy can be protected by variety of anonymization techniques [7], e.g., removal of personal identification information from the traces. Furthermore, introduction of uncertainty by data reduction, as proposed in this paper, may be a mechanism for addressing privacy concerns, so that the data cannot be traced back to the owner.

Spatio-temporal data mining applications face a severe storage-space problem, as well as a problem in efficiently accessing the motion data. For example, assuming that a GPS point takes 12 bytes and a GPS point is generated (i.e., sampled) every minute for 24 hours a day, 10M cellular subscribers[4] will generate a daily volume of over 160 GB. More frequent samplings will only increase this number. Spatio-temporal data reduction is an attractive solution to the storage and processing performance problems. If the size of spatio-temporal data is reduced by one or two orders of magnitude, the data or the index may become main-memory resident. Furthermore, in online tracking partial trajectories are transmitted periodically from moving objects to central servers [8], and data reduction can save power and wireless bandwidth.[5]

A key observation that lies at the foundation of the data reduction method proposed in this paper is that a GPS point (x, y, t) can be eliminated, and its space saved, if (x, y, t) can be approximated with a reasonable accuracy by interpolating the adjacent (i.e., before and after) GPS points. We formalize this intuition by employing a mechanism based on *line simplification* [9–14], that has been studied in computational geometry, cartography, and computer graphics. Basically, line simplification approximates a polygonal line by another that is “sufficiently close” (the term will be precisely defined), and has less straight-line segments (or points) and thus takes less storage-space. One may be tempted to propose to decrease the GPS sampling frequency to address the above problems. However, compared with line simplification, this simplistic approximation solution has no quality guarantees in terms of error bounds.

Our experimental results indicate that the storage-savings using line simplification is very significant. Specifically, when we used real datasets of moving objects trajectories as GPS traces[6], our experiments indicate that storage-size decreases in an exponential-like manner as the allowed imprecision increases. The trajectories dataset was obtained from the trace of GPS-points recorded by the shuttle buses of the UCLA campus (we elaborate on our experimental settings and results in Sect. 6.). We also conducted experiments on a trajectories dataset that consists of 1,000 trajectories describing the “expected” future motion plans of objects in Chicagoland.

The attractiveness of line simplification (compared to other lossy data compression techniques such as wavelets [15, 16]) stems from the fact that the approximation carries a given error bound. Namely, the distance between the original trajectory and the approximation is bounded by a parameter of the simplification called the error-tolerance. However, we discovered that although the approximation error is bounded, the error of the answers to queries[7] may not be bounded. Whether or not it is bounded, depends on the combination of the distance function (or distance for short) used in the approximation, and the spatio-temporal query type. In other words, for some combinations (query-type, distance) the answer-error is bounded (in this case we call the distance function *sound*[8] for the query type), for others it is not. For example, the Euclidean distance function is not sound for the query “where was moving object m at time t.” Broadly speaking, the reason for unsoundness is that the temporal dimension is treated by the Euclidean distance as a third spatial dimension. It turns out that when considering many types of queries, it is inappropriate to do so. We provide a comprehensive analysis of the soundness of the distance functions for query types.

appropriate projection is applied (e.g., Mercator [5]) to transform each (x, y, z) point into (x, y) for the chosen coordinate system.

[3] http://www.fedex.com/ca_english/about/overview/technology/techinnovation2.html?link=4

[4] Recall that such a point may be generated by PCS network triangulation, not necessarily GPS.

[5] However, throughout this paper, we only consider whole trajectories and do not address the problem of data reduction applied to partial trajectories beyond treating them as full trajectories.

[6] The trajectory of a moving object is the sequence of (x, y, t) points that represent a trip of the object.

[7] i.e., the distance between the answers on the original trajectory and the approximation.

[8] Our notion of soundness should not be confused with the mathematical logic notion by the same name (usually paired with completeness). Our notion is different.

Then, we considered linguistic constructs for querying a database of simplified trajectories using a sound combination. Since a simplified trajectory is an approximation, the operators must contain, in addition to spatio-temporal constructs, uncertainty constructs. Thus, we adopt a set of operators proposed in [17] to deal with location uncertainty, and interpret them for the uncertainty introduced by data reduction. Some examples of these operators are: "Retrieve the moving objects that Possibly intersect a region R Sometime in the time interval T", "Retrieve the moving objects that Definitely intersect a region R Always in the time interval T".

Then, we considered an aging mechanism by which a trajectory is represented by increasingly coarser approximations as time progresses. For example, initially, when the trajectory is stored, it is approximated by a polyline with distance at most 0.1 mile from the original, after 2 months it is approximated by a polyline (which is smaller in size than the first) at distance at most 0.2 miles from the original trajectory, etc. We show that some simplification algorithms are "aging-friendly" (e.g., the Douglas–Peucker heuristic), and some are not (e.g., the optimal simplification algorithm). By aging friendliness we mean that even though the original trajectory is not saved, at every stage we obtain a trajectory that could have been obtained using the larger tolerance from the original trajectory.

We also analyzed experimentally various simplification algorithms. One of the conclusions is that the Douglas–Peuker (DP) algorithm achieves near-optimal savings at a far superior performance.

In summary, the main contributions of this paper are as follows:

- We introduce the concept of soundness of a data compression mechanism.
- We analyze the soundness of several (distance, spatio-temporal query) combinations.
- We quantify experimentally the power of line simplification using different distances and simplification algorithms.
- We analyze the behavior of approximation (simplification) algorithms with respect to data aging, and show that some are well behaved whereas others are not.
- We introduce a set of operators that enable querying of the location database and take into consideration the uncertainty of location modeling.

Before discussing the structure of the paper, let us observe that the proposed data reduction mechanism does not rely on the knowledge of the topology of the road network, as represented e.g., in a digital map. It is possible that when it is known that the motion occurs on a network, this knowledge can enhance the power of line simplification. However, there are many cases in which motion does not occur on a network, e.g., pedestrians, animals, military units in a battlefield. And even if it is known that the motion occurs on a network, line simplification can be used in conjunction with map matching (i.e., snapping the GPS points onto the network; see [18, 19]). For example, line simplification can be used after map matching to eliminate a timestamp that can be inferred by the ones before and after it. The combination of line simplification and map matching is the subject of future work.

The rest of this paper is divided into two parts. Sections 2–5 provide a theoretical/conceptual analysis of trajectory reduction by line-simplification in terms of distance functions used in the simplification, query errors that result from the approximation and the related linguistic issues, and data aging. And Sect. 6 provides an experimental analysis of trajectory reduction by line-simplification. The experimental analysis concentrates on the two distance functions deemed viable (i.e., sound) in the first part of the paper, namely E_u and E_t; however, the reduction power of these functions is compared with that of the more traditional Euclidean distance.

More specifically, the structure of the paper is as follows. Section 2 discusses the concept of a trajectory and introduces the problem of trajectory reduction/simplification. Section 3 introduces the concept of soundness and analyzes it with respect to (distance, query type) combinations. Section 4 discusses uncertain queries, and Sect. 5 analyzes simplification algorithms with respect to aging. Section 6 presents our experimental results of trajectory simplification using different distances, tolerances, and algorithms. In Sect. 7 we position our paper with respect to the relevant works, and in Sect. 8 we provide concluding remarks and directions for future work.

2 Trajectory reduction

Representing the *(location, time)* information of the moving object as a trajectory is a typical approach (c.f. [17, 20, 21]):

Definition 1 A *trajectory* is a function $T : [1, n] \to \mathbb{R}^3$ with $n \in \mathbb{N}$ that satisfies the following conditions: (1) $T(1) = (x_1, y_1, t_1)$, $T(2) = (x_2, y_2, t_2)$, $\ldots$, $T(n) = (x_n, y_n, t_n)$, such that $t_i < t_{i+1}$ for all $i \in \{1, \ldots, n-1\}$; each (x_i, y_i, t_i) is called a *vertex* of the trajectory T; (2) For each $0 \leq \lambda \leq 1$ and for each $i \in \{1, \ldots, n-1\}$, $T(i+\lambda) = (1-\lambda)T(i) + \lambda T(i+1)$. For every point (x, y, t) on the trajectory we say that (x, y) is the *expected location* at time t. The projection of T on the X–Y plane is called the *route* of T.

Intuitively, a trajectory defines the location of a moving object in the X-Y plane as an implicit function of time t. The object is at (x_i, y_i) at time t_i. The vertices of a trajectory represent the location and time of the moving object in the chosen coordinate system. During each segment $[t_i, t_{i+1}]$ we assume that the object moves along a straight line, at constant speed, from (x_i, y_i) to (x_{i+1}, y_{i+1}). Thus the location of the moving object at a point in time t between t_i and t_{i+1}, $(1 \leq i < n)$, called the expected location at time t, is obtained by a linear interpolation between (x_i, y_i) and (x_{i+1}, y_{i+1}).

An illustration of a trajectory and its route is shown in Fig. 1.

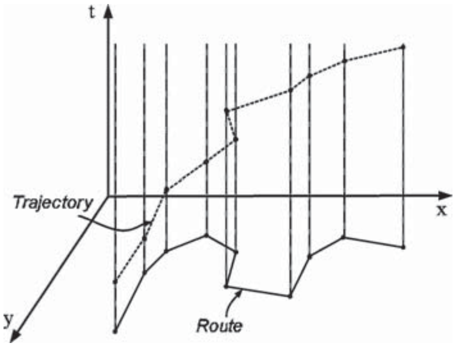


Fig. 1 A trajectory and its two-dimensional route

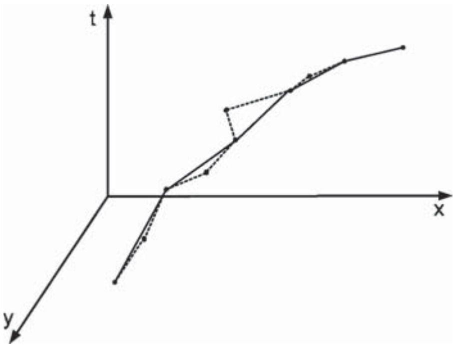


Fig. 2 A simplification (*solid line*) of trajectory in Fig 1.

The (location, time) vertices are obtained from a location technology such as GPS. The location technology may have errors. However, in this paper we assume that the vertices provide the best available estimate of the actual locations and do not attempt to correct or manage the errors. When discussing errors we refer to the errors introduced by line simplification rather than the location technology errors.

Trajectories may impose tremendous storage requirements when the location is sampled frequently. To address this problem, we propose to tradeoff accuracy for efficiency using line simplification. The subject of line simplification has been extensively studied in computational geometry and in many practical applications such as cartography, computer graphics, image processing [9–14] since the 1970s. The goal was similar to ours: given a polygonal curve, approximate it by another one which is "not very far" from the original, but has fewer points. In contrast to our present work, these references considered spatial data, whereas we consider spatio-temporal data. As we will demonstrate in Sect. 3, the temporal dimension has a different impact on queries and cannot be treated simply as a third spatial dimension.

Now we precisely define the "not very far" statement in the context of trajectories. Let M be the distance between a 3D point and a 3D line. The distance $d_M(p, T)$ between a point p and a trajectory T is the minimum (among all line segments of T) M-distance between p and a line segment of T. The distance between two trajectories is the *Hausdorff distance* [22] between them. The Hausdorff M-distance from a trajectory T to another trajectory T' is defined as

$$\tilde{D}_M(T, T') = \max_{p \in T} d_M(p, T')$$

i.e., the Hausdorff distance from T to T' is the maximum distance from a point of T to T'.

The symmetric Hausdorff distance between T and T' (or, for short, the Hausdorff distance between two trajectories) is defined as $D_M(T, T') = \max(\tilde{D}_M(T, T'), \tilde{D}_M(T', T))$; i.e., it is the maximum of the distances from T to T' and from T' to T.

Now we define one of the central concepts of this paper, the simplification of a trajectory.

Definition 2 Let $\{p_1, p_2, \dots p_n\}$ denote the set of vertices of a given trajectory T. For a subset $\{p'_1, p'_2, \dots, p'_s\} \subseteq \{p_1, p_2, \dots p_n\}$, denote by T' the trajectory with these vertices. Let ε be a real number. We say that T' is an ε-simplification of T with respect to M (equivalently, T' is a simplification of T with an *M-tolerance* ε), denoted by $T' = S(T, \varepsilon, M)$, if $D_M(T, T') \leq \varepsilon$.

Figure 2 shows a simplified trajectory corresponding to the original trajectory depicted in Fig. 1.

For a given trajectory T and a tolerance ε, an *optimal ε-simplification* is an ε-simplification with a minimum number of vertices. For a given trajectory, the optimal ε-simplification need not be unique. For example, the trajectory $abcde$ of Fig. 7(a) has three optimal 2.6-simplifications, abe, ace, and ade. An optimal ε-simplification can be found using dynamic programming techniques straightforwardly in $O(n^3)$ time, or in quadratic running time using improved algorithms [9, 10, 13]. For better performance, heuristic-based approaches are often used in practice, especially in GIS. Among them, the best known and studied algorithm is Douglas and Peucker's (DP) [11].

The DP algorithm recursively approximates a given polyline by a "divide and conquer" technique, where the farthest distance vertex is used to select the divide point in the polyline. Given a *begin_vertex* p_i and an *end_vertex* p_j, if the greatest distance from some vertex p_k to the line segment $\overline{p_i p_j}$ is greater than the tolerance ε, break the trajectory into two pieces at p_k and recursively call the procedure on each of the subchains $\overline{p_i p_k}$ and $\overline{p_k p_j}$; Otherwise, the vertices between p_i and p_j are removed from trajectory and this segment is simplified as a straight line $\overline{p_i p_j}$. The pseudo-code of DP is shown in Algorithm 1. Assuming that the trajectory T has n vertices, the algorithm is initially called with the parameters $(T, \varepsilon, 1, n)$.

The DP and optimal algorithms use the distance between a point and a line (denoted M above) and different variants of the algorithms result from using different distance functions. For a comparison of the time complexities of the DP and optimal algorithms, see Table 3 in Sect. 6.

Algorithm 1 DP algorithm—LineSimpDP (T, ε, i, j)

```
Input: Trajectory $T$, tolerance $\varepsilon$, start vertex index $i$ and
    end vertex index $j$.
Output: Simplified trajectory $T'$.
1:  for $k = i$ to $k = j$ do
2:      Find the first vertex $p_k$ farthest from the line $\overline{p_i p_j}$
        (i.e., among the ones with the largest distance, select
        the first). Denote its distance from the line by dist.
3:  end for
4:  If $dist > \varepsilon$ then
5:      LineSimpDP ($T$, $\varepsilon$, $i$, $k$)
6:      LineSimpDP ($T$, $\varepsilon$, $k$, $j$)
7:  else
8:      Remove all vertices from $i + 1$ to $j - 1$.
9:      return.
10: end if
```

Observe that the DP algorithm selects in line 2 the first farthest vertex. As a result, the simplification obtained by the algorithm is unique.

3 Bounded error queries on simplified trajectories

In this section we will analyze the impact of trajectory simplification on the error in the query answers. We focus on the choice of simplification distance functions that have bounds on the error produced when answering spatio-temporal queries.

In the first subsection we introduce the types of spatio-temporal queries that we analyze in the rest of this section and define the functions to measure the query-answer errors caused by trajectory simplification. In the second subsection we define the notion of soundness for a (query-type, distance) pair, which guarantees a bounded-error answer to the query, on a bounded error according to the distance; then we study the soundness of individual (query-type, distance) pairs. In the third subsection we study the sound distances for the spatial joins.

3.1 Spatio-temporal queries and their error

When querying simplified trajectories, the answers may deviate from those on the original trajectories. To incorporate trajectory reduction technique in MOD systems, the imprecision introduced by line simplification must be managed. We measure this imprecision by the error of the query answer on the simplified trajectory, compared to that on the original trajectory. In this subsection, the common spatio-temporal queries and the corresponding answer-error functions are introduced.

We consider the following five types of spatio-temporal queries, *where_at, when_at, intersect, nearest_neighbor*, and *spatial_join*. We introduce the semantics of each one of the operators on a trajectory $T = (x_1, y_1, t_1), (x_2, y_2, t_2), \ldots, (x_n, y_n, t_n)$, as follows:

- *where_at* (T, t)—returns the expected location (c.f. Definition 1) at time t. If $t < t_1$ or $t > t_n$ then the operator is undefined.
- *when_at* (T, x, y)—returns the time t at which a moving object on trajectory T is expected to be at location (x, y). If (x, y) is not on the route of the trajectory, or the moving object leaves the location and later returns to it, then the operator is undefined[9]. If the object is stationary at the location, then the operator returns a time interval $[t_1, t_2]$. We envision that the user will enter a when_at query by clicking a point on a map displayed on the user's monitor. The point will become the (x, y) parameter to when_at. Intuitively, since there will always be round-off errors and approximations, if (x, y) input by the user is not on the route of the trajectory, it is snapped by the system to the closest point on the route. Alternatively, the system does so only if the input point is closer than some threshold to the route.
- *intersect* (T, P, t_1, t_2)—is *true* if the trajectory T intersects the polygon[10] P between the times t_1 and t_2. (This is also called a spatio-temporal range query).
- *nearest_neighbor* (T, O, t)—The operator is defined for an arbitrary set of trajectories O, and it returns a trajectory T_1 from O. The object moving according to T_1, at time t, is closer than any other object of O to the object moving according to T.
- *spatial_join* (O, th)—O is a set of trajectories and the operator returns the trajectory pairs (T_1, T_2) such that their distance (according to some distance functions) is less than the threshold th. The distance used in the join may be different than the distance used for simplification. This operator will be further discussed in the last subsection.

Clearly, the composition of these query types can express more complex queries. For example "Retrieve the 2:p.m. location of the moving objects which will intersect the Parks P_a and P_b between noon and 5:p.m." (assuming that the parks are represented as polygons).

Now we define the notion of query answer error boundedness for a query type on a trajectory T. Intuitively, the error of query type q on T is bounded by δ if the difference between the answer of q on T and the answer of q on a ε-simplification of T is bounded by δ. The definition of the difference depends on the query type.

Definition 3 Let T be a trajectory, E be a distance function, ε be a positive number. For $T' = S(T, \varepsilon, E)$ we say that the error of query type q is bounded by δ, if the following condition is satisfied (the condition depends on the query type)[11]:

[9] If when_at is defined for a location visited more than once, then it returns a finite set of time intervals. In this case, the definitions of E_t and soundness become very complex. For simplicity, we omit consideration of this case.

[10] For simplicity of exposition we will assume throughout this paper that the polygons are convex.

[11] The *spatial_join* is separately discussed in the following subsection.

- *where_at*—For every t for which both T and T' are defined, let $(x, y) = where\_at(T, t)$ and let $(x', y') = where\_at(T', t)$. Then, $\sqrt{(x' - x)^2 + (y' - y)^2} \leq \delta$. *Intuitively, this means that for every time t, the Euclidean distance between the locations on T and T' at t is bounded by δ.*
- *when_at*—For a (x', y') for which $when\_at(T', x', y')$ is defined, let $[t'_b, t'_e] = when\_at(T', x', y')$[12]. Let t_1 be the time of the trajectory vertex of T' immediately before t'_b. Let t_2 be the time of the trajectory vertex of T' immediately after t'_e. Then, for the trajectory point (x, y, t) on the partial trajectory of T between t_1 and t_2 for which (x, y) is closest to (x', y') in terms of the 2D Euclidean distance, $|t - t'_e| \leq \delta$ and $|t - t'_b| \leq \delta$; if there are multiple (x, y, t) points for which (x, y) is closest to (x', y'), then $|t - t'_e| \leq \delta$ and $|t - t'_b| \leq \delta$ are satisfied for each one of them.
  Intuitively, this means for every location on T' and its most probable "actual" location (i.e., the closest one) on the original trajectory, the visiting time difference is bounded.
- *intersect*—For any polygon P, if *intersect* (T', P, t_1, t_2) is *true*, then there exists a time $t \in [t_1, t_2]$ such that the expected location of the original trajectory T at time t is no further than δ from $P \cup$ interior of P. Conversely, if $intersect(T', P, t_1, t_2)$ is *false*, then for every $t \in [t_1, t_2]$, the expected location of the original trajectory T at time t is either outside P, or, if inside, it is within δ of a side of P (i.e., it does not penetrate P by more than δ).
  Intuitively, this means that if the simplification T' intersects P, then T is not further than δ from P; and if T' does not intersect P, then T does not intersect P, or intersects it "very little." Thus, the user, knowing that the query addresses approximate trajectories, may decide to adjust the polygon P accordingly.
- *nearest_neighbor*–Let O be an arbitrary set of trajectories and let $o = nearest\_neighbor(T, O, t)$ and let $o' = nearest\_neighbor(T', O, t)$. Let d_o be the Euclidean distance between o and T at time t, and let $d_{o'}$ be the Euclidean distance between o' and T at time t. Then $|d_o - d_{o'}| \leq \delta$.
  Intuitively, it means that the difference between the distances (o to T) and (o' to T) is bounded by δ. In other words, for any set of trajectories (or moving objects) O, at any time t, the error of the nearest neighbor query is at most δ. □

Given $T' = S(T, \varepsilon, E)$, we denote by *diff*$(q(T), q(T'))$ the minimum δ such that the error of query type q is bounded by δ. The error of a query type q is *unbounded* if for some ε_0 there is no number δ such that for every trajectory T, *diff*$(q(T), q(T')) \leq \delta$; in other words, for ε_0, for every δ there exists a trajectory T such that *diff*$(q(T), q(T')) > \delta$.

The following theorem indicates that the errors of the *intersect and nearest_neighbor* query types are bounded if and only if the error of *where_at* is bounded.

Theorem 1 *For a trajectory T, a real number ε, and a distance function M, let $T' = S(T, \varepsilon, M)$. If for every trajectory T, diff(where_at(T), where_at(T')) $\leq \delta$; then for every trajectory T diff(intersect(T), intersect(T')) $\leq \delta$ and diff(nearest_neighbor(T), nearest_neighbor(T')) $\leq 2\delta$. Furthermore, if the query error of where_at is unbounded, then the query errors of intersect and nearest_neighbor are also unbounded.*

Proof For an arbitrary trajectory T and an arbitrary tolerance ε, let T' be an ε-simplification of T with respect to M. Assume that *diff (where_at*(T), *where_at*$(T')) \leq \delta$. Then for every point $(x', y', t) \in T'$, there is a point $(x, y, t) \in T$ such that the Euclidean distance between (x, y) and (x', y') is less than δ.

Let P be a polygon. Consider the *intersect* (T', P, t_1, t_2) query. If it returns true, then there exists a point (x', y', t') such that $(x', y') \in P \cap T'$ and $t' \in [t_1, t_2]$. Due to *diff (where_at* (T), $where\_at(T')) \leq \delta$, there is a point $(x, y, t') \in T$ such that (x, y) is δ-close to (x', y'). This means that (x, y) is no further than ε from $\{P \cup$ interior of $P\}$. Now, suppose that the *intersect* (T', P, t_1, t_2) query returns false. That means that T' is outside of P. Therefore, every point of T is either outside of P or at most within δ of a side of P.

Consider now the nearest_neighbor query. Let O be an arbitrary set of trajectories. Let $l(o, T, t)$ denote the Euclidean distance between some trajectory $o \in O$ and the trajectory T at time t. Let o' and o be the nearest_neighbors of T' and T, respectively. Based on the fact *diff (where_at*(T), *where_at*$(T')) \leq \delta$ and the triangle inequality, we have: (1) $|l(o', T, t) - l(o', T', t)| \leq \delta$ and (2) $|l(o, T', t) - l(o, T, t)| \leq \delta$. Due to the fact that o' is the nearest neighbor of T' at time t and o is the nearest neighbor of T: (3)$l(o', T', t) \leq l(o, T', t)$ and (4) $l(o, T, t) \leq l(o', T, t)$. Based on (3), we can open the absolute value in inequality (2) and obtain $l(o', T', t) \leq l(o, T, t) + \delta$. Similarly to this deduction, we obtain $l(o, T, t) \leq l(o', T', t) + \delta$. Putting them together, we obtain $|l(o, T, t) - l(o', T', t)| \leq \delta$.

In the above paragraph we assumed that the trajectories of O are not simplified. If they are, a similar methodology can be used to prove that error of the nearest neighbor query answer is at most 2δ.

Now, assume that the error of where_at is unbounded. We will show that the errors of the intersect and nearest_neighbor query types are also unbounded. By definition of unboundedness, there exists some ε_0 such that for every δ there exists a trajectory T and a time t such that the error of where_at at time t is bigger than δ. Thus, we can easily find a polygon P such that T' is inside P at time t, but T is more than δ away from P. Thus, the difference between the intersect query answers on T and on T' is bigger than δ. Thus the error of the intersect query type is unbounded. Similarly, we

[12] If T' is at (x', y') at a single time instance (i.e., it is not stationary at the location), then $t'_b = t'_e$.

can prove that the error of the nearest_neighbor query type is also unbounded. □

Now we define two distance functions between trajectories that will be used extensively in the rest of this paper.

Let $p_m = (x_m, y_m, t_m)$ denote a point, and $\overline{p_i, p_j}$ denote the straight line segment between the vertices $p_i = (x_i, y_i, t_i)$ and $p_j = (x_j, y_j, t_j)$ of a trajectory T. The E_u and E_t distances between the p_m and the straight line segment $\overline{p_i, p_j}$ are defined formally as follows:

- E_u—The three-dimensional time_uniform distance is defined when t_m is between t_i and t_j, as follows: $E_u(p_m, \overline{p_i p_j}) = \sqrt{(x_m - x_c)^2 + (y_m - y_c)^2}$ where $p_c = (x_c, y_c, t_c)$ is the unique point on $\overline{p_i p_j}$ which has the same *time* value as p_m (i.e., $t_c = t_m$). In other words, the time_uniform distance is the 2D Euclidean distance between p_m and the 3D point on $\overline{p_i p_j}$ at time t_m.
- E_t—The time distance is defined as: $E_t(p_m, \overline{p_i p_j}) = |t_m - t_c|$, where t_c is the time of the point on $\overline{p'_i p'_j}$ (which is the *X*-*Y* projection of $\overline{p_i p_j}$) that is closest in terms of the 2D Euclidean distance to p'_m (the *X*-*Y* projection of p_m); if the closest point on $\overline{p'_i p'_j}$ has more than one time point, choose the one that maximizes $|t_m - t_c|$. Intuitively, to find the time distance between p_m and the line segment proceed as follows. First project both on the *X*-*Y* plane, then find the point p'_c on the projected segment which is closest to p'_m, and finally find the difference between the times of p_c and p_m.

It is easy to see that for a trajectory T and its simplification T', the error of where_at is bounded by δ if and only if $D_{E_u}(T, T') \leq \delta$. Similarly, the error of when_at is bounded by δ if and only if $D_{E_t}(T, T') \leq \delta$.

3.2 Soundness of the distances

In this subsection we introduce a scheme of managing the query error introduced by the trajectory simplification, based on the following observation. If the users can predict a priori, namely before data reduction, the maximum error δ of answers to queries for each given simplification tolerance ε, then the simplification can be restricted to tolerances ε for which the error is acceptable. Therefore, the scheme is to simplify trajectories by a distance that allows an acceptable error in queries. In this scheme the maximum error depends on the simplification tolerance, but not on the individual trajectory. This scheme is possible only if the distance is sound. We say that distance function E is sound for q when there exists a bound δ on the error function between the two answers. The formal definition of soundness is as follows:

Definition 4 A distance function E is *sound* for the respective query q if it satisfies the following: For every simplification tolerance ε, there exists a positive number δ, called the *answer error bound*, such that for every trajectory T and for every simplification $T' = S(T, \varepsilon, E)$, *diff*$(q(T), q(T')) \leq \delta$.

In addition to the E_u and E_t distances, the generic Euclidean distance is of interest since it is more efficient in terms of data reduction, as we will demonstrate shortly (and by experiments later). We consider two possibilities. One possibility is to simplify the 3D trajectory using the Euclidean distance. Another possibility is to use a 2D simplification by projecting the trajectory onto its 2D route, and then raising back to 3D by considering the time of the vertices in the simplified route. Formally, let $p_m = (x_m, y_m, t_m)$ denote a point, and $\overline{p_i, p_j}$ denote the straight line segment between the vertices $p_i = (x_i, y_i, t_i)$ and $p_j = (x_j, y_j, t_j)$ of a trajectory T. The distances, denoted as E_2 and E_3, are formally defined as follows:

- E_2—The two-dimensional Euclidean distance, defined as: $E_2(p_m, \overline{p_i p_j}) = \sqrt{(x'_m - x'_c)^2 + (y'_m - y'_c)^2}$, where $p'_c = (x'_c, y'_c)$ is the point on the 2D straight line segment $\overline{p'_i p'_j}$ (i.e., the 2D projection of $\overline{p_i p_j}$) which is closest in terms of Euclidean distance to $p'_m = (x'_m, y'_m)$ (the 2D projection of $p_m = (x_m, y_m, t_m)$).
- E_3—The three-dimensional Euclidean distance, defined as: $E_3(p_m, \overline{p_i p_j}) = \sqrt{(x_m - x_c)^2 + (y_m - y_c)^2 + (t_m - t_c)^2}$ where $p_c = (x_c, y_c, t_c)$ is the point on $\overline{p_i p_j}$ which is closest to $p_m = (x_m, y_m, t_m)$.

The distance functions E_2, E_3, and E_u are illustrated in Fig. 3. As a consequence of their respective definitions, the relationships among E_2, E_3, and E_u, are expressed by the following claim:

Property 1 Given a 3D point p_m and a line segment $\overline{p_i p_j}$ between two vertices of a trajectory, if t_m is between t_i and t_j, then $E_2(p_m, \overline{p_i p_j}) \leq E_3(p_m, \overline{p_i p_j}) \leq E_u(p_m, \overline{p_i p_j})$.

E_t does not have a straightforward relationship to the other distances. It can be shown that E_t is smaller than the E_u divided by the average speed between p_i and p_j.

Property 1 implies that when E_2 is used in a simplification with a given tolerance ε, more vertices of a trajectory will be eliminated than when E_3 is used with ε. Similarly,

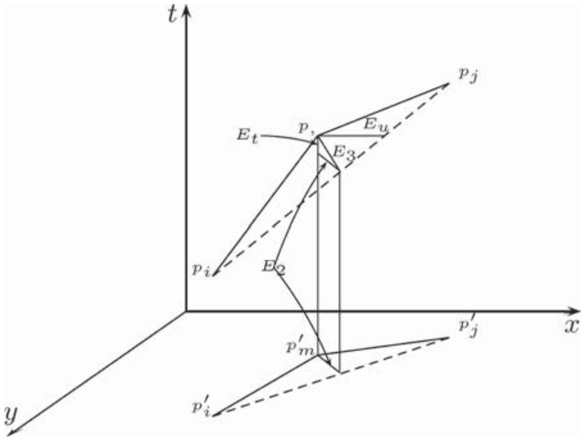


Fig. 3 The relationship among the distances

when using the E_3 distance more vertices will be eliminated than when using the E_u distance with the same tolerance. More formally, as a consequence of Property 1, we have (Let $\|T\|$ denote the “size”, i.e., the number of vertices of a trajectory T):

Corollary 1 *Let T be a trajectory and ε a tolerance. Let $T'_2 = S(T, \varepsilon, E_2)$, $T'_3 = S(T, \varepsilon, E_3)$, and $T'_u = S(T, \varepsilon, E_u)$ denote the respective optimal ε-simplifications. Then $\|T'_2\| \leq \|T'_3\| \leq \|T'_u\|$.*

The order relationship in the Property 1 can be extended on generic distances. Consider two distances M_1 and M_2. Suppose that for every pair of trajectories T and T', if $M_1(T, T') \leq \varepsilon$, then $M_2(T, T') \leq \varepsilon$. In this case, we say that distance M_1 is *weaker* than M_2, denoted as $M_1 \leq M_2$.

The following relationship among distances is also a consequence of Property 1.

Corollary 2 $E_u \leq E_3 \leq E_2$

The following subsumption relationships hold among distances with respect to soundness.

Theorem 2 *For two distances M_1 and M_2, if $M_1 \leq M_2$, then for every query type Q for which M_2 is sound, M_1 is also sound.*

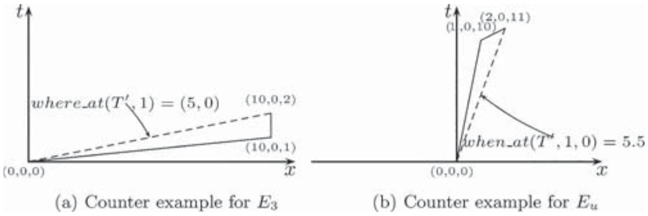
Proof Given two distances $M_1 \leq M_2$, for every trajectory T and every tolerance ε_1, if T' is a ε-simplification of T with respect to M_1, i.e., $T' \in S(T, \varepsilon, M_1)$, then $T' \in S(T, \varepsilon, M_2)$. So, if M_2 is sound for Q with error bound $\delta = f(\varepsilon)$, M_1 is also sound for Q with the same error bound δ. □

Throughout the rest of this section we inspect the soundness of the distances E_2, E_3, E_u, and E_t with respect to the query types. There are 16 possible combinations of distances and query types (four distances and four types). However, we can reduce the number of combinations to inspect by the subsumption relationships among query types and among distances. In this section, we only prove the soundness or unsoundness of some necessary combinations individually. It turns out that no single distance introduced is sound for all the query types. Thus, we introduce simplification with multiple distances and prove that the distance combining E_u and E_t is sound for all query types.

We begin with E_3 and where_at.

Theorem 3 *The E_3 distance function is not sound for the where_at query type.*

Proof Consider the following counterexample. Assume that an object m moves along the x-axis. For every tolerance ε and every answer error bound δ, suppose that the start location and time is (0, 0, 0) and since then m moves 10δ miles in ε min then stops there in next ε min. Then we have trajectory that represents the motion of m as $T = \langle p_1(0, 0, 0)$, $p_2(10\delta, 0, \varepsilon)$, $p_3(10\delta, 0, 2\varepsilon)\rangle$. T can be simplified by E_3 and ε as $T' = \langle$ (0,0,0), (10δ,0,2ε)$\rangle$. However, *diff*(*where_at* (T, ε), $where\_at(T', \varepsilon)$) = *diff*($(10\delta, 0)$, $(5\delta, 0)$) $= 5\delta > \delta$. Figure 4(a) shows an instance of this counterexample with $\varepsilon = 1$ and $\delta = 1$. □



(a) Counter example for E_3 (b) Counter example for E_u

Fig. 4 Counter examples. (The original trajectories are drawn in *solid lines* and the simplifications in *dashed lines*)

Together with Corollary 2 and Theorem 2, the above theorem implies that

Corollary 3 *The E_2 distance function is not sound for the where_at query type.*

However:

Theorem 4 *The E_u distance is sound for the where_at query type. Furthermore, for any simplification tolerance ε, the answer-error-bound of where_at is equal to ε.*

Proof Straightforward. □

Together with Theorem 1, the above theorem implies

Corollary 4 *The E_u distance function is sound for the intersect and nearest_neighbor query types.*

It can also be shown that for the distance E_u, for any simplification tolerance ε, the answer-error-bound of the intersect query type is equal to ε. The bound of the nearest_neighbor query type depends on whether or not the set of trajectories O is simplified; it is ε if O is not simplified, and 2ε if it is.

Theorem 5 *The E_u distance function is not sound for the query type when_at.*

Proof A counterexample is as follows. Assume that an object m moves along the x-axis. For every tolerance ε and every answer error bound δ, there exists a trajectory T with three vertices $\langle p_1(0, 0, 0)$, $p_2(\varepsilon, 0, 10\delta)$, $p_3(2\varepsilon, 0, 11\delta)\rangle$. Figure 4(b) shows an instance of this counterexample with $\varepsilon = 1$ and $\delta = 1$. Consider an E_u ε-simplification T' which consists of the two vertices p_1 and p_3. If we query *when_at*$(T', \varepsilon, 0)$, then the answer is 5.5δ. Observe that $(\varepsilon, 0)$ is also on the route of T and *when_at*$(T', \varepsilon, 0) = 10\delta$. So, the error between two when_at query answers is $4.5\delta > \delta$. □

Theorem 6 *The E_t distance function is sound for the when_at query type. Furthermore, for any simplification tolerance ε, the answer-error-bound of when_at is equal to ε.*

Table 1 The soundness of the distances for spatio-temporal query types

	Where_at	When_at	Intersect	Nearest Neighbor
E_2	No	No	No	No
E_3	No	No	No	No
E_u	Yes	No	Yes	Yes
E_t	No	Yes	No	No

Proof Straightforward. □

Theorem 7 *The E_t distance function is not sound for the where_at query type.*

Proof We prove this theorem using the counterexample of Theorem 3. Clearly, that simplification is also an E_t simplification. However, the answer error is unbounded, as we have illustrated in the proof of theorem 3. □

We can summarize the above results in Table 1.

Table 1 indicates that, of the distances defined previously, there is not a single one that is sound for the four spatio-temporal query types that we have studied. To obtain a distance that is sound for the four spatio-temporal queries, we proceed as follows. Given two distance functions between trajectories M_1 and M_2, we define the combined distance, denoted $M_1 \wedge M_2$, as: $M_1 \wedge M_2(T, T') = \max\{M_1(T, T'), M_2(T, T')\}$.

The following is easy to prove based on the definitions:

Property 2 $M_1 \wedge M_2 \leq M_1, M_1 \wedge M_2 \leq M_2$

Consequently:

Property 3 For a distance M_1 and a distance M_2, let the set of sound operators of M_1 be $SO(M_1)$ and the set of sound operators of M_2 be $SO(M_2)$. Then the distance $M_1 \wedge M_2$ is sound for all the query types in set $SO(M_1) \cup SO(M_2)$.

Thus:

Corollary 5 *The distance $E_u \wedge E_t$ is sound for all the spatio-temporal query types. For any simplification tolerance ε, the answer-error-bound is ε for where_at, when_at, and intersect, and 2ε for nearest-neighbor.*

In conclusion, the appropriate distance to use in a simplification depends on the type of queries expected on the database of simplified trajectories. If all spatio-temporal queries are expected, then $E_u \wedge E_t$ should be used. If only where_at, intersect, and nearest_neighbor queries are expected, then a more concise approximation with the same answer-error-bound can be achieved by using the E_u distance. If only when_at queries are expected, then the E_t distance should be used.

3.3 Spatial Join

The *spatial join* between trajectories is separately discussed in this section. As mentioned in Sect. 3.1, the definition of spatial join depends on the distance function between trajectories. For example, two of the most common distance functions are the Hausdorff distance (defined in Sect. 2) and the *mean square root error*(MSRE) defined as

$$MSRE(T, T') = \frac{1}{t_e - t_s}\int_{t_s}^{t_e}\sqrt{(x(t)-x'(t))^2+(y(t)-y'(t))^2}dt$$

where t_s and t_e are the start time and the end time of the comparison time period. Intuitively, MSRE is the average distance between the trajectories.

Definition 5 Let $D(T_1, T_2)$ be a distance function between two arbitrary trajectories T_1 and T_2. A spatial join with distance function D is called a D-join.

The soundness for the spatial join operation is defined as follows.

Definition 6 Let D be a distance function. A distance M is *sound* for the D-join if it satisfies the following. For every real positive number ε, there exists a real number δ such that for every trajectory T_1 and every simplification $T_1' = (T_1, \varepsilon, M)$ and for every trajectory T_2 and every simplification $T_2' = S(T_2, \varepsilon, M)$, $|D(T_1, T_2) - D(T_1', T_2')| \leq \delta$.

Intuitively, this means that if the approximation is bounded, then the difference between the D-distances is bounded. So if two original trajectories join, then their ε-approximations will join, provided that the join distance is increased by δ.

Theorem 8 *Consider a distance function D that is a metric. Let M be a distance function which is weaker than D (i.e., for every pair of trajectories T and T', if $M(T, T') \leq \varepsilon$, then $D(T, T') \leq \varepsilon$). Then M is sound for the D-join.*

Proof Let ε be an arbitrary positive real number. Consider a D-join between two arbitrary trajectories T_1 and T_2. Let T_1' and T_2' be the ε-simplifications of T_1 and T_2 with respect to M, i.e., $M(T_1, T_1') \leq \varepsilon$ and $M(T_2, T_2') \leq \varepsilon$. We will show that M is sound, i.e., that $|D(T_1, T_2) - D(T_1', T_2')| \leq 2\varepsilon$, by showing that $D(T_1, T_2) - D(T_1', T_2') \leq 2\varepsilon$ and $D(T_1', T_2') - D(T_1, T_2) \leq 2\varepsilon$, which will prove the theorem. First, $D(T_1, T_1') \leq \varepsilon$ and $D(T_2, T_2') \leq \varepsilon$ because $M \leq D$. Meanwhile, since D is a metric, $D(T_1', T_2') \leq D(T_1, T_1') + D(T_1, T_2') \leq D(T_1, T_1') + D(T_1, T_2) + D(T_2, T_2')$, where both inequalities are based on the triangle inequality. Thus, $D(T_1', T_2') - D(T_1, T_2) \leq D(T_1, T_1') + D(T_2, T_2') \leq 2\varepsilon$. Similarly, we have $D(T_1, T_2) - D(T_1', T_2') \leq 2\varepsilon$. Thus, if we take $\delta = 2\varepsilon$ the theorem follows. □

Together with Corollary 2, Theorem 8 implies

Corollary 6 *E_u is sound for E_2-join, E_3-join, and E_u-join.*

Based on Theorem 8, we get

Theorem 9 *E_u is sound for MSRE-join.*

Proof To prove that E_u is sound for MSRE-join, we need to prove that:

1. MSRE is a metric; and
2. $E_u \leq MSRE$.

(1) It is easy to see that MSRE is reflexive and symmetric. We will prove MSRE satisfies the triangle inequality. Let $D(T, T')$ denote the MSRE function between two arbitrary trajectories. Consider the MSRE distance between three arbitrary trajectories T_1, T_2, and T_3. Without loss of generality, we only need to show $D(T_1, T_2) \leq D(T_1, T_3) + D(T_3, T_2)$. $D(T_1, T_3) + D(T_3, T_2) = \frac{1}{t_e - t_s}\int_{t_s}^{t_e}\sqrt{(x_{T_1}(t) - x_{T_3}(t))^2 + (y_{T_1}(t) - y_{T_3}(t))^2}dt$

$$+ \frac{1}{t_e - t_s}\int_{t_s}^{t_e}\sqrt{(x_{T_2}(t) - x_{T_3}(t))^2 + (y_{T_2}(t) - y_{T_3}(t))^2}dt$$

$$= \frac{1}{t_e - t_s}\int_{t_s}^{t_e}(\sqrt{(x_{T_1}(t) - x_{T_3}(t))^2 + (y_{T_1}(t) - y_{T_3}(t))^2}$$

$$+ \sqrt{(x_{T_2}(t) - x_{T_3}(t))^2 + (y_{T_2}(t) - y_{T_3}(t))^2})dt$$

$$\geq \frac{1}{t_e - t_s}\int_{t_s}^{t_e}\sqrt{(x_{T_1}(t) - x_{T_2}(t))^2 + (y_{T_1}(t) - y_{T_2}(t))^2}dt$$

$$= D(T_1, T_2).$$

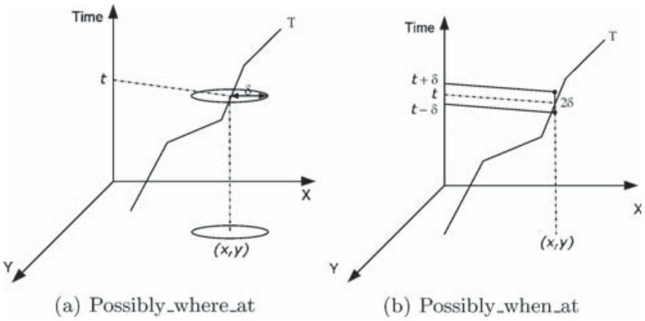
(2) We prove $E_u \leq MSRE$ as follows. For two trajectories T and T', $E_u(T, T')$ is the maximum distance between the same-time points on the two trajectories, and MSRE(T,T') is the average distance between two such points. Thus, if $E_u(T, T') \leq \varepsilon$, then clearly MSRE(T,T') $\leq \varepsilon$. Therefore, by definition, $E_u \leq MSRE$. □

In summary, E_u is sound for the spatial join with E_2, E_3, E_u, or MSRE functions.

4 Uncertainty and possibility queries

In the previous section we considered the same spatio-temporal queries applied to both, the original trajectories, and their simplification. The type of answer was also identical in both cases. When applied to the simplified trajectories, the user knows that there may be an error and, if the distance function used in the simplification is sound, that the error is bounded. But there is no support in the query language for dealing with this error. In this section we consider how to introduce such support. In other words, we propose constructs in the query language and in the format of answers, which take into consideration the uncertainty introduced by the possible error.

Consider spatio-temporal queries that pertain to the original trajectories, posed after the original trajectories have been deleted, and when only the simplifications according to some *sound* distance M are available. For example, consider the query where_at(T,t) on the original trajectory T, and suppose that only the simplification T' is available. Then, we propose that the query returns an "uncertainty disc" around the point-location where_at(T',t). In other words, the query can be computed on the simplified trajectory, and an uncertainty disc with M-radius δ (the answer-error-bound) can be wrapped around the answer to the query where_at(T',t); the answer to the query where_at (T,t) is guaranteed to be within this uncertainty disc.[13] Observe that this scheme works only if the distance M is sound. Observe also that the answer to the query is uncertain, i.e., instead of a point location it provides a region. Then the proper name of the query is "Possibly_where_at." In other words, the semantics of the query Possibly_where_at includes uncertainty. The same arguments apply to Possibly_when_at. The result will be a line segment on the time axis.



(a) Possibly_where_at (b) Possibly_when_at

Fig. 5 The results of possibly_where_at and possibly_when_at

Now consider range queries. We will define a set of query operators that pertain to the original trajectories, but will be processed on the simplified trajectories. For this we need the notion of a possible-original-trajectory. Given a trajectory T and a distance M, a *Possible-Original-Trajectory* POT is a trajectory whose ε-simplification with respect to M is T. The computation of the set of possible original trajectories of a given trajectory T depends on the distance M, and will be discussed later in this section.

We define a set of intersect operators. The location of a moving object changes continuously, hence one may ask if the intersect condition is satisfied *sometime* or *always* within a time interval $[t_1, t_2]$; due to the uncertainty, the object may *possibly* satisfy the condition or it may *definitely* do so. Thus it can be shown that, in effect, we have a total of six operators for spatio-temporal range queries(Sometime_Possibly_Intersect is equivalent to Possibly_Sometime_Intersect, Definitely_Always_Intersect is equivalent to Always_Definitely_Intersect). They are defined as follows for a given simplified trajectory T, polygon (region) R, and time interval $[t_1, t_2]$:

- *Possibly_Sometime_Intersect (T, R, t_1, t_2)*—is *true* iff there exist a possible original trajectory POT and there exists a time $t \in [t_1, t_2]$ such that POT at the time t is in the region R.
- *Possibly_Always_Intersect (T, R, t_1, t_2)*—is *true* iff there exists a possible original trajectory POT of the trajectory T which is in the region R for every t in $[t_1, t_2]$.
- *Always_Possibly_Intersect (T, R, t_1, t_2)*—is *true* iff for every time point $t \in [t_1, t_2]$, there exists a POT which intersects the region R at t (for different t's the POT's may be different).

[13] For the uncertainty disk of the vertices this is "too safe," since we know the actual location. For simplicity of the presentation we do not elaborate on this caveat.

- *Always_Definitely_Intersect (T , R, t1,t2)*—is *true* iff at every time $t \in [t_1, t_2]$, every possible original trajectory POT of the trajectory T, is in the region R.
- *Definitely_Sometime_Intersect* (T, R, t_1, t_2)—is *true* iff for every possible original trajectory POT of the trajectory T, there exists some time $t \in [t_1, t_2]$ in which POT is in the region R.
- *Sometime_Definitely_Intersect* (T, R, t_1, t_2)—is *true* iff there exists a time point $t \in [t_1, t_2]$ at which every possible route POT of the trajectory T is inside the region R.

The evaluation algorithms for the above operators depend on the distance M, and the development of efficient algorithms for a distance may involve extensive future work. However, for the distance E_u efficient algorithms were developed previously (see [17]), in a different context, specifically, uncertain trajectories. There, for a trajectory T and an uncertainty ε (which can be interpreted as the simplification tolerance), we defined a geometric 3D body called the trajectory volume. This corresponds to the set of possible original trajectories of an ε-simplification T. Then we discussed the equivalence and containment of operators (e.g., it turns out that for convex regions Possibly_Always_Intersects is true if and only if Always_Possibly_Intersects), and this discussion carries over verbatim to the present case of querying simplified trajectories. Similarly, we discussed storage of the trajectory volume in a spatial access method, and then processing of the operators via a filter and refinement strategy. Then we provided computational geometry algorithms for the refinement stage for each one of the operators. These results also carry over verbatim to the present case of querying simplified trajectories.

5 Aging of the trajectories

Often, the older the information gets, the less precision may be necessary. Thus it is possible that the coarseness of the trajectory approximation is allowed to increase as time progresses. In this case a data aging mechanism can be introduced.

Assume, for example, that the error tolerance of the trajectories is to be $\leq$ 0.1 mile after the first month; $\leq$ 0.2 after the second month; $\leq$ 0.3 after the third month; etc. Then, one can simplify the original trajectory T to $T' = S(T, 0.1, M)$ after the first month, to $T'' = S(T, 0.2, M)$ after the second month, etc. See Fig. 6 for an illustration. However, a problem arises. At the beginning of the second month one needs to generate $T'' = S(T, 0.2, M)$, but the original trajectory T does not exist anymore, only $T' = S(T, 0.1, M)$. By further simplifying the simplified trajectory T' can one obtain the trajectory T'', or a same-size trajectory? It turns out that the answer to this question depends on the simplification algorithm. Surprisingly, we have not found a paper that addresses this issue in the existing line simplification literature (e.g., [9–14]).

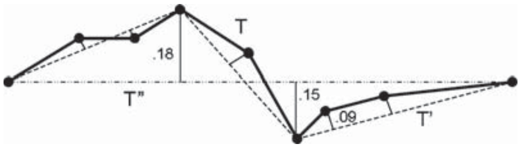


Fig. 6 The aging of a trajectory. The sold lines represent the original trajectory T. The *dashed lines* represent a DP simplified trajectory $T' = S(T, 0.1, M)$. The *dashed-dot line* represents a coarser DP simplification $T'' = S(T, 0.2, M)$. Note that T'' can also be obtained by DP simplifying T' with tolerance 0.2

Various levels of precision are also used in Geographic Information Systems when generating maps of different scales ([14, 23]) and in surface modeling ([24]). In this section, we address the problem of obtaining a lower level of precision from a higher one. We call this data aging.

Now we define this notion precisely. Denote by S_A the line simplification produced by a simplification algorithm A.

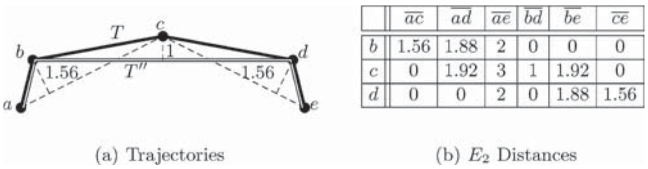
Definition 7 A simplification algorithm A is *aging friendly* with respect to a distance function M if for every ε_1 and every ε_2 such that $\varepsilon_1 < \varepsilon_2$, and for every trajectory T, $S_A(T, \varepsilon_2, M) = S_A(S_A(T, \varepsilon_1, M), \varepsilon_2, M)$

Now suppose that the error tolerances are $\varepsilon_1 < \varepsilon_2 < \varepsilon_3 < \ldots$ for periods of time $P_1 < P_2 < P_3 < \ldots$. If an algorithm is aging friendly, then the simplification for period P_i (namely the P_i trajectories) can be obtained by simplifying the P_{i-1} trajectories using the tolerance ε_i. The result will be the same as if the original trajectories were simplified by the algorithm using the tolerance ε_i.

An *optimal* line simplification algorithm is an algorithm that produces an optimal line simplification for every input trajectory.

Theorem 10 *An optimal line simplification algorithm is not aging-friendly with respect to the E_2 distance.*

Proof (By construction) Consider the example shown in Fig. 7. The original trajectory T consists of vertices $abcde$. The coordinates of the trajectory vertices are a = (0,0, 0), b = (0.5,2,1), c = (6,3,2), d = (11.5,2, 3), e = (12,0, 4). The 2D projection of this trajectory is shown in Fig. 7a. Let $\varepsilon_1 = 1$, $\varepsilon_2 = 1.6$, and the distance M be E_2. The Euclidean distances from the (X, Y) locations of the vertices to the 2D lines are listed in Fig. 7b. The optimal ε_1-simplification is $T' = abde$. If we continue to simplify T' optimally with $\varepsilon_2 = 1.6$, the resulting trajectory T'' is still $abde$ (connected



(a) Trajectories

	$\overline{ac}$	$\overline{ad}$	$\overline{ae}$	$\overline{bd}$	$\overline{be}$	$\overline{ce}$
b	1.56	1.88	2	0	0	0
c	0	1.92	3	1	1.92	0
d	0	0	2	0	1.88	1.56

(b) E_2 Distances

Fig. 7 Counter example to the aging friendliness of the optimal simplification with respect to E_2

by the double lines in Fig. 7a). However, the optimal ε_2-simplification of T is ace. Thus, T'' is not an optimal ε_2-simplification of T. □

Similarly, we can prove that the optimal algorithm is not aging friendly with respect to E_u by constructing a similar example shown in Fig. 8. Assume that the object moves along the X-axis and the original trajectory T consists of vertices $abcde$. The coordinates of these trajectory vertices are $a = (0{,}0{,}0)$, $b = (2{,}0{,}.5)$, $c = (3{,}0{,}6)$, $d = (2{,}0,\ 11.5)$, $e = (0{,}0,\ 12)$. This trajectory is shown in Fig. 8(a). Let $\varepsilon_1 = 1$, $\varepsilon_2 = 1.8$, and the distance M be E_u. In this example, $S(T, \varepsilon_2, E_u)$ is ace whereas $S(S(T, \varepsilon_1, E_u), \varepsilon_2, E_u)$ is $abde$. Therefore,

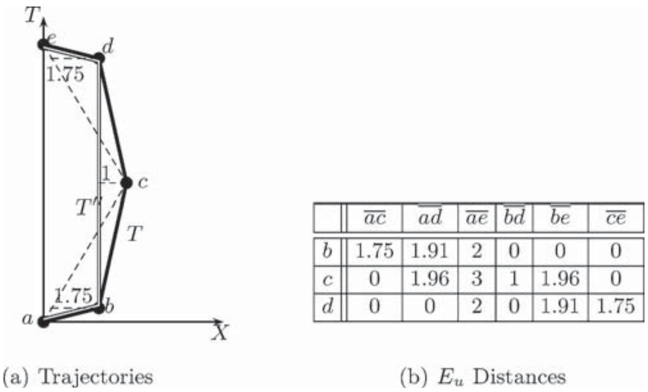
Theorem 11 *An optimal line simplification algorithm is not aging-friendly with respect to the E_u distance.*

The previous theorems indicate that T'' in the proofs is not an optimal ε_2-simplification of T. In fact, it is easy to see that T'' is not even an optimal $(\varepsilon_1 + \varepsilon_2)$-simplification of T. Furthermore, $S_{optimal}(S_{optimal}(T, \varepsilon_1, M), \varepsilon_2, M)$ may not even be an ε_2-simplification of T (let alone an optimal one). To see this, let the original trajectory T, shown in Fig. 9, consist of vertices $abcdefghi$, where $a = (0{,}0{,}0)$, $b = (0{,}1{,}1)$, $c = (1{,}1{,}2)$, $d = (1{,}2.2{,}3)$, $e = (2{,}2{,}4)$, $f = (3{,}2.2{,}5)$, $g = (3{,}1{,}6)$. $h = (4{,}1{,}7)$, $i = (4{,}0{,}8)$. Let $\varepsilon_1 = 0.85$, $\varepsilon_2 = 2$, and the distance M be E_2. Now, consider simplifying T with ε_1. The resulting trajectory T' has vertices aei. Further simplifying T' with $\varepsilon_2 = 2$, we get $T'' = ai$. T'' is not a ε_2-simplification of T since the distance from vertex d to the line ai is larger than ε_2.

Now consider the DP algorithm [12].

Theorem 12 *The DP algorithm (Algorithm 1) is aging friendly with respect to any distance function.*

Proof Let M be a distance function, let $\varepsilon_1 < \varepsilon_2$ be two positive real numbers, and let T be a trajectory. We will prove the theorem by total induction on the number of vertices of T.



	$\overline{ac}$	$\overline{ad}$	$\overline{ae}$	$\overline{bd}$	$\overline{be}$	$\overline{ce}$
b	1.75	1.91	2	0	0	0
c	0	1.96	3	1	1.96	0
d	0	0	2	0	1.91	1.75

(a) Trajectories (b) E_u Distances

Fig. 8 Counter example to the aging friendliness of the optimal simplification with respect to E_u

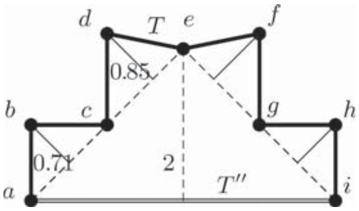


Fig. 9 An example of T'' which is not an ε_2-simplification of T

1.) (Base Case:) For a trajectory T with one or two vertices the simplification is identical to the original trajectory. Consider a trajectory $T = \langle P_1, P_2, P_3 \rangle$ $(n = 3)$. If the distance from P_2 to $\overline{P_1P_3}$ is shorter than ε_1, then $S(T, \varepsilon_1, M) = \langle P_1, P_3 \rangle$. Obviously $S(T, \varepsilon_2, M)$ and $S(\underline{S(T, \varepsilon_1, M), \varepsilon_2}, M))$ are $\langle P_1, P_3 \rangle$ too; If the distance from P_2 to $\overline{P_1P_3}$ is larger than ε_1, then $S(T, \varepsilon_1, M) = T$, so $= S(S(T, \varepsilon_1, M), \varepsilon_2, M)) = S(T, \varepsilon_2, M)$.
2.) (Hypothesis:) Assume that the proposition is true for every trajectory T which has n or less vertices.
3.)(Step:) Consider a trajectory with $n + 1$ vertices. Let $dist$ denote the maximum distance between some vertex P_i on T and the straight line $\overline{P_1P_{n+1}}$. If $dist < \varepsilon_2$, then $S(T, \varepsilon_2, M)$ and $S(S(\underline{T, \varepsilon_1, M}), \varepsilon_2, M))$ are the same straight line segment—$\overline{P_1P_{n+1}}$.

On the other hand, if $\underline{dist > \varepsilon_2} > \varepsilon_1$, let P_i be the first vertex that is farthest from $\overline{P_1P_{n+1}}$. Regardless whether the DP algorithm has ε_1 or ε_2 as a tolerance, it will split the trajectory T into two segments $\overline{P_1P_i}$ and $\overline{P_iP_{n+1}}$, each of which has $\leq n$ vertices and, consequently, falls into the inductive hypothesis case. □

6 Experimental study

In this section we give a description of our experimental results and the conclusions based on them. In the first subsection we describe the setting for the experiments, namely the input datasets, the methodology, and the environment. Then, in the following subsection, we compare the errors of the line simplification and the wavelet transform. We also analyze the data reduction obtained by the different distances and by two simplification methods, the DP algorithm and the optimal one. Finally, We compare the execution times of the DP algorithm and the optimal one.

6.1 Datasets and methodology

We used two datasets in our experiment, both of which are generated based on real data. One dataset, denoted D1, represents the past motion, and the other one, denoted D2, represents the estimation of future motion. A *D1 trajectory* is a sequence of (x, y, t) vertices representing a trace of the GPS points recorded by the on-board GPS device. A *D2 trajectory* is a sequence of vertices describing the motion plan of an object. It is constructed as follows. Suppose that we have a set of locations that the object is going to visit, and we assume that in between these locations the object is moving along the shortest path. Given an electronic map, we first find the route of future trajectory as the shortest path along the "to-be-visited" locations. Then, we compute the arrival time of each vertex in the route using the average speed in that block(which is provided in the map). The time of the first trajectory vertex is given as the beginning time of the trajectory. D1 trajectories are denser than D2 trajectories in the sense that they have more vertices per time unit.

The *D2* trajectory dataset has 1,000 trajectories generated from real GDT electronic maps of 10 counties around Chicago, Illinois. Given two random points on the map, for a start location and a destination, and a random start time of the trip, we used an Avenue[14] script to create a trajectory with optimal drive time, using the construction method mentioned above. The average number of vertices per trajectory is 418. The average length of a trajectory is 45.13 miles. The longest trajectory is 306.82 miles long, and the shortest one is 1.7325 miles. The lengths of trajectory segments also vary. However, Nearly 65% of segments' lengths are less then 0.1 mile and only 0.2% of them are larger than 1 mile.

The other dataset consists of 38 *D1* trajectories which are obtained from the on-board GPS receivers on the UCLA campus shuttle buses. The location was sampled every second[15]. The data was collected on April 24, 2002. Each trajectory represents a continuous trip of a UCLA campus shuttle-bus on that day. The average number of vertices per trajectory is 7085 and the average length of a trajectory is 16.352 miles.

The data reduction is expressed by the reduction ratio (rr), which is *number of vertices of the simplified trajectory / number of vertices of the original trajectory (i.e., the trajectory before simplification)*. In other words, the storage savings of the simplification is $1 - rr$. For each data reduction experiment we varied the simplification tolerance ε from 0.05 mile to 1 mile.

However, we cannot compare the reduction ratio between the line simplification and other data compression schemes directly. As we mentioned, only line simplification can provide deterministic error bounds a priori. For many other methods, we can only measure the maximum error and the average error for a given reduction ratio. To make the comparison fair, we also combined the two measures as one ratio, producing a normalized figure of merit $FOM = rr \times ie$. Where ie is the integral error (IE) between the original trajectory and the simplified trajectory with respect to the distance function used in the simplification.

All experiments reported were performed on a Pentium III 866MHz machine with 512MB of SDRAM main memory, running on Suse Linux.

Table 2 The average MSRE and E_u errors for varying compression ratios

	MSRE		E_u	
Ratios	DP	Wavelet	DP	Wavelet
D1 trajectories				
1%	0.0267	0.0410	≈0.098	0.522
2%	0.0124	0.0203	≈0.051	0.272
5%	0.0032	0.0076	≈0.013	0.168
10%	0.0011	0.0035	≈0.004	0.037
D2 trajectories				
1%	0.7182	1.917	≈2.542	77.61
2%	0.3045	0.944	≈1.036	23.65
5%	0.1599	0.363	≈0.564	4.29
10 %	0.0984	0.176	≈0.350	1.97

6.2 Experimental results

6.2.1 Line simplification vs. wavelet

First, we compared the errors of wavelets and line simplification as measured by the MSRE and E_u distances. It is important to observe that wavelets do not provide a bound on the error, but we measured the error for every compression ratio obtained by wavelets. We used the Haar wavelets variant [16] and the DP algorithm in the comparison. The wavelets method has been shown to outperform others such as DFT and DCT [16, 25]. The results of this comparison are shown in Table 2. It shows that the error of the DP algorithm is consistently lower than that of wavelets. According to the MSRE, DP is at most 65% of wavelet, and according to E_u it is at most 20%.

Figure 10 shows the average figures of merit as the functions of simplification tolerance or maximum error of wavelet approximation. One can observe that the DP algorithm nearly outperforms the Haar wavelet everywhere except the tolerance region between 0 and 0.3 for the future trajectory dataset. However, a close look at the reduction ratios in that region reveals that the Haar wavelet can only reduce the trajectory size to 45–88%. On the contrary, the DP simplification reduces the trajectories to 10–50%. The haar wavelet has better FOM in that region just because it nearly does not reduce those future trajectories.

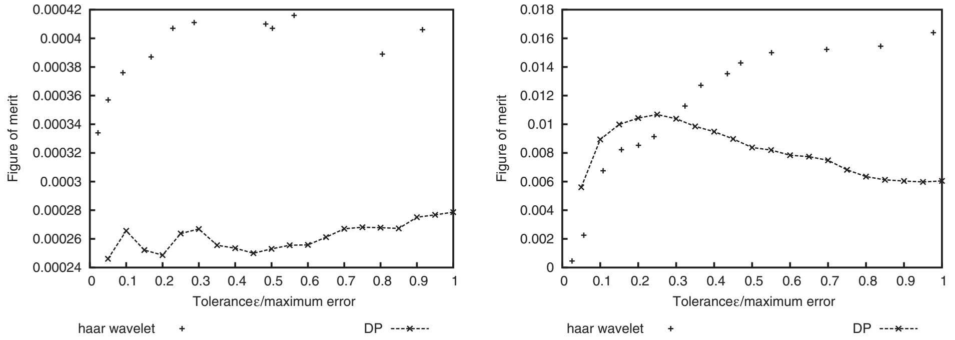
6.2.2 Douglas–Peucker vs. optimal algorithms

An optimal simplification algorithm, regardless of the distance functions chosen, yields the most concise simplifications for trajectories. However, in practice, one is also interested in the performance (running time) aspect. Using our datasets, we compared the savings and the performance of the optimal algorithms and the DP algorithm, with E_2 and E_u, the lower bound distance and upper bound distance on savings in Corollary 1. We used the algorithm of Chan and Chin(CC) as given in [10] in the performance comparison, which is the fastest one for the E_2 distance.

Figure 11 presents the comparisons of the average savings (reduction in size ratio of original/simplified trajectory)

[14] Avenue is the programming environment of the ArcView GIS.

[15] For more information about UCLA shuttle-bus trajectories, please visit http://www.cs.ucla.edu/ cjlai/bustrack/



(a) D1 Trajectories

(b) D2 trajectories

Fig. 10 The average figures of merit(FOM) of the Haar wavelet and the DP line simplification. E_u distance is used in the comparison

obtained by each of the algorithms. We have observed that the biggest difference between the savings generated by the two algorithms is only 6.75%, when the distance is E_u and tolerance is $\varepsilon = 0.1$ mile for the D2 trajectories.

Next, we compared the time-performance of the optimal and the DP algorithms. The time complexities of these algorithms for the various distances are given in Table 3 (algorithms for E_2, E_3, and E_u were studied more extensively in the literature, and therefore the time complexities compared with that of the generic algorithm were improved). The DP algorithm has better performance asymptotically in all cases. We have also experimentally compared the running time for the optimal and DP algorithms for the E_2 distance function (for the other distances, the optimal algorithms will fall farther behind the DP heuristic). The experiments were carried out on the D1 trajectories dataset and the results were averaged over all trajectories. The experimental results are summarized in Table 4. As the results indicate, the DP algorithm is between 8546 and 67,846 times faster than the optimal algorithm, with the advantage of DP increasing with the tolerance ε. For the D2 trajectories, the DP algorithm is still at least 35–444 times faster, according to the given tolerances.

In conclusion, the storage savings of both methods are close, but the DP algorithm is much faster. For some trajectories (i.e., the GPS traces in the UCLA bus dataset), the DP algorithm can even be approximately 10^4 times faster than the optimal algorithm.

6.2.3 Reduction performance of distance functions

We also investigated the nature of the savings of each of the four distances E_2, E_3, E_u, and E_t[16] and the combined distance $E_u \wedge E_t$. Figure 12 presents the *average* size of the reduced trajectory as a percentage of that of the original average trajectory, for each one of the distances. First, for all distances, the reduction ratio monotonically decreases as the value of ε increases. One can also observe that Corollary 1 is quantified by the experiments. Namely, E_2 has more savings than E_3, and E_3 has more savings than E_u, for the same simplification tolerance. Additionally, observe that each of E_u and E_t alone reduces the trajectories more than $E_t \wedge E_u$ for the same tolerance (as indicated by Property 2).

Table 3 Time complexities of the DP algorithm and the optimal algorithms

	DP	Optimal
E_2	$O(n \log n)$	$O(n^2)$
E_u and E_3	$O(n^2)$	$O(n^2 \log n)$
generic(arbitrary distance)	$O(n^2)$	$O(n^3)$

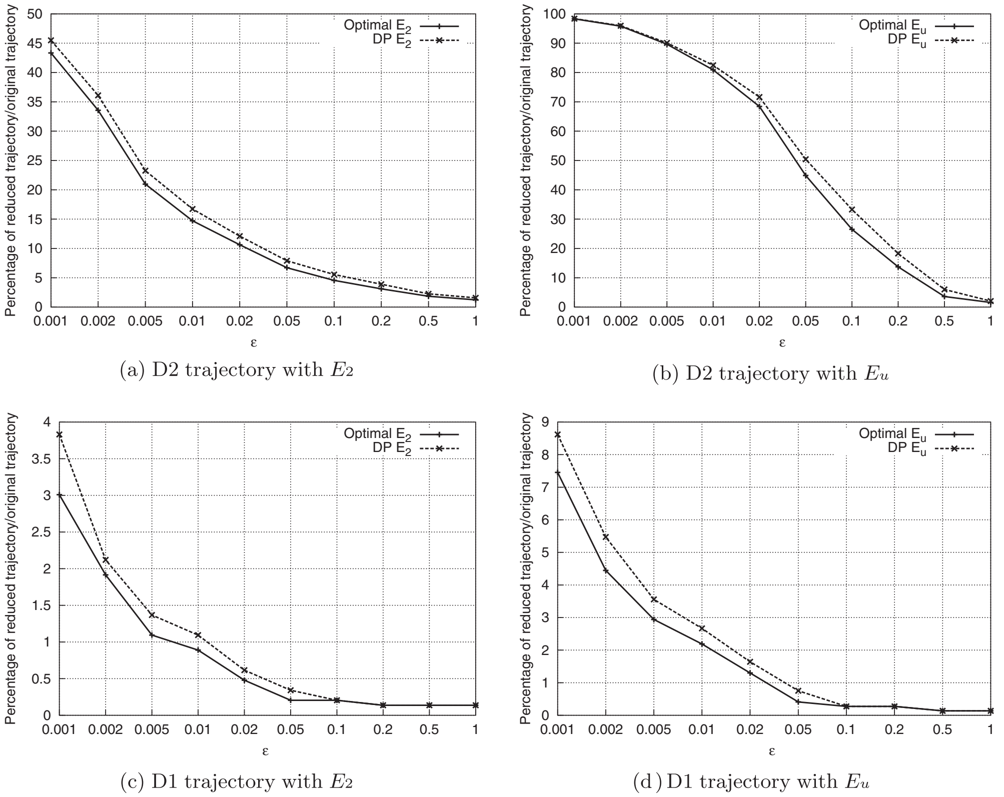
Another observation concerns the comparison of distance functions in terms of their data reduction. For DP on the D1 trajectories, E_u achieves a data reduction which is five to six times higher than E_t and $E_u \wedge E_t$ for the same simplification tolerance (see Fig. 12(a)). For the other cases, the differences are much smaller.

For the D1 trajectories(c.f. Fig. 12(a) and (b)), in all the cases, the reduction obtained by the optimal algorithm can be several times higher than the reduction of the DP heuristic, with the exact value depending on the distance and the tolerance. However, the savings of both methods is over 90%. Another observation is that the DP heuristic does not reduce the data as much as the E_t and the $E_u \wedge E_t$ distances. Specifically, we see that for the optimal algorithm the savings for these distances ranges from 98.4–99.6% depending on the tolerance ε. However, for the DP heuristic, this range is 90.5–94%.

Table 4 Comparing the per trajectory average running time of the optimal(OP) and the DP algorithm(DP), using E_2

ε(mile)	0.05	0.1	0.2	0.5	1
D1 trajectories					
OP(ms)	13118	32228	42768	42625	42336
DP(ms)	1.535	1.136	0.63	0.641	0.624
D2 trajectories					
OP(ms)	35.899	47.347	65.980	116.978	286.273
DP(ms)	1.018	0.935	0.862	0.739	0.644

[16] In order to plot E_t on the same graph as the other distances, the tolerances of E_t are normalized to distance by dividing the time-tolerance ε by the average speed of the trajectory.



(a) D2 trajectory with E_2

(b) D2 trajectory with E_u

(c) D1 trajectory with E_2

(d) D1 trajectory with E_u

Fig. 11 Comparison of the DP algorithm and the optimal algorithm on reduction savings

Fig. 12(c) and Fig. 12(d) depict the reduction ratios for the D2 trajectory dataset. Similar reduction patterns to those for the D1 trajectories are observed. Due to the "sparse" nature of the D2 trajectories (they only use one trajectory vertex to describe the motion on a street block), the savings is not as large as those for the D1 trajectories. To achieve significant savings, for example 9/10 of the original trajectories, one has to select a tolerance which is between 0.3 miles to 0.4 miles.

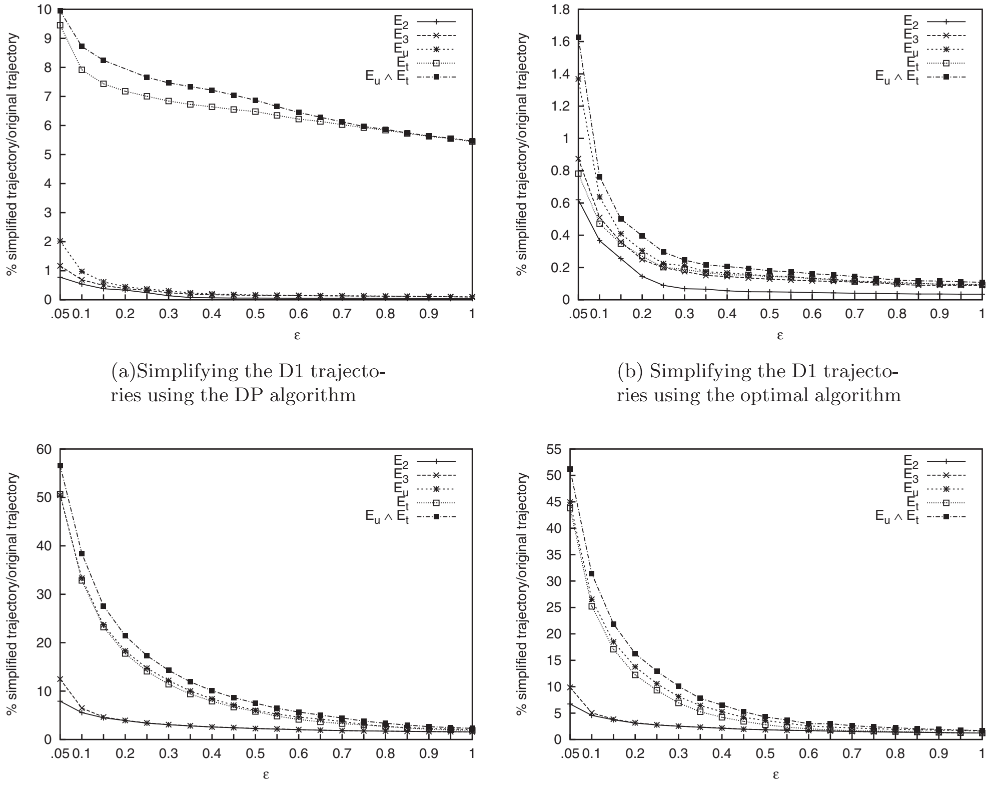
7 Related work

Line simplification has been well-studied from various perspectives: geographic information systems [11, 14]; digital image analysis [26]; and computational geometry [9, 10]. There are two variants of the problem: (1). *min-# problem*—given a tolerance ε, compute an approximation of original polygonal chain (polyline) C, with smallest number of vertices k_{min}; and (2). *min-ε problem*—given a number of vertices k (for the *reduced* polyline), compute an approximation of the original polyline C with at most k vertices and minimal error ε_{min}. Our approach to the trajectory reduction is a min-# problem, since our goal was to to obtain a reduction which ensures a bound on the error of the answer to the important spatio-temporal queries for all the trajectories in a moving objects database.

Most of the works on line simplification [9, 10] follow the graph-theoretic approach, as introduced by Imai and Iri [13]. The optimal algorithms for simplifying 2D polygonal chains run in $O(n^2)$ time for any Euclidian metric ($O(n^{4/3+\delta})$ for L_1 and L_∞ metrics [9]). As we have demonstrated, for our problem domain Douglas–Peuker algorithm produces simplification results which are very close to the optimal ones [10] except for E_t, and it has much better running time.

Unlike previous work, we study line simplification from the data management perspective for the spatio-temporal domain. We focus on the impact that the simplification has on the errors to query answers, due to combining the spatial and temporal domains. Thus, we propose the concept of soundness of the data reduction mechanism, and we analyze the soundness of several (distance-function, query-type) combinations. Additionally, we analyzed the data reduction power of line simplification on trajectories of moving objects.

Data compression is a very popular topic in the database research (e.g., [27, 28]). The techniques are targeted toward reduced storage requirements and improved I/O performance. When it comes to generating the answers to the queries, there are two main categories of approaches: 1. The data is decompressed when answering a query [29]; and 2. The compressed data is used to answer the query, and the answer contains some errors [15, 30, 31]. Our approach is *lossy* (i.e., we do not recover the original trajectories after simplification), and we aim at utilizing the reduced/simplified trajectories to get a faster response. Thus,



(a)Simplifying the D1 trajectories using the DP algorithm

(b) Simplifying the D1 trajectories using the optimal algorithm

(c)Simplifying the D2 trajectories using the DP algorithm

(d) Simplifying the D2 trajectories using the optimal algorithm

Fig. 12 The reduction ratio with different tolerances

our results cannot be directly compared to the first category of works above, which decompress the data when answering the queries. As an example of the second category, *wavelets* have recently become a popular paradigm for data reduction which provides fast and "reasonably approximate" answer to queries [15, 31]. The original data is reduced to compact sets of coefficients (*wavelet synopses*) which are used to answer the queries. The main difference with our approach is that these works either do not ensure a bound on the error of the query answers, or ensure an *asymptotic*/*probabilistic* bound on the error. Similar observation holds for the works which use *histograms* or *sampling* to compress the data and provide a reasonably accurate answer to the queries ([16] provides a survey of several data reduction techniques). In contrast, in our approach we address the *min-#* problem, but we ensure a *deterministic* bound on the error of the answers to spatio-temporal queries. To the best of our knowledge, no existing papers have done so.

A good survey on location modeling is provided in [32]. Moving objects databases have actively been studied from several aspects: 1. *modeling and querying* [33–36]; 2. *indexing* in primal or dual space [37, 38]; 3. *uncertainty* and its impact on the queries [17, 39]. A recent collection of articles addressing various issues in spatio-temporal databases, also providing a very comprehensive list of references, is presented in [40]. However, to the best of our knowledge, none of these works addressed the issue of simplification from the aspect of storage and processing savings. As we have demonstrated, the uncertainty can be related to the error introduced by the simplification and some of the results [17] can be used to answer spatio-temporal range queries in our context.

8 Conclusions

In this paper we addressed the problem of spatio-temporal data reduction, particularly the reduction of sets of (x, y, t) records aggregated into trajectories. The data reduction is by line simplification, a technique that guarantees bounds on the error of the approximated trajectories. Experimental results have shown that the storage-savings using line simplification is very significant. The bounded-error approximation may produce answers to queries for which the error is unbounded. In other words, even though the approximation is bounded-error, there are query types that when posed on this approximation produce answers whose error is unbounded. It turns out that the type of approximations for which this undesirable phenomenon, called unsoundness, arises depends on the distance used to approximate the trajectories and the type of spatio-temporal query. For example, the Euclidean distances (in two and three dimensions) are unsound for the query that asks "where is a particular moving object at a given time," i.e., the query, when posed on the simplified trajectory, may produce an answer which is arbitrarily far from the answer to the same query posed on the original trajectory. In our opinion, soundness is a new important concept in database research. This paper provides a classification of (approximation-distance, query-type) pairs into sound and unsound sets.

Next, we discussed uncertain queries on the database of approximate trajectories. The uncertainty arises since the query pertains to the original trajectories, but is processed on the approximations. We adapted a set of uncertain spatio-temporal range queries introduced previously (see [17]) for this purpose. It turns out that the algorithms provided in [17] work without any change for the range query with the distance function E_u. Uncertainty operators for other types of queries is the subject of future work.

We also discussed the aging of trajectories, namely producing increasingly compact (but also coarser) approximations of trajectories over time. Here we discovered that the optimal simplification algorithm (i.e., the one that produces minimum-size trajectories for a given error bound) is "aging-unfriendly" in the sense that it cannot be naturally used in aging. In contrast, the DP heuristic, which provides good but not optimal approximations, is "friendly." This concept of aging-friendliness is explained and defined in Sect. 5, and we believe that it will also prove important in other types of approximations. This is the subject of future work.

Finally, we compared experimentally the performance of the optimal algorithm versus the DP heuristic. We have shown that both achieve close storage savings, although the data-reduction obtained by the optimal algorithm is several times better. The exact storage saving of each algorithm depends on the distance, on the approximation-error tolerance, and the type of trajectories. However, experimental results show that the DP heuristic on trajectories with thousands of (x, y, t) records is also thousands of time faster than the optimal algorithm. We have also shown that savings of line simplification outperforms wavelets.

There are several immediate extensions of our present work. Although we investigated the impact of the uncertainty in the sense of imprecision introduced by the simplification process, it is interesting to investigate the issues that arise when the uncertainty of the location technology (e.g., GPS) is combined with the one due to the simplification. Another line of research is how to combine the simplification concept with the known constraints of the existing road networks, and what is the role of the uncertainty in such settings (c.f. [41, 42]). Additionaly, a challenging issue is online simplification, i.e., applying line-simplification before the whole trajectory becomes available. This is another important topic of future work.

Acknowledgements We thank the referees and the editor for their insightful comments.

References

1. Hage, C., Jensen, C.S., Pedersen, T.B., Speicys, L., Timko, I.: Integrated data management for mobile services in the real world. In: Proceedings of the 26th International Conference on Very Large Data Bases, pp. 1019–1030 (2003)
2. Schiller, J., Voisard, A. (eds.): Location-Based Services. Morgan Kaufmann (2004)
3. Wolfson, O.: Moving objects information management: The database challenge. In: Proceedings of the 5th Workshop on Next Generation Information Technologies and Systems (NGITS'2002) (2002)
4. Hightower, J., Borriella, G.: Location systems for ubiquitous computing. IEEE Comput. **34**(8), 57–66 (2001)
5. Snyder, J.P., Tobler, W.R., Yang, O.H., Yang, Q.H.: Map Projection Transformations: Principles and Applications. CRC, Boca Raton, FL (2000)
6. Arctur, D., Zeiler, M.: Designing Geodatabases: Case Studies in GIS Data Modeling
7. Samarati, P., Sweeney, L.: Generalizing data to provide anonymity when disclosing information. In: PODS '98: Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, p. 188 (1998)
8. Veijalainen, J., Ojanen, E., Haq, M.A., Vahteala, V.-P., Matsumoto, M.: Energy consumption tradeoffs for compressed wireless data at a mobile terminal. IEICE Trans., Spec. Issu. Multimedia Commun. **E87-B**(5), 1123–1130 (2004)
9. Agarwal, P.K., Varadarajan, K.R.: Efficient algorithms for approximating polygonal chains. Discrete Comput. Geom. **23**, 273–291 (2000)
10. Chan, W., Chin, F.: Approximation of polygonal curves with minimum number of line segments or minimum error. Int. J. Comput. Geom. Appl. **6**, 50–77 (1996)
11. Douglas, D., Peuker, T.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Can. Cartographer **10**(2), 112–122 (1973)
12. Hershberger, J., Snoeyink, J.: Speeding up the Douglas-Peucker line-simplification algorithm. In: The 5th International Symposium on Spatial Data Handling (1992)
13. Imai, H., Iri, M.: Polygonal approximations of a curve-formulations and algorithms. In: Comput. Morphol., pp. 71–86 (1988)
14. McMaster, R.: Automated line generalization. Cartographica **24**(2), 74–111 (1987)
15. Garofalakis, M., Gibbons, P.B.: Wavelet synopses with error guarantees. In: Proceedings of ACM SIGMOD, pp. 476–487 (2002)
16. Special issue on data reduction techniques. IEEE Data Eng. **20** (1998)
17. Trajcevski, G., Wolfson, O., Hinrichs, K., Chamberlain, S.: Managing uncertainty in moving objects databases. ACM Trans. Database Syst. (TODS) **29**(3), 463–507 (2004)
18. Greenfeld, J.S.: Matching GPS observations to locations on a digital map. In: The 81th Annual Meeting of the Transportation Research Board. Washington, DC (2002)
19. White, C.E., Bernstein, D., Kornhauser, A.L.: Some map matching algorithms for personal navigation assistants. Transp. Res. Part C **8**, 91–108 (2000)
20. Pfoser, D., Jensen, C.S., Theodoridis, Y.: Novel approaches in query processing for moving object trajectories. In: Proceedings of the 26th International Conference on Very Large Data Bases, pp. 395–406 (2000)
21. Vlachos, M., Kollios, G., Gunopulos, D.: Discovering similar multi-dimensional trajectories. In: Proceedings of the 18th International Conference on Data Engineering, pp. 673–684. San Jose, CA (2002)
22. Alt, H., Guibas, L.J.: Discrete geometric shapes: Matching, interpolation, and approximation A survey. Technical Report B 96-11 (1996)

23. Weibel, R.: Generalization of spatial data: Principles and selected algorithms. In: van Kreveld, M., Nievergelt, J., Roos, T., Widmayer, P. (eds.) Algorithmic Foundations of Geographic Information Systems. LNCS Springer-Verlag, Berlin Heidelberg New York (1998)
24. Veltkamp, R.C.: Hierarchical approximation and localization. Vis. Comput. **14**(10), 471–487 (1998)
25. Hardle, V., Kerkyacharian, G., Picard, D., Tsybakov, A.: Wavelets, Approximation, and Statistical Applications. Springer, Berlin Heidelberg New York (1998)
26. Hobby, J.D.: Polygonal approximations that minimize the number of inflections. In: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 93–102 (1993)
27. Graefe, G., Shapiro, L.D.: Data compression and database performance. In: Proceedings of the ACM/IEEE-CS Symposium on Applied Computing (1991)
28. Westmann, T., Kossmann, D., Helmer, S., Moerkotte, G.: The implementation and performance of compressed databases. SIGMOD Rec. **29**(3), 55–67 (2000)
29. Chen, Z., Gehrke, J., Korn, F.: Query optimization in compressed database systems. In: Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 271–282. ACM (2001)
30. Gibbons, P.B., Matias, Y., Poosala, V.: Fast incremental maintenance of approximate histograms. ACM Trans. Database Syst. **27**(3), 261–298 (2002)
31. Chakrabarti, K., Garofalakis, M., Rastogi, R., Shim, K.: Approximate query processing using wavelets. VLDB J. **10**(2–3), 199–223 (2001)
32. Pitoura, E., Samaras, G.: Locating objects in mobile computing. IEEE Trans. Knowledge Data Eng. **13**(4), 571–592 (2001)
33. Florizzi, L., Gutting, R.H., Nardelli, E., Schneider, M.: A data model and data structures for moving objects databases. SIGMOD Rec. **29**, 319–330 (2000)
34. Güting, R.H., Böhlen, M.H., Erwig, M., Jensen, C.S., Lorentzos, N., Nardeli, E., Schneider, M., Viqueira, J.R.R.: Spatio-temporal models and languages: An approach based on data types. In: Spatio-Temporal Databases: The Chorochronos Approach (2003)
35. Lema, J.A.C., Forlizzi, L., Güting, R.H., Nardeli, E., Schneider, M.: Algorithms for moving objects databases. Comput. J. **46**(6), 680–712 (2003)
36. Vazirgiannis, M., Wolfson, O.: A spatio-temporal model and language for moving objects on road networks. In: SSTD ’01: Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases, pp. 20–35 (2001)
37. Agarwal, P.K., Arge, L., Erickson, J.: Indexing moving points. J. Comput. Syst. Sci. **66**(1), 207–243 (2003)
38. Kollios, D., Gunopulos, D., Tsotras, V.J.: On indexing mobile objects. In: Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 261–272 (1999)
39. Pfoser, D., Jensen, C.: Capturing the uncertainty of moving objects representation. In: Advances in Spatial Databases, 6th International Symposium, SSD’99, pp. 111–132 (1999)
40. Koubarakis, M., Sellis, T., Frank, A.U., Grumbach, S., Güting, R.H., Jensen, C.S., Lorentzos, N., Manolopoulos, Y., Nardelli, E., Pernici, B., Scheck, H.-J., Scholl, M., Theodoulidis, B., Tryfona, N. (eds.): Spatio-Temporal Databases—The CHOROCHRONOS Approach. Springer-Verlag, Berlin Heidelberg New York (2003)
41. Ding, Z., Güting, R.H.: Managing moving objects on dynamic transportation networks. In: International Conference on Scientific and Statistical Database Management (SSDB), pp. 287–296 (2004)
42. Ding, Z., Güting, R.H.: Uncertainty management for networks-constrained moving objects. In: International Conference on Database and Expert Systems Applications (DEXA), pp. 411–421 (2004)