Nonmaterialized Motion Information in Transport Networks

Hu Cao and Ouri Wolfson

Department of Computer Science,
University of Illinois at Chicago,
851 S. Morgan Street, Chicago, IL, 60607, USA
{hcao2, wolfson}@cs.uic.edu

Abstract. The traditional way of representing motion in 3D space-time uses a trajectory, i.e. a sequence of (x,y,t) points. Such a trajectory may be produced by periodic sampling of a Global Positioning System (GPS) receiver. The are two problems with this representation of motion. First, imprecision due to errors (e.g. GPS receivers often produce off-the-road locations), and second, space complexity due to a large number of samplings. We examine an alternative representation, called a nonmaterialized trajectory, which addresses both problems by taking advantage of the a priori knowledge that the motion occurs on a transport network.

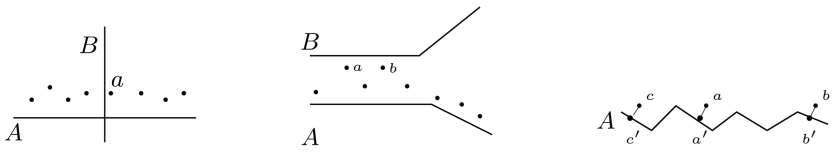
1 Introduction

Location management, i.e. the management of transient location information, is an enabling technology for location based service applications. It is also a fundamental component of other technologies such as fly-through visualization (the visualized terrain changes continuously with the location of the user), context awareness (location of the user determines the content, format, or timing of information delivered), augmented reality (location of both the viewer and the viewed object determines the type of information delivered to viewer), and cellular communication.

Usually, locations of a moving object are obtained by sensors and are given as a set of spatio-temporal points of the form (x, y, t). Such a point indicates that a moving object m (represented as a 2-dimensional point) was at 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, RFID).

The first problem arising in location management is that GPS receivers are imprecise, and thus this raw data is noisy and error prone. Indeed, a data point of a typical GPS receiver has an error that ranges from several feet to tens of meters. In most cases, the motion occurs on a road network, and thus the error of a GPS point can be corrected by "snapping" the point onto the road network. This correction is very important for many natural queries such as "retrieve the

T. Eiter and L. Libkin (Eds.): ICDT 2005, LNCS 3363, pp. 173–188, 2005.
© Springer-Verlag Berlin Heidelberg 2005



(a) Example 1 (b) Example 2 (c) Example 3

Fig. 1. Figure 1(a) and Figure 1(b) illustrate the problem with naive snapping; Figure 1(c) illustrates space saving of nonmaterialized trajectory.

number of vehicles that traveled on the highway between exits 48 and 52 of I80 in the last hour". Such a query is impossible to answer precisely if the locations of moving objects are off-the-road, since vehicles traveling on parallel roads may seem to have traveled on the highway, and vice versa. Similarly, the query " What is the route taken by Bill today" requires translation of motion from raw GPS data into a higher level of abstraction.

One may be tempted to propose a simple solution to the error-correction problem, namely snap each GPS point to the closest road segment. However this is a simplistic solution that may produce incorrect results. For example, consider Figure 1(a) that illustrates a road network, and several GPS points. Clearly the vehicle traveled on road segment A, and thus GPS point a needs to be snapped to A, although B is the closest road segment to a. Another example is shown in Figure 1(b). Clearly the vehicle traveled on road A, but this is deduced only by examining the whole sequence of GPS points, and snapping GPS points a, b onto the closest road segment will produce an incorrect result.

As a first result of this paper, we propose an efficient algorithm that, given a trajectory T of a moving object[1], a road network, and a GPS receiver error bound ε, determines whether there exists another trajectory T', called the *road-snapped trajectory*, such that: (i) T' is on the road network, and (ii) the distance between T and T' is not higher than ε; and if so it finds T'. In other words, we find for a trajectory T, a possible route in the road-network that was followed by the moving object.

The second problem addressed in this paper is data volume. In principle, a GPS receiver can generate a new (x, y, t) point every second, and the number of moving objects may be hundreds of millions to billions. The problem is compounded by the fact that one is interested in historical spatio-temporal information for data mining.

Now consider that usually computation of the location at any point in time is enabled by linear interpolation between consecutive trajectory vertices[7, 14]. Then the road-snapped trajectory may have more points than the original trajectory, since it contains the snapped vertices of T, as well as the vertices of the

[1] The trajectory of a moving object is a polygonal line in 3D that represents a piecewise linear function from time to location in two-dimensional space; the GPS points are the vertices of the polygonal line. Thus the trajectory models a trip.

route. For example, if the two consecutive GPS points a and b of Figure 1c are snapped onto the depicted road, the trajectory between a' and b' will consist of six vertices rather than two.

Thus one can immediately recognize the storage-space problem that location based services applications will face, as well as the computation burden for processing such large amount of information. Additionally, in online tracking where the spatio-temporal points are transmitted from a moving object to a server, this storage problem translates into a bandwidth and power problem.

Our nonmaterialized trajectory concept addresses this problem by separating the motion description into two components, namely the spatial component represented by the road network (i.e. the map) that is common to all the trajectories, and the temporal component that is specific to each trajectory. So, for example, assuming constant speed motion, the nonmaterialized trajectory representing the motion of Figure 1(c) consists of the street A and two time-points, the time at a' and the time at b' (rather than six points used by the materialized representation). Together with the coordinates of street A given by the map, this nonmaterialized trajectory can provide the location of the moving object at any point in time. And actually, the nonmaterialized trajectory has even fewer points because it is a bounded error approximation of the original trajectory. So, for example, assume that c' precedes a', and consider the nonmaterialized trajectory T': "on street A, at time point t_c at location c' and at time point t_b at location b'". If the distance of T' from the original trajectory is not higher than ε, then a' can be eliminated from the nonmaterialized trajectory. Thus the nonmaterialized representation is an abstraction that is concise because it encapsulates two mechanisms, namely: separation of the temporal component from the spatial one, and bounded error approximation. Obviously, the map will also need to be kept. However, the same map is shared among many trajectories.

The concept of a nonmaterialized trajectory can be demonstrated by the following analogy. When giving driving directions, one could indicate: starting from (x_1, y_1) drive in a straight line to (x_2, y_2), from there drive in a straight line to (x_3, y_3), etc. This would be analogous to a regular, i.e. materialized trajectory given as a function from time to space. Nobody uses this form of directions. Instead, directions are given as: drive on Halsted Street, make a left of Canal street, then make a right on Division street. This is equivalent to the concept of a nonmaterialized trajectory that gives the time $\rightarrow$ space function implicitly; using the map, the function can be made explicit.

As a second result of this paper, we provide an efficient algorithm that constructs a nonmaterialized trajectory T'' for a given road-snapped trajectory T' and an error bound ε, such that the distance between the original trajectory T and T'' is at most ε; furthermore, the size of T'' is minimum among all nonmaterialized trajectories that can be constructed based on T'. Why not find a minimum-size nonmaterialized trajectory that is at distance ε from the original trajectory T? We conjecture that this problem is NP-complete.

Then we analyze the errors to spatio-temporal queries introduced by the approximation, and we show that these errors are bounded. In other words,

the nonmaterialized trajectory T'' (which is also a road-snapped trajectory) is an approximation of the original trajectory T. What is the "distance" in the answers of a given spatio-temporal query posed to T and T''? We show in this paper that this distance is bounded for all natural spatio-temporal queries. One may be tempted to discount these results, on the ground that it is intuitively clear that if the error of the approximation is bounded (i.e. the distance between T and T'' is bounded), then the error of the answer to each query is also bounded. However, we show that this is not necessarily the case. Specifically, we show that for every ε and δ there exists a trajectory T with a road-snapped trajectory T' such that the Euclidean distance between T and T' is at most ε, but the distance between the answers to the query "where is the moving object at time 2pm" on T and T' is higher than δ. Similarly, the error to other natural spatio-temporal queries is unbounded in a sense made precise in this paper. The reason the our snapping algorithm produces error-bounded approximations is that it does not use the Euclidean distance, but another, called time_uniform distance.

Trajectory snapping is supposed to correct location sensing errors, so one may wonder why we are concerned about queries on the original trajectory T. The answer is that one may never be sure what the actual motion function was, and so we want to limit the damage in case the snapping is to an incorrect route.

In summary, **the main results of this paper are as follows**. First, we provide an efficient algorithm that, given a trajectory T, a road network, and an error bound ε, determines whether there exists an ε-distant road-snapped trajectory; and if so it finds it. Second, we provide an algorithm that, for a road-snapped trajectory T' and an original trajectory T and a bound ε, finds a nonmaterialized trajectory T'' that is at distance at most ε from T, and has minimum size among all nonmaterialized trajectories derived from T'. Third, we defined the notion of error boundness for spatio-temporal queries, and we show that the time_uniform distance used by our snapping algorithm is error-bounded with respect to the spatio-temporal query types: where is a moving object at a given time, range query, nearest neighbor, and join. We also show that the Euclidean distance is not error bounded w.r.t. these query types.

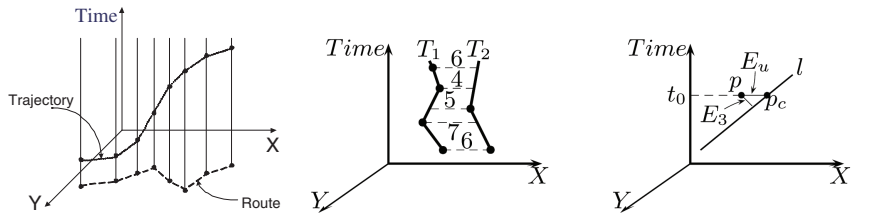
The rest of the paper is organized as follows. In section 2 we introduce the model. In sections 3, 4, and 5 we devise the first, second and third results discussed in the previous paragraph, respectively. In section 6 we compare our work to relevant literature, and in section 7 we conclude the paper.

2 The Model

Representing the *(location,time)* information of the moving object as a trajectory is a typical approach (c.f. [7, 14]). Point locations are represented as longitude-latitude (x, y)-coordinates. We do not discuss moving objects with a third altitude dimension, although our results can be extended to this case. Time is a real number t. Thus every *(location,time)* of a moving point object is given as a 3-dimensional (x, y, t) point. We do not discuss moving objects with an extent such as weather phenomena.

Definition 1. *A* trajectory *T is a piece-wise linear function from the time interval $[t_1, t_n]$ to (X, Y) space. It is represented by the vertices of the function polygonal-line $T_1 = (x_1, y_1, t_1)$, $T_2 = (x_2, y_2, t_2)$, ..., $T_n = (x_n, y_n, t_n)$, such that for all $i \in \{1, \ldots, n-1\}$, $t_i < t_{i+1}$. For a given trajectory T, its projection on the (X, Y) plane is called the* route *of T, denoted as $R(T)$. The* location *of T at time t is the value of the function at t.*

A trajectory (or a *materialized trajectory*) defines the location of a moving object in the (X, Y) plane as a function of time t. The vertices of the trajectory are the known locations (e.g. the GPS points), and the trajectory function is obtained by straight-forward linear interpolation between these locations. An illustration of trajectory and its route is shown in Figure 2(a). Observe that this representation cannot model nonlinear motion such as acceleration, but can approximate it with arbitrary precision given enough trajectory vertices.



(a) A trajectory and its 2D route.

(b) The distance between T_1 and T_2 is 7.

(c) Euclidean distance E_3 and time_uniform distance E_u.

Fig. 2. Distance between trajectories and distance functions

Next, for the purpose of trajectory snapping, we define a map. A *map*, or a *road network*, is a directed graph; the nodes are labeled by (X, Y) coordinates and represent junctions [2] and the arcs represent straight-line road segments between junctions. Each arc has a length, which is the Euclidean distance between its two endpoints. The arcs of the map are partitioned into *streets*, where each street is a path in the map, the streets are disjoint (i.e. we assume that 42nd-st-going-east and 42nd-st-going-west are different streets), and every arc of the map belongs to some street. We assume without loss of generality that each street is acyclic.

Every point location on the map (node or point on an arc) can be also be defined in the *linear reference system*, i.e. the distance from the beginning of a street. Thus a point-location on the map can be defined in the Cartesian system as an (x, y) location, or in the linear reference system as *(street, distance)*.

[2] A junction is not necessarily intersection of two streets, but maybe the vertex of a polyline representing the geometry of the road.

Our objective is to construct a trajectory on the map that is an ε-approximation of the original trajectory, i.e. at ε distance from the original trajectory. For this purpose we need to define the distance between two trajectories. The Hausdorff distance[2] between trajectories is defined as follows. Let M be the distance between a 3D point and the 3D straight line between two consecutive trajectory vertices. Examples of two possible M's, the Euclidean and the time_uniform, are given at the end of this section. The distance $d_M(p,T)$ between a 3D point p and a trajectory T is the minimum (among all straight line segments of T) M-distance between p and a line segment of T. The Hausdorff M-distance from a trajectory T to another trajectory T' is defined as $\tilde{D}_M(T,T') = \max_{p\in T} d(p,T')$, 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 (see Figure 2(b)).

Definition 2. *Given a trajectory T, a road network N, a tolerance $\varepsilon > 0$, and a distance M between a 3D point and a 3D line, the* ε_M-road-snapped *trajectory T' is a trajectory whose route is a path in the graph N, and $D_M(T,T') \leq \varepsilon$.*

In the above definition the tolerance ε is the sum of two maximum errors. One is the error of the location sensing device such as a GPS receiver, and the second is the error of the map. The ε-road-snapped trajectory is a possible actual trajectory of the moving object.

Now we take the inner distance function M to be the three dimensional *time_uniform* distance E_u defined as follows. For a point $p = (x_0, y_0, t_0)$ on one trajectory and a line segment l on the other, $E_u(p,l) = \sqrt{(x_0 - x_c)^2 + (y_0 - y_c)^2}$, where $p_c = (x_c, y_c, t_0)$ is the unique point on l which has the same *Time* value as p (see Figure 2(c)); if such a point does not exist, then the time_uniform distance between p and l is infinity.

Observe that the time_uniform distance is different than the Euclidean distance between p and l (see Figure 2(c)). Intuitively, the time_uniform distance between a trajectory point p and a trajectory line l is the distance between p and the point on l that has the same time as p. Whereas the Euclidean distance between p and l is the minimum distance between p and a point on l.

In section 5 we will discuss the Euclidean distance function, but until then we will always assume that M is the time_uniform distance and will omit M.

3 Road-Snapped Trajectory Construction

Assume that we are given a trajectory T, a map M, and a tolerance ε. In this section we devise an efficient algorithm that determines whether or not there exists an ε-road-snapped trajectory T'. If so, it finds it. The algorithm constructs a Snapping Configuration Graph (SCG), finds a certain path in SCG, and then extracts T' from this path.

The SCG construction uses the following definition. Given two polygonal lines in the (X, Y) coordinate system, the *ε-neighborhood* of a vertex p in one polygonal line is the set of 2D points on the other polygonal line that are at distance at most ε from p. The concept is illustrated in figure 4(b).

The snapping configuration graph is constructed as follows. For the trajectory T given as a sequence of vertices $T_1, \ldots, T_i, \ldots, T_n$, and a map with arcs $(l_1, \ldots, l_m)$, the nodes of SCG are of two types: (1) (T_i, l_j) for each trajectory vertex T_i and arc l_j for which the Euclidean distance between the 2D projection of T_i and l_j is at most ε. Intuitively, this indicates that T_i can be snapped onto l_j. And (2) $(l_j, \overrightarrow{T_i T_{i+1}})$, when the distance between the front end-point of l_j and the (X, Y) projection of $\overrightarrow{T_i T_{i+1}}$ is at most ε. Intuitively, this means that in the road-snapped trajectory, some point between T_i and T_{i+1} can be snapped onto the front endpoint of l_j.

The arcs of SCG indicate the possible pairwise sequences of individual nodes to construct a road-snapped trajectory. The arcs of SCG are of four types:

- (1) $(T_i, l_j) \rightarrow (T_{i+1}, l_j)$, if the ε-neighborhood of the 2D projection of T_{i+1} on directed line segment l_j is not totally behind (in l_j) the ε-neighborhood of the 2D projection of T_i on l_j. Intuitively, this arc indicates that if T_i is snapped onto l_j, then T_{i+1} can be snapped onto l_j as well. Observe that this can be done only if "not totally behind" restriction is satisfied. In other words, there must be a point p of l_j that is in the neighborhood of T_{i+1}; and p appears on l_j before another point q of l_j that is in the neighborhood of T_i.
- (2) $(T_i, l_j) \rightarrow (l_j, \overrightarrow{T_i T_{i+1}})$. Intuitively, this arc means that if one vertex of the road-snapped trajectory is T_i snapped onto l_j; then the next vertex of the road-snapped trajectory can be the point of the trajectory between T_i and T_{i+1} that is snapped onto the front endpoint of l_j. In this case, T_{i+1} is snapped onto another line segment of the map.
- (3) $(l_j, \overrightarrow{T_i T_{i+1}}) \rightarrow (T_{i+1}, l_{j'})$, where $l_{j'}$ is one of the adjacent arcs that follows l_j in the map. Intuitively, this arc means that if one vertex of the road-snapped trajectory is some point of the trajectory between T_i and T_{i+1} that is snapped onto the front endpoint of l_j; then the next vertex of the road-snapped trajectory can be T_{i+1} that is snapped onto an arc of the map that is adjacent to l_j.
- (4) $(l_j, \overrightarrow{T_i T_{i+1}}) \rightarrow (l_{j'}, \overrightarrow{T_i T_{i+1}})$, where $l_{j'}$ is one of the adjacent arcs that follows l_j in the map, and the ε-neighborhood of the front end-point of l'_j on the 2D projection $\overrightarrow{T_i T_{i+1}}$ is not totally behind that of l_j. Intuitively, this arc means that if one vertex of the road-snapped trajectory is some point of the trajectory between T_i and T_{i+1} that is snapped onto the front endpoint of l_j; then the following vertex can be be another point between T_i and T_{i+1} that is snapped onto the front endpoint of l'_j.

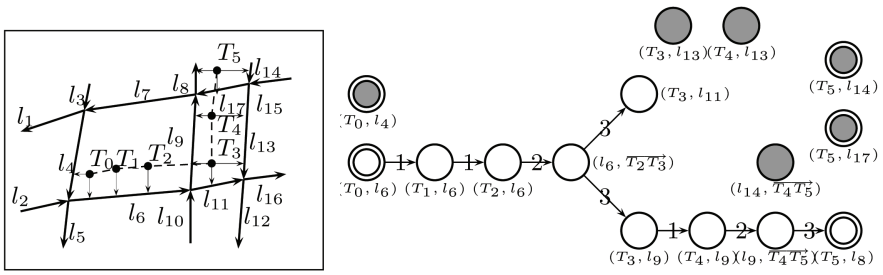
Intuitively, the four types of arcs represent four cases for adjacent vertices of the road-snapped trajectory TT, Tl, lT, ll; where T represents a vertex derived from the trajectory and l represents a vertex derived from the map.

Theorem 1. *Given a trajectory $T = T_1, \ldots, T_n$, a map M, and a tolerance ε, there exists an acyclic path π in SCG starting at a node (T_1, l) and ending at a node (T_n, l') if and only if there exists an ε-road-snapped trajectory T'.*

The above theorem provides the necessary and sufficient condition for the existence of an ε-road-snapped trajectory, and its proof[3] is constructive, i.e. it finds the trajectory. It is easy to see that the time complexity of the algorithm is $O(nm^2)$ for a trajectory with n vertices and a map with m arcs at distance ε from T. Observe that the complexity does not depend on the total number of arcs in the map, only on the ones that are at distance ε from the trajectory.

Assume now that there exists a trajectory T' whose route is on the map M, such that T' is at distance at most ε from T. A route R of such a trajectory is called a *feasible route* of T. Given the path π (see Theorem 1), its feasible route is: the set of arcs that appear in π, either in the first component of a node, or in the second. By construction of SCG, this is a path in the map. This procedure of constructing a SCG and finding a feasible route is illustrated in example 1.

Example 1. Consider the map M and the trajectory T shown in Figure 3(a). The map is drawn as a directed graph with 17 directed arcs $l_1, \ldots, l_{17}$. The trajectory consists of six trajectory vertices from T_0 to T_5, shown as the dashed polygonal line in Figure 3(a). The arrowed lines indicate that the corresponding vertex has an ε-neighborhood in the line segment to which it points.



(a) The road network and trajectory.

(b) The snapping configuration graph.

Fig. 3. Example of snapping configuration graph

Figure 3(b) depicts the snapping configuration graph generated from the map and the trajectory in Figure 3(a). There are 15 nodes in the SCG. The labels of the SCG arcs indicate the arc types. The connected nodes are colored white and the isolated ones are in gray. Especially observe that $(T_3, l_{13}) \rightarrow (T_4, l_{13})$ is not a valid arc since the ε-neighborhood of the 2D projection of T_4 on l_{13} is behind that of T_3. The nodes that pertain to the start trajectory vertex and the end trajectory vertex are illustrated by double circles. According to Theorem 1, a

[3] The proof and the proofs of other theorems are omitted, due to space constraint.

road-snapped trajectory of T exists since there is a path π from (T_0, l_6) to (T_5, l_8) and it is the only one in the SCG. Thus, we can extract a road-snapped trajectory T' with eight vertices $T'_0, T'_1, T'_2, \ldots, T'_7$ which correspond nodes (T_0, l_6) to (T_5, l_8) on π respectively. The feasible route R of T' is (l_6, l_9, l_8). □

4 Nonmaterialized Trajectory Construction

In this section we devise an algorithm that, given a path π in SCG, constructs a nonmaterialized ε-road-snapped trajectory T'' of minimum size. The route R of T'' is the feasible route of π and T'' has minimum size among all nonmaterialized trajectories on R.

We start with the definition of a nonmaterialized trajectory. Intuitively, a nonmaterialized trajectory describes the motion in the linear reference system. For example, started at 0.2 mile-post of Broadway-north (the mile-post simply indicates location from the beginning of the street) at 2pm and drove to the 3.2 mile-post, then turned on 42nd street-west at 2.2 mile-post at 2:10pm, etc. Formally, a nonmaterialized trajectory is defined as follows.

Definition 3. (Trajectory, Nonmaterialized) *Consider a map M consisting of a set of streets P. A* nonmaterialized trajectory *T is a function from time to map locations represented as a sequence of tuples $(\langle p_1, l_1, t_1\rangle, \ldots, \langle p_m, l_m, t_m\rangle)$, where each p_i is a street in P, l_i is a real number that indicates T's location at time t_i in p_i's linear reference coordinate. The location of T at any time-point between t_i and t_{i+1} is the linear interpolation between (l_i, t_i) and (l_{i+1}, t_{i+1}) along p_i. The nonmaterialized trajectory T must be consistent with the transport network N, in the following sense. For every two adjacent tuples (p_i, l_i, t_i) and $(p_{i+1}, l_{i+1}, t_{i+1})$ of T, if their streets are different, then p_i must intersect p_{i+1} at the distance l_{i+1} from the beginning of p_{i+1}.*

This concept is illustrated in example 2.

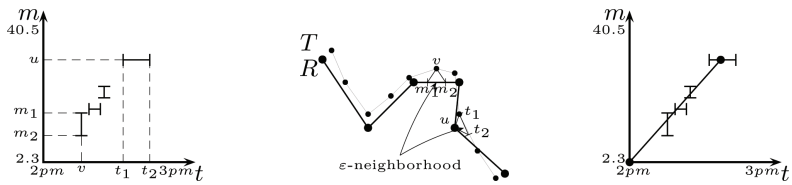
Example 2. What is the nonmaterialized view of trajectory T' in Example 1? Assume that we have constructed a road-snapped trajectory T' from the path π and the route $R = (l_6, l_9, l_8)$. Further assume that R is on two streets S_1 and S_2, where l_6 is on S_1 from the 3.2 mile-post to 3.6 mile-post, l_9 is on S_2 from the 0.3 mile-post to the 1 mile-post, and l_8 is on S_2 from the 1 mile-post to the 1.4 mile-post. The first vertex T'_0 of T' is on l_6 at the 0.1 mile-post from the intersection of l_2 and l_6 at 1:00pm. The second vertex T'_1 is on l_6 at the 0.2 mile-post from the intersection of l_2 and l_6 at 1:01pm. The third vertex T'_2 is on l_6 at the 0.3 mile-post from the intersection of l_2 and l_6 at 1:03pm. The fourth vertex T'_3 is at the intersection of l_6 and l_9 at 1:06pm. The fifth vertex T'_4 is the snapping of T_3 on l_9 at the 0.2 mile-post from the intersection of l_6 and l_9 at 1:09pm. The sixth vertex T'_5 is the snapping of T_4 on l_9 at the 0.5 mile point from the intersection of l_6 and l_9 at 1:11pm . The seventh vertex T'_6 is at the intersection of l_9 and l_8 at 1:14pm. The last vertex T'_7 is on l_8 at the 0.2 mile-post from intersection of l_9 and l_8 at 1:16pm . Then, the nonmaterialized representation

of T' is the sequence (S_1, 3.3, 1:00pm), (S_1, 3.4, 1:01pm), (S_1, 3.5, 1:03pm), (S_2, 0.3, 1:06pm), (S_2, 0.5, 1:09pm), (S_2, 0.8, 1:11pm),(S_2, 1, 1:14pm),(S_2, 1.2, 1:16pm). Note that the size of this representation is eight, longer than the size of the original trajectory, because, as mentioned in the introduction, it represents both the vertices of trajectory and the road network. □

A nonmaterialized trajectory can easily be transformed to the equivalent materialized representation, in linear time, by traversing, in sequence, the tuples of the nonmaterialized representation, and for each one, interpolating the arrival time at every vertex of the route. Similarly, one can transform in linear time a road-snapped materialized trajectory T into a nonmaterialized one by creating a nonmaterialized tuple for each vertex of T.

The nonmaterialized trajectory is created based on the feasible route R and the SCG path π found in the previous section. The number of tuples in the nonmaterialized trajectory is minimum for the path π.

The Nonmaterialized Trajectory Construction (NTC) algorithm starts with a feasible route R and a trajectory T and constructs the nonmaterialized trajectory for each street S of R, i.e it works one street at a time, starting from the first to the last. If the same street appears on the feasible route more than once, then the procedure is repeated for each occurrence of the street on R. For ease of exposition assume that the feasible path consists of a single street S.



(a) ε-neighborhood bars (b) (x, y) space (c) Nonmaterialized trajectory

Fig. 4. Nonmaterialized trajectory construction in the linear reference/time space

The NTC algorithm constructs the minimum size nonmaterialized trajectory in the (m, t) two-dimensional space. It uses the following result.

Theorem 2. *If there exists an acyclic path π in SCG starting at a node (T_1, l) and ending at a node (T_n, l'), then each vertex T_i of the trajectory T appears in π as the first component of a node, at most once.*

The procedure is as follows. In the (m, t) space the t axis is the time linear reference built based on trajectory T, and the m axis is the street linear reference. So if the trajectory starts at 2pm and ends at 3pm the t axis has these endpoints. And if the feasible route starts at the 2.3 milepost of S and ends at the 40.5 mile-post, the m axis has 2.3 and 40.5 as the endpoints (see Fig. 4). Observe that since each street is acyclic, each arc of the map appears in m at most once.

NTC then constructs an m-(vertical) line segment for each vertex v of T, and a t-(horizontal) line segment for each arc u represented in m (Fig. 4(a)).

The vertical line segment for v is constructed as follows. According to Theorems 1 and 2 there is exactly one node (v, l) in the SCG path π, and l is represented in m at most once. If l is represented in m, then consider the ε-neighborhood on l of the projection of v on the (X, Y) space. Assume that this neighborhood in the m coordinate is (m_1,m_2). Then we draw the vertical bar (m_1,m_2) at the time corresponding to v on t (see Figure 4(a)).

The horizontal line segment for the arc u is constructed as follows. Since there is a single street in the feasible path, in the SCG path π there is exactly one node $(u, \overrightarrow{T_i T_{i+1}})$. Then, in the (X, Y) space, compute the ε-neighborhood of the front-end point of u on the projection of $\overrightarrow{T_i T_{i+1}}$. Assume that this neighborhood in the t coordinate is [t_1,t_2]. Then we draw the horizontal bar of length $t_2 - t_1$ at the linear reference point which is the front-end of u (Fig. 4(a)).

To construct the nonmaterialized trajectory, we proceed as follows. Let $m = f(t)$ be some piecewise linear monotonic function that stabs all the line segments constructed by the above procedure. If the vertices of its polygonal line are $(t_1, m_1), (t_2, m_2), ...(t_n, m_n)$, then this sequence is a nonmaterialized trajectory on the street S.

This NTC procedure is illustrated in example 3.

Example 3. Figure 5 illustrates the nonmaterialized trajectory construction procedure. It is applied to the road-snapped trajectory of Example 1. The trajectory is of eight vertices corresponding to eight nodes in π, which are labeled by the first components of the node in Figure 5. We combine the construction of nonmaterialized trajectory on two streets S_1 and S_2 in one figure. We first construct the time linear reference/street linear reference space (m, t). Next, the horizontal/vertical line segment for each node in path π is computed, shown as the bars in the figure. For each street that route $R = (l_6, l_9, l_8)$ travels, we find a minimal-size (i.e. minimum number of vertices) polyline stabbing. We stab all the bars of street $S_1(l_6)$ using one straight line. Then, from the bars of the intersection of l_6 and l_9, we stab the rest of the bars on street S_2 (l_9 and l_8) with a two piece polyline. Writing down the street name, the m and t value for each dot in the figure, we get the nonmaterialized representation of trajectory T on map M. Note that the size of the nonmaterialized trajectory is four and T has six vertices. In this sense, the figure shows the data reduction aspect of our approach. □

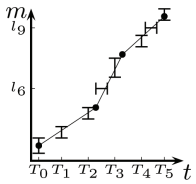


Fig. 5. Constructing the nonmaterialized trajectory

The question now is what ensures that a piece-wise linear function required by the above theorem exists? The answer is given by Theorem 1. Namely, if the path π exists there is a nonmaterialized trajectory, and therefore a stabbing. We are interested in a stabbing that has a minimum number of straight line segments, because this will ensure a minimum number of tuples in the nonmaterialized trajectory for π. This can be done using the results of [9, 12]. It provides a greedy algorithm for stabbing n line segments with a polygonal line of minimum size in linear time.

Theorem 3. *For every trajectory T, map M, and positive real number ε, a nonmaterialized trajectory T'' created with the above algorithm satisfies: (1) The route $R(T'')$ is a path in the map. (2) The distance between the original trajectory T and T'' at most ε. (3) It has minimum size among all nonmaterialized trajectories on $R(T'')$.*

The total number of vertical and horizontal bars is $O(n + m)$, each bar can be constructed in constant time, and the piecewise linear stabbing $m = f(t)$ can be constructed in linear time, using the approach in [9]. Therefore, the time complexity of the NTC algorithm is $O(n + m)$. Thus, the time complexity of finding the nonmaterialized trajectory is dominated by the previous step of the algorithm (finding π), and is $O(nm^2)$.

5 Bounded Error of Spatio-temporal Queries

Our proposed nonmaterialized trajectory T'' is a road-snapped trajectory at distance ε from T. In this section we will analyze the relationship between a trajectory and its road-snapped trajectory with respect to the error in answering spatio-temporal queries. We show that in general, although the distance between a trajectory T and its road-snapped trajectory T' is bounded, the error of spatio-temporal queries may be unbounded. In other words, distance between the answer to a query on T and the same query on T' may be arbitrarily large. Particularly, even if the Euclidean distance between T and T' is bounded, then this undesirable phenomenon, namely unbounded query errors, may occur. We also show that this undesirable behavior does not occur for the road-snapped trajectories produced by the algorithm introduced in this paper. The reason is that the algorithm uses the time_uniform distance between T and T'.

We consider the following basic spatio-temporal *query types*, whose semantics for a trajectory $T = (x_1, y_1, t_1)$, (x_2, y_2, t_2), ..., (x_n, y_n, t_n), are as follows:

- *where_at*(T, t) – returns the location of the trajectory T at time t.
- *intersect*(T, P, t_1, t_2) – is *true* if the location of T is inside the convex polygon P sometime between t_1 and t_2. (This is also called a range query).

We first define the concept of query-error-boundedness for a distance-function between trajectories. The concept is defined for a query type. Then we show that the Euclidean distance is not query-error-bounded for the spatio-temporal query types, but the time_uniform distance is query-error-bounded for them.

Now we explain the notion of query-error-boundedness. So far we restricted the discussion to the time_uniform distance between a trajectory T and its ε-road-snapped trajectory T' (see def. 2). Here we relax this restriction. Let $q(T)$ denote the answer of some spatio-temporal query q with respect to a trajectory T. Similarly, let $q(T')$ denote the answer of the same query q when posed to an ε_D-road-snapped trajectory T' of T. We say that the distance function D is query-error-bounded for q when there exists a bound δ on the distance between the two answers. More precisely, if we let *dist(q(T),q(T'))* denote the distance between the two answers, query-error-boundedness of D means that for every ε there exists a δ such that for every trajectory T, *dist(q(T),q(T'))* $\leq \delta$.

We formalize this notion for each of the query types, as follows. A distance function D is error-bounded with respect to query-type q if for every tolerance ε, there exists a positive number δ, called the answer error bound, such that for every trajectory T and a ε_D-road-snapped trajectory T' of T (the rest of the definition depends on the query-type as follows):

- *where_at* – For every t for which both T and T' are defined, let $(x, y) = where\_at(T, t)$ and let $(x', y') = where\_at(T', t)$. The distance between (x, y) and (x', y') is bounded by δ, namely $\sqrt{(x' - x)^2 + (y' - y)^2} \leq \delta$.
- *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 ε_D-road-snapped trajectory 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 road-snapped trajectories, may decide to adjust the polygon P accordingly.

The following subsumption relationship holds among query types.

Theorem 4. *Any distance function D is error-bounded w.r.t. the where_at query type if and only if it is error-bounded for the intersect query type.*

Interestingly, the Euclidean distance is not error-bounded w.r.t. where_at query type. While the time_uniform distance is error-bounded.

Theorem 5. *The 3D Euclidean distance is not error-bounded w.r.t the where_at query type.*

Theorem 6. *The time_uniform distance is error-bounded w.r.t. the where_at query type. Furthermore, for any tolerance ε, the answer-error-bound of the where_at query-type is ε.*

Together with Theorem 4, the above result implies that the time_uniform distance is also error-bounded w.r.t. the intersect type. It can also be shown that for the distance E_u, for any tolerance ε, the answer-error-bound of the intersect query type is equal to ε.

6 Related Work

Recently, modeling, management, and query processing of network confined movement has received significant attention[8, 14]. However, the required error-correction to make the work applicable has been ignored. Our study provides the necessary preprocessing step. Some papers adopted the similar idea of separating spatial and temporal components of trajectories [8]. However, their objective was to improve the performance of indexing, whereas our objective here is to correct errors, provide a higher level of motion abstraction, and reduce size.

Trajectory snapping is also studied for car navigation under the title *map matching*[11, 16]. Most of those works take a heuristic approach to snapping, and their main purpose is to determine in real time the current block the driver is on. In order to do so they only consider the last GPS point, or the last few GPS points. Therefore, when considering the snapped blocks one may obtain a route that is not connected. However, since the purpose is simply to determine the current location of a user in real-time, this drawback is not important for their purpose. The two-page paper [17] provides a heuristic for map matching. To the best of our knowledge, our road-snapped trajectory construction algorithm is the first complete map matching algorithm, which determinates whether a road-snapped trajectory exists for the given error bound.

Similar to map matching, researchers are also studying the matching problems between different spatial datasets[5], and the problem of robotic mapping[15]. However, the objectives of these papers are different than ours, and their techniques are probably not directly applicable here.

Nonmaterialized trajectory representation of motion is related to data reduction, a very popular topic in the database research. When it comes to generating the answers to the queries, there are two approaches: 1. The data is decompressed before answering a query [6]; and 2. The compressed data is used to answer the query, and the answer contains some error [4, 10]. Our approach is *lossy*, i.e we do not recover the original trajectories after snapping. Recently *wavelets* have become a popular paradigm for data reduction which provides fast and "reasonably approximate" answer to queries [4]. 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 which provides deterministic error-bounds to queries, is that these works either do not ensure a bound on the error of query answers, or ensure an *asymptotic/probabilistic* bounds on the error. A similar observation holds for the works which use *histograms* or *sampling* to compress the data and provide a reasonably accurate answer to the queries (see [1]).

Finally, let us mention some previous work on data reduction by strong line-simplification ([3][13]). These work did not address road-snapping or nonmaterialized trajectories, thus the line simplified trajectories may still be off the road network. However, [3] did consider soundness of queries. The concept of error-boundedness in this paper is a generalization of soundness to the case where the vertices of the approximate trajectory are not necessarily a subset of those of the original trajectory (in contrast to line simplification which imposes such a

restriction). [13] also used the time_uniform distance and studied the error and the compression ratio experimentally.

7 Conclusions

With the proliferation of location based services and mobile devices including sensors, computers, and GPS receivers, the importance of motion information will increase tremendously. In this paper we addressed the problem of producing a higher level of abstraction for motion data, based on constraints provided by road networks. We introduced an algorithm for "adjusting" a given trajectory T to fit the road network; the adjustment is called a road-snapped trajectory, T'', and it has several properties. First, it is within a distance ε (the location-sensor error) from T. Second, it is on the road network. Third, it is nonmaterialized, i.e. it provides the temporal information separately from the spatial information common to all the trajectories. Fourth, it is minimized in a local sense, i.e. for a given materialized snapped trajectory. Fifth, it is error bounded with respect to the spatio-temporal queries: where is a moving object at a given time, range query, nearest neighbor, and join. In other words, the answers to any such query posed on T and T' are close. We have shown that this property is not trivial even though T and T' are ε-close; i.e. the property holds for the time_uniform distance metric, but not for the Euclidean metric. The time-complexity of the algorithm is $O(nm^2)$, for a trajectory with n vertices and a map with m straight line segments at distance ε from T.

References

1. Special issue on data reduction techniques. *IEEE Data Engineering*, 20(4), 1998.
2. H. Alt and L. J. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation A survey. Technical Report B 96-11, Institut für Informatik, Freie Universität Berlin, 1996.
3. H. Cao, O. Wolfson, and G. Trajcevski. Spatiotemporal data reduction with deterministic error bounds. In *DIALM-POMC'03*, pages 33–42, 2003.
4. K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *VLDB 2000*, September 2000.
5. C. Chen, S. Thakkar, C. Knoblock, and C. Shahabi. Automatically annotating and integrating spatial datasets. In *SSTD'03*, 2003.
6. Z. Chen, J. Gehrke, and F. Korn. Query optimization in compressed database systems. In *ACM SIGMOD 2001*, pages 271–282. ACM Press, 2001.
7. L. Florizzi, R. H. Guting, E. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. Technical Report 260-10, Fern-Universität Hagen, 1999.
8. E. Frentzos. Indexing objects moving on fixed networks. In *Proc. 8th Int'l Symposium on Spatial and Temporal Databases, SSTD'03*, 2003.
9. S. K. Ghosh. Computing the visibility polygon from a convex set and related problem. *Journal of Algorithms*, 12:75–95, 1991.
10. P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *VLDB*, 1997.

11. J. S. Greenfeld. Matching gps observations to locations on a digital map. In *The 81th Annual Meeting of the Transportation Research Board*, Washington D.C, 2002.
12. L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. xS. Snoeyink. Approximating polygons and subdivisions with minimum link paths. In *ISAAC' 91*, 1991.
13. N. Meratnia and R. A. de By. Spatiotemporal compression techniques for moving point objects. In *EDBT*, pages 765–782, 2004.
14. D. Pfoser and C. S. Jensen. Indexing of network constrained moving objects. In *ACM GIS*, pages 25–32. ACM Press, 2003.
15. S. Thrun. Robotic mapping: a survey. In *Exploring artificial intelligence in the new millennium*, pages 1–35. Morgan Kaufmann Publishers Inc., 2003.
16. C. E. White, D. Bernstein, and A. L. Kornhauser. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C*, 8:91–108, 2000.
17. H. Yin and O. Wolfson. A weight-based map matching method in moving objects databases. In *SSTDM*, 2004.