

The Geometry of Uncertainty in Moving Objects Databases

Goce Trajcevski¹, Ouri Wolfson^{1,2,*}, Fengli Zhang¹, and Sam Chamberlain³

¹ University of Illinois at Chicago, Dept. of CS
{gtrajcev,wolfson,fzhang}@cs.uic.edu

² Mobitrac, Inc., Chicago

³ Army Research Laboratory, Aberdeen Proving Ground, MD wildman@arl.mil

Abstract. This work addresses the problem of querying moving objects databases. which capture the inherent uncertainty associated with the location of moving point objects. We address the issue of modeling, constructing, and querying a trajectories database. We propose to model a trajectory as a 3D cylindrical body. The model incorporates uncertainty in a manner that enables efficient querying. Thus our model strikes a balance between modeling power, and computational efficiency. To demonstrate efficiency, we report on experimental results that relate the length of a trajectory to its size in bytes. The experiments were conducted using a real map of the Chicago Metropolitan area.

We introduce a set of novel but natural spatio-temporal operators which capture uncertainty, and are used to express spatio-temporal range queries. We also devise and analyze algorithms to process the operators. The operators have been implemented as a part of our DOMINO project.

1 Introduction and Motivation

Miniaturization of computing devices, and advances in wireless communication and sensor technology are some of the forces that are propagating computing from the stationary desktop to the mobile outdoors. Important classes of new applications that will be enabled by this revolutionary development include location based services, tourist services, mobile electronic commerce, and digital battlefield. Many existing applications will also benefit from this development: transportation and air traffic control, weather forecasting, emergency response, mobile resource management, and mobile workforce. Location management, i.e. the management of transient location information, is an enabling technology for all these applications. It is also a fundamental component of other technologies such as fly-through visualization, context awareness, augmented reality, cellular communication, and dynamic resource discovery.

Database researchers have addressed some aspects of the problem of modeling and querying the location of moving objects. Largest efforts were made in

* Research supported by ARL Grant DAAL01-96-2-0003, NSF Grants ITR-0086144, CCR-9816633, CCR-9803974, IRI-9712967, EIA-0000516, INT-9812325

the area of access methods. Aside from a purely spatial ([8] surveys 50+ structures) and temporal databases [27], there are several recent results which tackle various problems of indexing *spatio-temporal* objects and dynamic attributes [1,12,13,17,22,28,30,31]. Representing and querying the location of moving objects as a function of time is introduced in [24], and the works in [36,37] address policies for updating and modeling imprecision and communication costs. Modeling and querying location uncertainties due to sampling and GPS imprecision is presented in [16]. Algebraic specifications of a system of abstract data types, their constructors and a set of operations are given in [4,6,9].

In this paper we deal in a systematic way with the issue of *uncertainty* of the *trajectory* of a moving object. Uncertainty is an inherent aspect in databases which store information about the location of moving objects. Due to continuous motion and network delays, the database location of a moving object will not always precisely represent its real location. Unless uncertainty is captured in the model and query language, the burden of factoring uncertainty into answers to queries is left to the user.

Traditionally, the trajectory of a moving object was modeled as a polyline in three dimensional space (two dimensions for geography, and one for time). In this paper, in order to capture uncertainty we model the trajectory as a cylindrical volume in 3D. Traditionally, spatio-temporal range queries ask for the objects that are inside a particular region, during a particular time interval. However, for the moving objects one may query the objects that are inside the region *sometime* during the time interval, or for the ones that are *always* inside during the time interval. Similarly, one may query the objects that are *possibly* inside the region or for the ones that are *definitely* there. For example, one may ask queries such as:

Q1: “Retrieve the current location of the delivery trucks that will *possibly* be inside a region *R*, *sometime* between 3:00PM and 3:15PM”.

Q2: “Retrieve the number of tanks which will *definitely* be inside the region *R* *sometime* between 1:30PM and 1:45PM.”.

Q3: “Retrieve the police cars which will *possibly* be inside the region *R*, *always* between 2:30AM and 2:40AM”.

We provide the syntax of the operators for spatio-temporal range queries, and their processing algorithms. It turns out that these algorithms have a strong geometric flavor. We also wanted to determine whether for realistic applications the trajectories database can be stored in main memory. We generated over 1000 trajectories using a map of Chicagoland and analyzed their average size – approximately 7.25 line segments per mile. Thus, for fleets of thousands of vehicles the trajectories database can indeed be stored in main memory.

The model and the operators that we introduce in this paper have been implemented in our DOMINO system. The operators are built as User Defined Functions (UDF) in Informix IDS2000. A demo version of the DOMINO system is available at <http://131.193.39.205/mapcafe/mypage.html>. The operators are built as User Defined Functions (UDF) in Informix IDS2000. Our main contributions can be summarized as follows:

1. – We introduce a trajectory model with uncertainty, and its construction based on electronic maps;
2. – We experimentally evaluate the average length of a trajectory and determine that it is about 7.25 line segments per mile;
3. – We introduce a set of operators for querying trajectories with uncertainty. We provide both linguistic constructs and processing algorithms, and show that the complexity of the algorithms is either linear or quadratic.

The rest of the article is structured as follows. In section 2 we define the model of a trajectory and show how it can be constructed based on electronic maps. Section 3 discusses the experiments to determine the average size of a trajectory. Section 4. defines the uncertainty concepts for a trajectory. In Section 5. we present the *syntax* and the *semantics* of the new operators for querying trajectories with uncertainty. Section 6. provides the processing algorithms and their analysis. Section 7 concludes the paper, positions it in the context of related work, and outlines the future work.

2 Representing and Constructing the Trajectories

In this section we define our model of a *trajectory*, and we describe how to construct it from the data available in electronic maps. In order to capture the spatio-temporal nature of a moving object we use the following:

Definition 1. A trajectory of a moving object is a polyline in three-dimensional space (two-dimensional geography, plus time), represented as a sequence of points (x_1, y_1, t_1) , (x_2, y_2, t_2) , ..., (x_n, y_n, t_n) ($t_1 < t_2 < \dots < t_n$). For a given a trajectory Tr , its projection on the X-Y plane is called the route of Tr .

A trajectory defines the location of a moving object as an implicit function of time. The object is at (x_i, y_i) at time t_i , and during each segment $[t_i, t_{i+1}]$, the object moves along a straight line from (x_i, y_i) to (x_{i+1}, y_{i+1}) , and at a constant speed.

Definition 2. Given a trajectory Tr , the expected location of the object at a point in time t between t_i and t_{i+1} ($1 \leq i < n$) is obtained by a linear interpolation between (x_i, y_i) and (x_{i+1}, y_{i+1}) .

Note that a trajectory can represent both the past and future motion of objects. As far as future time is concerned, one can think of the trajectory as a set of points describing the *motion plan* of the object. Namely, we have a set of points that the object is going to visit, and we assume that in between the points the object is moving along the shortest path. Given an electronic map, along with the beginning time of the object's motion, we construct a trajectory as a superset of the set of the given – “to-be-visited” – points. In order to explain how we do so, we need to define an electronic map (or a map, for brevity).

Definition 3. A map is a graph, represented as a relation where each tuple corresponds to a block with the following attributes:

- Polyline: Each block is a polygonal line segment. Polyline gives the sequence of the endpoints: $(x_1, y_1), (x_2, y_2), \dots (x_n, y_n)$.
 - Length: Length of the block.
 - Fid: The block id number.
 - Drive_Time: Typical drive time from one end of the block to the other, in minutes.
- Plus, among others, a set of geo-coding attributes which enable translating between an (x, y) coordinate and an address, such as “1030 North State St.”:
(e.g. – L.f.add: Left side from street number.)

Such maps are provided by, among the others, Geographic Data Technology¹ Co. An intersection of two streets is the endpoint of the four block – polylines. Thus, each map is an undirected graph, with the tuples representing edges of the graph.

The route of a moving object O is specified by giving the starting address or (x, y) coordinate, namely the *start_point*; the starting time; and the destination address or (x, y) coordinate, namely the *(end_point)*. An external routine, available in most Geographic Information Systems, which we assume is given a priori, computes the shortest cost (distance or travel – time) path in the map graph. This path, denoted $P(O)$, is a sequence of blocks (edges), i.e. tuples of the map. Since $P(O)$ is a path in the map graph, the endpoint of one block polyline is the beginning point of the next block polyline. Thus, the route represented by $P(O)$ is a polyline denoted by $L(O)$. Given that the trip has a starting time, we compute the trajectory by computing for each straight line segment on $L(O)$, the time at which the object O will arrive to the point at the end of the segment. For this purpose, the only relevant attributes of the tuples in $P(O)$ are *Polyline* and *Drive_Time*.

Finally, let us observe that a trajectory can be constructed based on past motion. Specifically, consider a set of 3D points $(x_1, y_1, t_1), (x_2, y_2, t_2), \dots, (x_n, y_n, t_n)$ which were transmitted by a moving object periodically, during its past motion. One can construct a trajectory by first “snapping” the points on the road network, then simply connecting the snapped points with the shortest path on the map.

3 Experimental Evaluation of Trajectory Sizes

In this section we describe our experiments designed to evaluate the number of line segments per trajectory.

As a part of our DOMINO project we have constructed 1141 trajectories based on the electronic map of 18 counties around Chicagoland. The map size is 154.8MB, and has 497,735 records representing this many city-blocks.

The trajectories were constructed by randomly choosing a pair of end points, and connecting them by the shortest path in the map (shortest in terms of the

¹ (www.geographic.com)

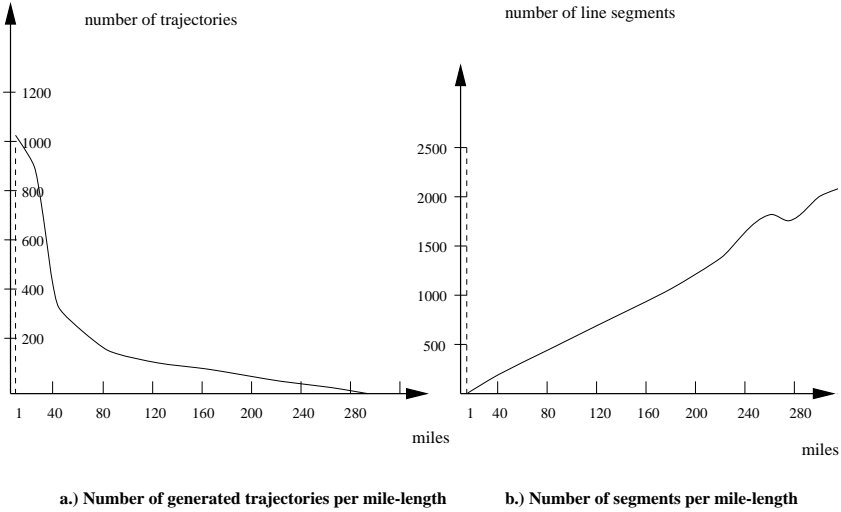


Fig. 1. Number of segments in real - map trajectories

Drive-Time) Our results are depicted on Figure 1. The length of the routes was between 1 and 289 miles, as shown on the left graph.

The right graph represents the number of segments per trajectory, as a function of the length of the route. We observed a linear dependency between the “storage requirements” (number of segments) and the length of a route.

The average number of segments per mile turned out to be 7.2561. Assuming that a trajectory point (x, y, t) uses 12 bytes and that each vehicle from a given fleet (e.g. a metropolitan delivery company), drives a route of approximately 100 miles, we need $\approx 10\text{K}$ bytes for a trajectory. Then the storage requirements for all the trajectories of a fleet of 1000 vehicles is ≈ 10 MB. This means that the trajectories of the entire fleet can be kept in the main memory.

4 Uncertainty Concepts for Trajectories

An uncertain trajectory is obtained by associating an uncertainty threshold r with each line segment of the trajectory. For a given motion plan, the line segment together with the uncertainty threshold constitute an “agreement” between the moving object and the server. The agreement specifies the following: the moving object will update the server if and only if it deviates from its expected location (according to the trajectory) by r or more. How does the moving object compute the deviation at any point in time? Its on-board computer receives a GPS update every two seconds, so it knows its actual location. Also, it has the trajectory, so by interpolation it can compute its expected location at any point in time. The deviation is simply the distance between the actual and the expected location.

Definition 4. Let r denote a positive real number and Tr denote a trajectory. An uncertain trajectory is the pair (Tr, r) . r is called the uncertainty threshold.

Definition 5. Let $Tr \equiv (x_1, y_1, t_1), (x_2, y_2, t_2), \dots, (x_n, y_n, t_n)$ denote a trajectory and let r be the uncertainty threshold. For each point (x, y, t) along T , its r-uncertainty area (or the uncertainty area for short) is a horizontal circle with radius r centered at (x, y, t) , where (x, y) is the expected location at time $t \in [t_1, t_n]$.

Note that our model of uncertainty is a little simpler than the one proposed in [16]. There, the uncertainty associated with the location of an object traveling between two endpoints of a line segment is an ellipse with foci at the endpoints.

Definition 6. Let $Tr \equiv (x_1, y_1, t_1), (x_2, y_2, t_2), \dots, (x_n, y_n, t_n)$, be a trajectory, and let r be an uncertainty threshold. A Possible Motion Curve $PMC^T r$ is any continuous function $f_{PMC^T r} : Time \rightarrow R^2$ defined on the interval $[t_1, t_n]$ such that for any $t \in [t_1, t_n]$, $f_{PMC^T r}(t)$ is inside the uncertainty area of the expected location at time t .

Intuitively, a possible motion curve describes a route with its associated times, which a moving object may take, without generating an update. An object does not update the database as long as it is on some possible motion curve of its uncertain trajectory (see Figure 2). We will refer to a 2D projection of a possible motion curve as a possible route.

Definition 7. Given an uncertain trajectory (Tr, r) and two end-points $(x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1}) \in Tr$, the trajectory volume of Tr between t_i and t_{i+1} is the set of all the points (x, y, t) such that (x, y, t) belongs to a possible motion curve of Tr and $t_i \leq t \leq t_{i+1}$. The 2D projection of the trajectory volume is called an uncertainty zone.

Definition 8. Given a trajectory $Tr \equiv (x_1, y_1, t_1), (x_2, y_2, t_2), \dots, (x_n, y_n, t_n)$ and an uncertainty threshold r , the trajectory volume of (Tr, r) is the set of all the trajectory volumes between t_i and t_{i+1} ($i = 1, \dots, (n - 1)$).

Definitions 6, 7 and 8, are illustrated in Figure 2. Viewed in 3D, a trajectory volume between t_1 and t_n is sequence of volumes, each bounded by a cylindrical body. The axis of each is the vector which specifies the 3D trajectory segment, and the bases are the circles with radius r in the planes $t = t_{begin}$ and $t = t_{end}$. Observe that the cylindrical body is different from a tilted cylinder. The intersection with of a tilted cylinder with a horizontal plane (parallel to the (X,Y) plane) yields an ellipse, whereas the intersection of our cylindrical body with such a plane yields a circle. Thus, the trajectory volume between two points resembles a set of circles of the uncertainty areas, stacked on top of each other. Let v_{xi} and v_{yi} denote the x and y components of the velocity of a moving object along the i -th segment of the route (i.e. between (x_i, y_i) and (x_{i+1}, y_{i+1})). It can be shown [32] that the trajectory volume between t_i and t_{i+1} is the set of all the points which satisfy: $t_i \leq t \leq t_{i+1}$ and $(x - (x_i + v_{xi} \cdot t))^2 + (y - (y_i + v_{yi} \cdot t))^2 \leq r^2$

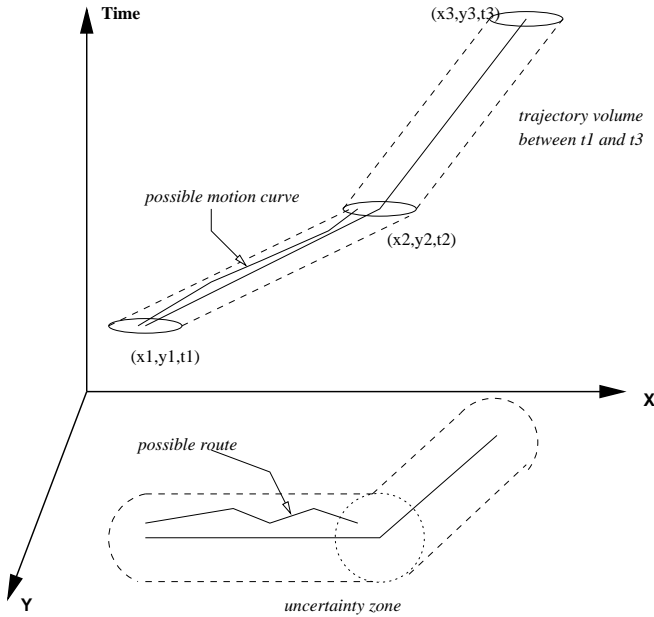


Fig. 2. Possible motion curve and trajectory volume

5 Querying Moving Objects with Uncertainty

In this section we introduce two categories of operators for querying moving objects with uncertainty. The first category, discussed in section 5.1, deals with *point* queries and it consists of two operators which pertain to a single trajectory. The second category, discussed in section 5.2, is a set of six (boolean) predicates which give a qualitative description of a relative position of a moving object with respect to a region, within a given time interval. Thus, each one of these operators corresponds to a spatio-temporal *range* query.

5.1 Point Queries

The two operators for point queries are defined as follows:

- *Where_At(trajectory Tr , time t)* – returns the expected location on the route or Tr at time t .
- *When_At(trajectory Tr , location l)* – returns the times at which the object on Tr is at expected location l . The answer may be a set of times, in case the moving object passes through a certain point more than once. If the location l is not on the route of the trajectory Tr , we find the set of all the points C on this route which are closest to l . The function then returns the set of times at which the object is expected to be at each point in C .

The algorithms which implement the point query operators are straightforward. The *Where_at* operator is implemented in $O(\log n)$ by a simple binary

search, where n is the number of line segments of the trajectory. The *When_at* operator is implemented in linear time by examining each line segment of a trajectory. As we demonstrated in Section 3, any reasonable trajectory has no more than several thousand line segments. It can be stored in the main memory and the processing time of each one of the above operators is acceptable.

5.2 Operators for Spatio-temporal Range Queries

The second category of operators is a set of conditions (i.e. boolean predicates). Each condition is satisfied if a moving object is inside a given region R , during a given time interval $[t_1, t_2]$. Clearly, this corresponds to a spatio-temporal range query. But then, why more than one operator? The answer is threefold: 1. – The location of the object changes continuously, hence one may ask if the condition is satisfied *sometime* or *always* within $[t_1, t_2]$; 2. – The object may satisfy the condition *everywhere* or *somewhere* within the region R ; 3. – Due to the uncertainty, the object may *possibly* satisfy the condition or it may *definitely* do so.

Thus, we have three domains of quantification, with two quantifiers in each. Combining all of them would yield $2^3 \cdot 3! = 48$ operators. However, some of them are meaningless in our case. In particular, it makes no sense to ask if a point object is *everywhere* within a 2D region R (we do not consider “everywhere in all the route segments within a region” in this paper). Hence we have only $2^2 \cdot 2! = 8$ operators.

A region is a polygon². In what follows, we let $PMCT$ denote a possible motion curve of a given uncertain trajectory $T = (Tr, r)$:

- *Possibly_Sometime_Inside*(T, R, t_1, t_2) – is *true* iff there exist a possible motion curve $PMCT$ and there exists a time $t \in [t_1, t_2]$ such that $PMCT$ at the time t , is inside the region R . Intuitively, the truth of the predicate means that the moving object may take a possible route, within its uncertainty zone, such that the particular route will intersect the query polygon R between t_1 and t_2 .
- *Sometime_Possibly_Inside*(T, R, t_1, t_2) – is *true* iff there exist a time $t \in [t_1, t_2]$ and a possible motion curve $PMCT$ of the trajectory T , which at the time t is inside the region R . Observe that this operator is semantically equivalent to *Possibly_Sometime_Inside*. Similarly, it will be clear that *Definitely_Always_Inside* is equivalent to *Always_Definitely_Inside*. Therefore, in effect, we have a total of 6 operators for spatio-temporal range queries.
- *Possibly_Always_Inside*(T, R, t_1, t_2) – is *true* iff there exists a possible motion curve $PMCT$ of the trajectory T which is inside the region R for every t in $[t_1, t_2]$. In other words, the motion of the object is such that it may take (at least one) specific 2D possible route, which is entirely contained within the polygon R , during the whole query time interval.
- *Always_Possibly_Inside*(T, R, t_1, t_2) – is *true* iff for every time point $t \in [t_1, t_2]$, there exists a $PMCT$ which will intersect the region R at t .

² We will consider *simple* polygons (c.f. [18,19]) and without any holes.

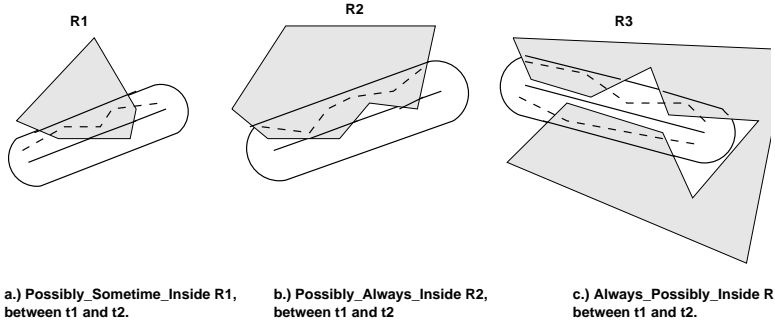


Fig. 3. Possible positions of a moving point with respect to region R_i

Figure 3 illustrates (a 2D projection of) a plausible scenario for each of the three predicates above (dashed lines indicate the possible motion curve(s) due to which the predicates are satisfied; solid lines indicate the routes and the boundaries of the uncertainty zone).

The next theorem indicates that one of the last two predicates is stronger than the other:

Theorem 9. Let $Tr = (T, r)$ denote an uncertain trajectory; R denote a polygon; and t_1 and t_2 denote two time points. If *Possibly_Always_Inside* (T, R, t_1, t_2) is true, then *Always_Possibly_Inside* (T, R, t_1, t_2) is also true³.

Note that the converse of Theorem 9 is not true. As illustrated on Figure 3, the predicate *Always_Possibly_Inside* maybe satisfied due to two or more possible motion curves, none of which satisfies *Possibly_Always_Inside* by itself. However, as the next theorem indicates, this situation cannot occur for a convex polygon:

Theorem 10. Let $Tr = (T, r)$ denote an uncertain trajectory; R denote a convex polygon; and t_1 and t_2 denote two time points. *Possibly_Always_Inside* (T, R, t_1, t_2) is true, iff *Always_Possibly_Inside* (T, R, t_1, t_2) is true.

The other three predicates are defined as follows:

- *Always_Definitely_Inside* (T, R, t_1, t_2) – is true iff at every time $t \in [t_1, t_2]$, every possible motion curve $PMCT$ of the trajectory T , is in the region R . In other words, no matter which possible motion curve the object takes, it is guaranteed to be within the query polygon R throughout the entire interval $[t_1, t_2]$. Note that this predicate is semantically equivalent to *Definitely_Always_Inside*.
- *Definitely_Sometime_Inside* (T, R, t_1, t_2) – is true iff for every possible motion curve $PMCT$ of the trajectory T , there exists some time $t \in [t_1, t_2]$ in which the particular motion curve is inside the region R . Intuitively, no matter which possible motion curve within the uncertainty zone is taken by the moving object, it will intersect the polygon at some time between t_1 and t_2 . However, the time of the intersection may be different for different possible motion curves.

³ Due to lack of space, we omit the proofs of the Theorems and the Claims in this paper (see [32]).

- *Sometime_Definitely_Inside*(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 $PMCT^T$ of the trajectory T is inside the region R . Satisfaction of this predicate means that no matter which possible motion curve is taken by the moving object, at the specific time t the object will be inside query polygon.

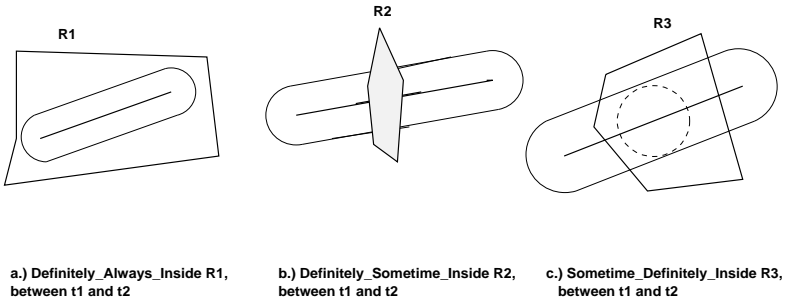


Fig. 4. *Definite* positions of a moving point with respect to region R_i

The intuition behind the last three predicates is depicted on Figure 4.

Again we observe that *Sometime_Definitely_Inside* is stronger than *Definitely_Sometime_Inside*:

Theorem 11. Let $Tr = (T, r)$ denote an uncertain trajectory; R denote a polygon; and t_1 and t_2 denote two time points. If *Sometime_Definitely_Inside*(T, R, t_1, t_2) is true, then *Definitely_Sometime_Inside*(T, R, t_1, t_2) is also true.

However, the above two predicates are not equivalent even if the polygon R is convex. An example demonstrating this is given in of Figure 4(b). The polygon R_2 satisfies *Definitely_Sometime_Inside*, but since it does not contain the uncertainty area for any time point, it does not satisfy *Sometime_Definitely_Inside*.

Note that the proofs of Theorems 9 and 11 are straightforward consequence of $\exists x \forall y P(x, y) \rightarrow \forall y \exists x P(x, y)$ (where P denotes “the property”). However, Theorem 10 is specific to the problem domain.

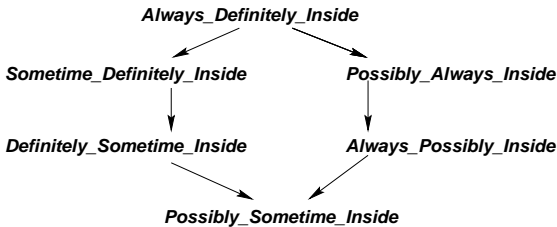


Fig. 5. Relationships among the spatiotemporal predicates

The relationships among our predicates is depicted on Figure 5, where the arrow denotes an implication. More complex query conditions can be expressed by composition of the operators. Consider, for example, “Retrieve all the objects which are possibly within a region R , always between the times the object A arrives at locations L_1 and L_2 ”. This query can be expressed as:

Possibly_Always_Inside($T, R, \text{When_At}(T_A, L_1), \text{When_At}(T_A, L_2)$).

We have implemented the six spatio-temporal range query operators in our DOMINO project. The implementation algorithms are described in the next section, but we conclude this section with a discussion of the user interface.

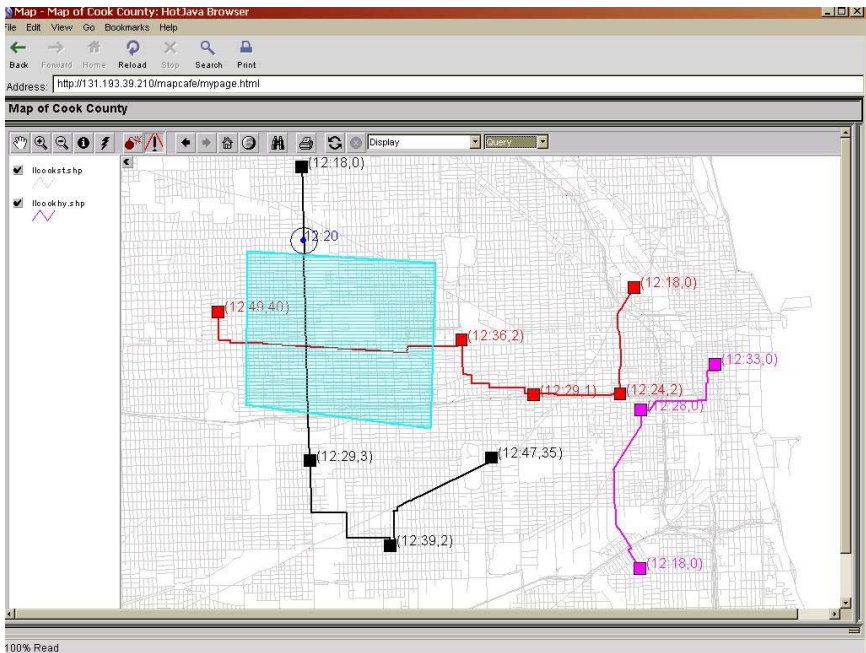


Fig. 6. Visualization of *Possibly_Sometime_Inside*

Figure 6 illustrates the GUI part of the DOMINO project which implements our operators. It represents a visual tool which, in this particular example, shows the answer to the query: “Retrieve the trajectories which *possibly* intersect the region *sometime* between 12:15 and 12:30”. The figure shows three trajectories in the Cook County, Illinois, and the query region (polygon) represented by the shaded area. The region was drawn by the user on the screen when entering the query. Each trajectory shows the route with planned stops along it (indicated by dark squares). It also shows the expected time of arrival and the duration of the job (i.e. the stay) at each stop. Observe that only one of the trajectories satisfies the predicate *Possibly_Sometime_Inside* with respect to the polygon. It is the one with the circle labeled 12:20, the earliest time at which the object could enter

the query polygon. The other two trajectories fail to satisfy the predicate, each for a separate reason. One of them will not intersect the polygon ever (i.e. the polygon is not on the route). Although the other trajectory’s route intersects the polygon, the intersection will occur at a time which is not within the query time - interval [12 : 15, 12 : 30].

6 Processing the Range Operators

In this section, for each of the operators we identify the topological properties which are necessary and sufficient conditions for their truth, and we present the algorithms which implement them. The complexities of the algorithms we provide assume relatively straightforward computational geometry techniques. Some of them may be improved using more sophisticated techniques (c.f. [18,19]), which we omit for space consideration. We only consider query regions that are represented by convex polygons⁴.

Throughout this section, let t_1 and t_2 be two time-points. Taking time as a third dimension, the region R along with the query time-interval $[t_1, t_2]$ can be represented as a prism P_R in 3D space: $P_R = \{(x, y, t) \mid (x, y) \in R \wedge t_1 \leq t \leq t_2\}$. P_R is the *query-prism*.

For the purpose of query processing, we assume an available 3D indexing scheme in the underlying DBMS, similar to the ones proposed in [17,28,34]. The insertion of a trajectory is done by enclosing, for each trajectory, each trajectory volume between t_i and t_{i+1} in a Minimum Bounding Box (MBB). During the *filtering* stage we retrieve the trajectories which have a MBB that intersect with P_R . Throughout the rest of this work we focus on the *refinement* stage of the processing. Let VTr denote the trajectory volume of a given uncertain trajectory $T = (Tr, r)$ between t_1 and t_2 . Also, let $VT' = VTr \cap P_R$.

Theorem 12. *The predicate Possibly_Sometime_Inside is true iff $VT' \neq \emptyset$*

To present the processing algorithm, we need the following concept used in Motion Planning (c.f. [19,23,?]). The operation of *Minkowski sum* – denoted as \oplus is described as follows: Let P denote a polygon and d_r denote a disk with radius r . $P \oplus d_r$ is the set of all the points in plane which are elements of $\{P \cup \text{interior of } P \cup \text{the points which are in the “sweep” of } d_r \text{ when its center moves along the edges of } P\}$. Visually, the outer boundary of $P \oplus d_r$, for a convex polygon P , will consist of line segments and circle segments by the vertices of P , as illustrated on Figure 7. If P has n edges, then the complexity of constructing the Minkowski sum $P \oplus d_r$ is $O(n)$ (c.f. [19]).

In what follows, let $Tr_{X,Y}$ denote the projection of the trajectory Tr between t_1 and t_2 , on the $X - Y$ plane (i.e. its route).

⁴ Due to lack of space, we do not present the formal treatment of the concave query polygons. Detailed description is presented in [32].

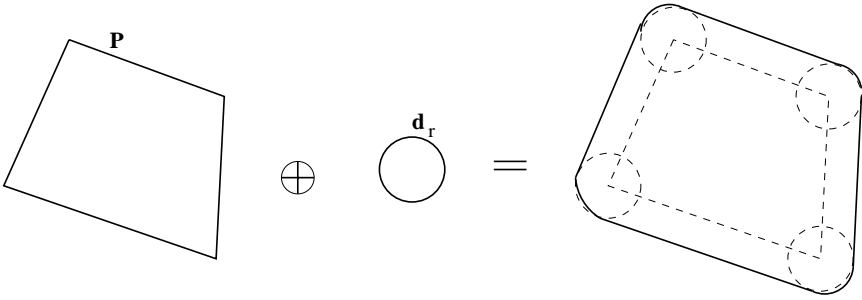


Fig. 7. Minkowski sum of a polygon with a disk

Algorithm 1. (Possibly_Sometime_Inside(R, T, t_1, t_2))

1. Construct the Minkowski sum of R and the disk d_r with radius r , where r is the uncertainty of T .
Denote it $R \oplus d_r$;
2. If $Tr_{X,Y} \cap (R \oplus d_r) = \emptyset$
3. return **false**;
4. else
5. return **true**;

In other words, VT' is nonempty if and only if $Tr_{X,Y}$ intersects the expanded polygon. The complexity of the Algorithm 1 is $O(kn)$ where k is the number of segments of the trajectory between t_1 and t_2 , and n is the number of edges of R .

The next Theorem gives the necessary and sufficient condition for satisfaction of the *Possibly_Always_Inside* predicate:

Theorem 13. *Possibly_Always_Inside(T, R, t_1, t_2) is true if and only if VT' contains a possible motion curve between t_1 and t_2 .*

The implementation is given by the following:

Algorithm 2. (Possibly_Always_Inside(R, T, t_1, t_2))

1. Construct the Minkowski sum of R and the disk d_r with radius r , where r is the uncertainty of T .
Denote it $R \oplus d_r$;
2. If $Tr_{X,Y}$ lies completely inside $R \oplus d_r$
3. return **true**;
4. else
5. return **false**;

The complexity of Algorithm 2 is, again, $O(kn)$.

Recall that we are dealing with convex polygonal regions. As a consequence of the Theorem 10, we can also use the last algorithm to process the predicate *Always_Possibly_Inside*.

Now we proceed with the algorithms that implement the predicates which have the *Definitely* quantifier in their spatial domain.

Theorem 14. *The predicate $Definitely_Always_Inside(Tr, R, t_1, t_2)$ is true if and only if $VTr \cap P_R = VTr$*

As for the implementation of the predicate, we have the following:

Algorithm 3. Definitely_Always_Inside(Tr, R, t_1, t_2)

1. For each straight line segment of Tr
2. If the uncertainty zone of the segment is not entirely contained in R ;
3. return **false** and exit;
4. return **true**.

Step 2 above can be processed by checking if the route segment has a distance from some edge of R which is less than r , which implies a complexity of $O(kn)$ again.

Theorem 15. *Sometime_Definitely_Inside(T, R, t_1, t_2) is true if and only if $VTr \cap P_R$ contains an entire horizontal disk (i.e. a circle along with its interior)*

Let us point out that Theorem 15 holds for concave polygon as well [32].

The implementation of the predicate *Sometime_Definitely_Inside* is specified by the following:

Algorithm 4. Sometime_Definitely_Inside(Tr, R, t_1, t_2)

1. For each segment of Tr such that $Tr_{X,Y} \cap R \neq \emptyset$
2. If R contains a circle with radius r centered at some point on $Tr_{X,Y}$;
3. return **true** and exit
4. return **false**

The complexity of Algorithm 4 is again $O(kn)$.

Now we discuss the last predicate. The property of connectivity is commonly viewed as an existence of some *path* between two points in a given set. Clearly, in our setting we are dealing with subsets of \mathcal{R}^3 . Given any two points a and b in \mathcal{R}^3 , a *path* from a to b is any continuous function⁵ $f : [0, 1] \rightarrow \mathcal{R}^3$ such that $f(a) = 0$ and $f(1) = b$. Given two time – points t_1 and t_2 , we say that a set $\mathcal{S} \subseteq \mathcal{R}^3$ is *connected between* t_1 and t_2 if there exist two points (x_1, y_1, t_1) and $(x_2, y_2, t_2) \in \mathcal{S}$ which are connected by a path in \mathcal{S} . Thus, we have the following Theorem for the predicate *Definitely_Sometime_Inside* (a consequence of Claim 5 below):

Theorem 16. *Definitely_Sometime_Inside(T, R, t_1, t_2) is true if and only if $VT'' = VTr \setminus P_R$ is not connected between t_1 and t_2 .*

Claim. If $VT'' = VTr \setminus P_R$ is connected between t_1 and t_2 , then there exists a possible motion curve PMC^T between t_1 and t_2 which is entirely in VT'' .

⁵ There are propositions (c.f. [26]) about the equivalence of the connectedness of a topological space with the path – connectedness.

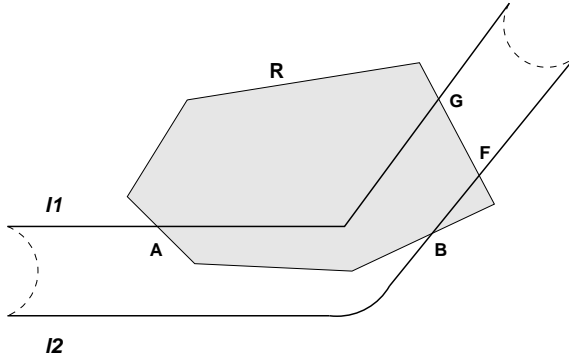


Fig. 8. Processing of *Definitely-Sometime-Inside* predicate

Now we present the algorithm that processes the *Definitely-Sometime-Inside* predicate. Let PT_r be the uncertainty zone of the trajectory (equivalently, the 2D projection of VT_r , the trajectory volume). Let PT'_r be PT_r with the uncertainty areas at t_1 and t_2 eliminated. Let L be the boundary of PT'_r . L will consist of (at most) $2k$ line segments and $k + 1$ circular segments (at most one around the endpoints of each segment). Let $L' = L \setminus D$, where D denotes the two half-circles which bound the uncertainty areas at t_1 and t_2 . Clearly, L' consists of two disjoint “lines” l_1 and l_2 which are left from the initial boundaries of the uncertainty zone. Figure 8 illustrates the concepts that we introduced. Note that the boundary l_2 has a circular segment at the end of the first route-segment. Dashed semi-circles correspond to the boundaries of the uncertainty areas at t_1 and t_2 , which are removed when evaluating the predicate. For the query region R , we have the path between A on l_1 and B on l_2 (also the path between F and G) which makes the predicate true.

Algorithm 5. *Definitely-Sometime-Inside*(T, R, t_1, t_2)

1. If there exists a path P between a point on l_1 and one on l_2 which consists entirely of edges of R (or parts thereof) AND P is entirely in PT'_r ,
2. return **true** and exit
3. return **false**

It is not hard to see that the complexity of Algorithm 5 is $O(kn^2)$.

7 Conclusion, Related Work and Future Directions

We have proposed a model for representing moving objects under realistic assumptions of location uncertainty. We also gave a set of operators which can be used to pose queries in that context. The model and the operators combine spatial, temporal, and uncertainty constructs, and can be fully implemented on top of “off the shelf” existing ORDBMS’s.

Linguistic issues in moving objects databases have been addressed before. Modeling and querying issues have been addressed from several perspectives.

Sistla et al. [24] introduce the MOST model for representing moving objects (similar to [22]) as a function of (location, velocity_vector). The underlying query language is nonstandard, and is based on the Future Temporal Logic (FTL). Similar issues are addressed in [34]. A trajectory model similar to ours is given in [35] where the authors extend range queries with new operators for special cases of spatio-temporal range queries. The series of works [4,6,9] addresses the issue of modeling and querying moving objects by presenting a rich algebra of operators and a very comprehensive framework of abstract data types. However, in all the above works there is no treatment of the uncertainty of the moving object's location.

As for uncertainty issues, Wolfson et al. [36,37] introduce a cost based approach to determine the size of the uncertainty area (r in this paper). However, linguistic and querying aspects are not addressed in these papers. A formal quantitative approach to the aspect of uncertainty in modeling moving objects is presented in [16]. The authors limit the uncertainty to the past of the moving objects and the error may become very large as time approaches *now*. It is a less “collaborative” approach than ours – there is no clear notion of the motion *plan* given by the trajectory. Uncertainty of moving objects is also treated in [25] in the framework of modal temporal logic. The difference from the present work is that here we treat the uncertainty in traditional range queries.

A large body of work in moving objects databases has been concentrated on indexing in primal [17,22,20,21,28] or dual space [1,12,13]. [30,31] present specifications of *what* an indexing of moving objects needs to consider, and generation of spatial datasets for benchmarking data. These works will be useful in studying the most appropriate access method for processing the operators introduced in this paper.

On the commercial side, there is a plethora of related GIS products [33,5,7]; maps with real – time traffic information [11] and GPS devices and management software. IBM's DB2 Spatial Extender [3], Oracle's Spatial Cartridge [15] and Informix Spatial DataBlade [29] provide several 2D – spatial types (e.g. line, polyline, polygon, . . .); and include a set of predicates (e.g. *intersects*, *contains*) and functions for spatial calculations (e.g. *distance*). However, the existing commercial products still lack the ability to model and query spatio-temporal contexts for moving objects.

In terms of future work, a particularly challenging problem is the one of query optimization for spatio-temporal databases. We will investigate how to incorporate an indexing schema within the existing ORDBMS (c.f. [2,14]), and develop and experimentally test a hybrid indexing schema which would pick an appropriate access method for a particular environment.

Acknowledgement

We wish to thank Pankaj Agrawal for pointing to us a way to significantly simplify the algorithms for processing the Possibly-Sometime and Always-Possibly

operators. We are grateful to Prof. Eleni Kostis from Truman College, and we also thank the anonymous referees for their valuable comments.

References

1. A. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *19th ACM PODS Conference*, 2000.
2. W. Chen, J. Chow, Y. Fuh, J. grandbois, M. Jou, N. Mattos, B. Tran, and Y. Wang. High level indexing of user – defined types. In *25th VLDB Conference*, 1999.
3. J. R. Davis. *Managing geo - spatial information within the DBMS*, 1998. IBM DB2 Spatial Extender.
4. M. Erwig, M. Schneider, and R. H. G üting. Temporal and spatio – temporal datasets and their expressive power. Technical Report 225-12/1997, Informatik berichte, 1997.
5. ESRI. *ArcView GIS: The Geographic Information System for Everyone*. Environmental Systems Research Institute Inc., 1996.
6. L. Forlizzi, R. H. G üting, E. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. In *ACM SIGMOD*, 2000.
7. Geographic Data Technology Co. *GDT Maps*, 2000. <http://www.geographic.com>.
8. V. Graede and O. G ünther. Multidimensional access methods. *ACM Computing Surveys*, 11(4), 1998.
9. R. H. G üting, M. H. B öhlen, M. Erwig, C. Jensen, N. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and queirying moving objects. *ACM TODS*, 2000.
10. C. M. Hoffman. Solid modeling. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 1997.
11. Intelligent Transportation Systems. *ITS maps*, 2000. <http://www.itsonline.com>.
12. D. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *18th ACM PODS Conference*, 1999.
13. G. Kollios, D. Gunopulos, and V. J. Tsotras. Nearest neighbour queries in a mobile environment. In *STDBM*, 1999.
14. M. Kornacker. High - performance extensible indexing. In *25th VLDB Conference*, 1999.
15. Oracle Corporation. *Oracle8: Spatial Cartridge User's Guide and Reference, Release 8.0.4*, 2000. <http://technet.oracle.com/docs/products/oracle8/doc-index.htm>.
16. D. Pfoser and C. Jensen. Capturing the uncertainty of moving objects representation. In *SSDB*, 1999.
17. D. Pfoser, Y. Theodoridis, and C. Jensen. Indexing trajectories of moving point objects. Technical Report 99/07/03, Dept. of Computer Science, University of Aalborg, 1999.
18. F. P. Preparata and M. I. Shamos. *Computational Geometry: an introduction*. Springer Verlag, 1985.
19. J. O' Rourke. *Computational Geometry in C*. Cambridge University Press, 2000.
20. S. Saltenis and C. Jensen. R-tree based indexing of general spatio-temporal data. Technical Report TR-45, TimeCenter, 1999.
21. S. Saltenis and C. Jensen. Indexing of moving objects for location-based services. Technical Report TR-63, TimeCenter, 2001.

22. S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. Technical Report TR - 44, TimeCenter, 1999.
23. M. Sharir. Algorithmic motion planning. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 1997.
24. A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *13th Int'l Conf. on Data Engineering (ICDE)*, 1997.
25. A.P. Sistla, P. Wolfson, S. Chamberlain, and S. Dao. Querying the uncertain positions of moving objects. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases: Research and Practice*. 1999.
26. W. A. Sutherland. *Introduction to Metric and Topological Spaces*. Oxford University Press, 1998.
27. A. Tansel, J. Clifford, S. Jajodia, A. Segev, and R. Snodgrass. *Temporal Databases: Theory and Implementation*. Benjamin/ Cummings Publishing Co., 1993.
28. J. Tayeb, O. Ulusoy, and O. Wolfson. A quadtree - based dynamic attribute indexing method. *The Computer Journal*, 41(3), 1998.
29. Informix Documentation Team. Informix datablade technology: Transforming data into smart data. Informix Press, 1999.
30. Y. Theodoridis, T. Sellis, A. N. Papadopoulos, and Y. Manolopoulos. Specifications for efficient indexing in spatiotemporal databases. In *IEEE SSDBM*, 1999.
31. Y. Theodoridis, J. R. O. Silva, and M. A. Nascimento. On the generation of spatiotemporal datasets. In *6th Int'l symposium on Large Spatial Databases*, 1999.
32. G. Trajcevski, O. Wolfson, and B. Xu. Modeling and querying trajectories of moving objects with uncertainty. Technical Report UIC - EECS - 01 - 2, May 2001.
33. U S Dept. of Commerce. *Tiger/Line Census Files: Technical Documentation*, 1991.
34. M. Vazirgiannis, Y. Theodoridis, and T. Sellis. Spatiotemporal composition and indexing for large multimedia applications. *Multimedia systems*, 6(4), 1998.
35. M. Vazirgiannis and O. Wolfson. A spatiotemporal model and language for movign objects on road networks. In *SSTD*, 2001.
36. O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendez. Cost and imprecision in modeling the position of moving objects. In *14 -th ICDE*, 1998.
37. O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7, 1999.