Supporting Uncertainty in Moving Objects in Network Databases [*]

Victor Teixeira de Almeida
victor.almeida@fernuni-hagen.de

Ralf Hartmut Güting
rhg@fernuni-hagen.de

Praktische Informatik IV Fernuniversität Hagen
D-58084 Hagen, Germany

ABSTRACT

The management of moving objects has been intensively studied in the recent years. A wide and increasing range of database applications has to deal with spatial objects whose position changes continuously over time, called moving objects. Due to the continuous and unpredictable nature of the movements, they cannot be precisely stored in a database, and therefore objects' positions are sampled, and between these sampled positions interpolation is used. This sampling/interpolation approach results in uncertainty in the objects' positions in the whole trajectory of the moving objects. In this paper, we try to analyze this problem about uncertainty when the movement is restricted to a network. Examples of such movements are cars in highways and trains in railroads. The uncertainty problem is simpler in such cases compared to the free movement in 2-dimensional space. We describe the geometry of the uncertain trajectories of the objects with movement constrained to networks, an extension to the framework in [18, 16] to support uncertainty, as well as some implementation considerations using Secondo, an extensible database system that supports non-standard applications.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design—*data models*; H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Design, Languages

[*]This work was partially supported by a grant Gu 293/8-2 from the Deutsche Forschungsgemeinschaft (DFG), project “Datenbanken für bewegte Objekte” (Databases for Moving Objects)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GIS'05, November 4-5, 2005, Bremen, Germany.
Copyright 2005 ACM 1-59593-146-5/05/0011 ...$5.00.

Keywords

Moving objects databases, network, uncertainty

1. INTRODUCTION

In moving objects databases, due to limited resources, it is impossible for the database server to know the exact positions of all objects all the time. Objects' positions are then sampled, which means that objects send their positions to the server from time to time and between these times, their positions are calculated using interpolation. In most of the works, linear movements are assumed.

Two main models of moving object databases are proposed in the literature: one for querying current and future position of the moving objects (MOST) in [22] and the second for querying past trajectories of the moving objects [9, 11, 18, 3].

As stated in [20] we then have two kinds of uncertainty: measurement error and sampling error. With the help of GPS devices, the measurement error can be very small compared to the sampling error. In the sampling error, the uncertainty depends directly on the update frequency of moving objects. Update policies are presented in [29] and later extended in [30, 13] to try to solve two problems: when moving objects should update their positions and how can the DBMS provide a bound on the error in query processing. The main idea in these models is that costs are assigned to the imprecision and to database updates in a way that when the imprecision cost is higher than the update cost, an update is triggered.

Following these ideas, in [26], the geometry of the moving objects trajectory with uncertainty is proposed, which is a 3-dimensional cylindrical geometry around the moving object trajectory. Operators for spatio-temporal range queries are proposed in order to handle uncertainty with a combination of the keywords *sometimes* and *always*, *everywhere* and *somewhere*, and *possibly* and *definitely*.

The works in [24, 23, 25] propose extensions to the model in [18] to handle uncertainty. Probability density functions are used to model the uncertain positions of moving objects, which can be points, lines, and regions from [18]. Uncertain boolean values are proposed using a special added value *maybe*.

In [1] the model of uncertainty is not fixed, but at each time t, it is assumed that there exists an uncertain region of an object such that it is a closed region and the object can be found only inside it. A probability density function is also used to define the object's position inside this uncertain region. Probabilistic answers to queries are used instead of uncertain boolean values. The probabilistic approach was proposed in [30] for range queries. A probabilistic range query is a query like "Retrieve all objects o which are within the region R" and its result is a set of pairs (o, p) where o is an object and p is the probability that the object is in region R. Actually, the algorithms proposed by the authors retrieve only those pairs for which p is greater than some minimum value.

Most of the works in the moving objects databases literature assume unconstrained movements, i.e. free movements in the 2-dimensional space. It is shown in [16] that, for objects moving in networks the moving object trajectory representation is simpler. Indexing structures for moving objects in networks also achieved better performance taking the network into account ([12, 19, 4]). In [7, 8], update policies that take such constrained movements into account are presented.

In this paper we propose an extension of the model in [16] to handle uncertainty in moving objects in networks. We present the geometry of the uncertain trajectories of the objects with movement constrained to networks and the interaction between such objects in terms of operations.

The contributions of this paper can be summarized as:

- The geometry of the moving objects in networks with uncertainty is presented;
- the framework in [18, 16] is extended containing data types and operations supporting uncertainty; and
- some implementation issues are discussed using the Secondo [17] extensible database management system as well as an indexing approach that can be plugged in to speed up query processing.

This paper is organized as follows: Section 2 presents an overview of the model proposed in [16] with the data types and some operations. Section 3 presents the geometry of the movement with uncertainty given the update policies in [7, 8]. In Section 4 the extension of the framework in [16] is presented as an algebra, where the data types are presented in Section 4.1 and the operations in Section 4.2. Some implementation considerations are presented in Section 5 and, finally, Section 6 concludes the paper.

2. MOVING OBJECTS IN NETWORKS MODEL

In this section we review the model presented in [16] for moving objects in networks. This model is presented as an *algebra*, which is defined in two steps. First, a type system is designed by introducing some basic types as well as some type constructors. For each type in the type system, its semantics is given by defining a *carrier set*. In this paper we define the data types only in their discrete representation, which means that we present only the discrete data model. For a discussion about abstract and discrete models, see [9]. In the second step, a collection of operators is designed over the types in the type system. For each operation, its signature is defined, describing the syntax of the operation, and its semantics is given by defining a function on the carrier sets of the argument types.

The core of the model in [16] are the data types $\underline{network}$, $\underline{gpoint}$, and $\underline{gline}$ to represent the underlying network, network positions, and network regions (lines), respectively. Obviously, the corresponding moving data types for network positions and regions are defined, namely $\underline{moving}(\underline{gpoint})$ and $\underline{moving}(\underline{gline})$.

The type system from [16] is shown in Table 1. The specification of a type system consists of a set of signatures consisting of sorts and operations, as presented in [15]. The sorts are *kinds* and represent sets of data types, and the operations are *type constructors*. For example the kind $SPATIAL$ contains the types $\underline{point}$, $\underline{points}$, $\underline{line}$, and $\underline{region}$. Type constructors may take arguments, for example the $\underline{moving}$ type constructor. It can receive arguments from the kinds $BASE$, $SPATIAL$, and $GRAPH$. Thus, example types in the kind $TEMPORAL$ are $\underline{moving}(\underline{bool})$, $\underline{moving}(\underline{region})$, and $\underline{moving}(\underline{gpoint})$. For the sake of simplicity, we use short versions of these type constructors, namely $\underline{mbool}$, $\underline{mregion}$, and $\underline{mgpoint}$, respectively.

A *network* is composed by *routes* and *junctions*. A route corresponds to roads or highways in real life, and junctions are the connections between two routes. Representing the network in terms of routes and junctions instead of edges and nodes of a graph has several advantages, which are detailed in [16]. The main advantage is that, in this way, the representation of moving object trajectories becomes much more compact.

A route consists of an identifier, a length, a curve describing its geometry in the plane, a route type (the model allows one to distinguish between bi-diretional routes, like highways for example, and simple ones), and a flag indicating how route locations are to be embedded into space. Let the domain of routes be defined as

$$\begin{aligned} Route = \{(id, l, c, type, start) \mid\ & id \in int, l \in real, c \in line, \\ & type \in \{simple, dual\}, \\ & start \in \{smaller, larger\}\} \end{aligned}$$

Let R be a finite set of distinct routes. A *route measure* in R consists of a route identifier and a real number giving a relative position on that route.

$$\begin{aligned} RMeas(R) = \{(rid, d) \mid\ & rid \in int, d \in real, \\ & (rid, l, c, k, s) \in R \text{ such that } 0 \leq d \leq l\} \end{aligned}$$

A *junction* in R is a triple consisting of two route measures in R with distinct route identifiers and a connectivity code, an integer value encoding which movements through the junction are possible.

$$\begin{aligned} Junction(R) = \{(rm_1, rm_2, cc) \mid\ & rm_1, rm_2 \in RMeas(R), \\ & rm_1 = (r_1, d_1), rm_2 = (r_2, d_2), \\ & r_1 < r_2, cc \in int\} \end{aligned}$$

A *network* is a pair $N = (R, J)$ where R is a finite set of distinct routes and J is a finite set of junctions in R. Let $Network$ denote the set of all such pairs.

A *route location* in R is represented by a route measure and a value representing the side of the route. We use a side value *none* for simple routes.

Table 1: **The type system of [GAD05].**

	$\rightarrow$ *BASE*	$\underline{int}$, $\underline{real}$, $\underline{string}$, $\underline{bool}$
	$\rightarrow$ *SPATIAL*	$\underline{point}$, $\underline{points}$, $\underline{line}$, $\underline{region}$
	$\rightarrow$ *GRAPH*	$\underline{gpoint}$, $\underline{gline}$
	$\rightarrow$ *TIME*	$\underline{instant}$
BASE $\cup$ *SPATIAL* $\cup$ *GRAPH*	$\rightarrow$ *TEMPORAL*	$\underline{moving}$, $\underline{intime}$
BASE $\cup$ *TIME*	$\rightarrow$ *RANGE*	$\underline{range}$

$$\begin{aligned} RLoc(R) = \{(rid, d, side) \mid & (rid, d) \in RMeas(R), \\ & side \in \{up, down, none\}, \\ & \forall (rid, l, c, kind, start) \in R : \\ & \quad kind = simple \Leftrightarrow side = none\} \end{aligned}$$

Given a network $N = (R, J)$, the set of network locations is $Loc(N) = RLoc(R)$.

A *route interval* in a network N is basically a pair of route locations on the same route. It can be represented as a tuple $(rid, d_1, d_2, side)$ where $(rid, d_1, side)$ and $(rid, d_2, side)$ are route locations and $d_1 \leq d_2$. Semantically, a route interval $ri = (rid, d_1, d_2, side)$ comprises the set of all route locations $(rid, d, side)$ with $d_1 \leq d \leq d_2$; this set is denoted as $Locs(ri)$.

Two route intervals r_{i1} and r_{i2} are *quasi-disjoint* iff they are either on different routes, or they are on the same route and their sets of route locations are disjoint, i.e., $Locs(r_{i1}) \cap Locs(r_{i1}) = \emptyset$.

A *region* of a network N is a finite set of quasi-disjoint route intervals in N. The set of all possible regions of network N is denoted as $Reg(N)$.

The carrier set for the $\underline{network}$ data type is then

$$D_{\underline{network}} = Network$$

The data types $\underline{gpoint}$ and $\underline{gline}$ obviously depend on existing networks. Let $N = N_1, ..., N_k$ be the set of networks present in the database. The data types $\underline{gpoint}$ and $\underline{gline}$ are then defined with carrier sets

$$\begin{aligned} D_{\underline{gpoint}} &= \{(i, gp) | 1 \leq i \leq k \wedge gp \in (Loc(N_i) \cup \{\perp\})\} \\ D_{\underline{gline}} &= \{(i, gl) | 1 \leq i \leq k \wedge gl \in Reg(N_i)\} \end{aligned}$$

The moving counterparts of the $\underline{gpoint}$ and $\underline{gline}$ use the concept of *sliced representation* for moving objects in [11]. The sliced representation is provided by the $\underline{mapping}$ type constructor which represents a moving object as a set of so-called *temporal units* (*slices*). Each temporal unit consists of a time interval and a description of the temporal development during this time interval via a *unit function* ι, which in this case provides linear functions for the time-dependent location for the $\underline{mgpoint}$ and also linear functions for each route interval boundary for the $\underline{mgline}$ data type.

Some example operation signatures that we can have with these data types are:

$\underline{mgpoint} \times \underline{gline}$	$\rightarrow \underline{mbool}$	**inside**
$\underline{mgpoint} \times \underline{region}$	$\rightarrow \underline{mbool}$	**inside**
$\underline{mgline} \times \underline{mgline}$	$\rightarrow \underline{mgline}$	**union**
$\underline{mgline}$	$\rightarrow \underline{mreal}$	**size**[**length**]
$\underline{mgpoint} \times \underline{gpoint}$	$\rightarrow \underline{mreal}$	**distance**

Their semantics are straightforward and can be found in [16].

3. UNCERTAIN MOVING OBJECTS IN NETWORKS

The most important type of moving object is the moving point object. With this abstraction we can model the movement of cars, trains (if their extent is ignored), people, etc. In this section, we then make a clear definition of the trajectory's geometry of an uncertain moving point object with movement restricted to a network, which we call *uncertain moving graph point*.

First, let us state our assumptions:

- The position at measurement points is precise. This assumption comes from the observation that measurement errors are small compared to the sampling error ([20]). We will then only handle uncertainty between measurement points.
- The moving objects use the ITLU, DTTLU, and STTLU location updates proposed in [7, 8] to send (update) their positions to a server. The ITLU (ID-Triggered Location Update) triggers an update whenever an object moves from one route to another via a junction. The DTTLU (Distance-Threshold-Triggered Location Update) triggers an update whenever the difference between the computed object position and the actual position exceeds a certain predefined threshold ξ. Finally, the STTLU (Speed-Threshold-Triggered Location Update) triggers an update whenever the difference between the moving object's current speed and the last recorded speed exceeds a certain predefined threshold ψ.
- At some predefined points in time, each object knows its exact current position using, for example, a GPS attached to an on-board computer. This position must be known in terms of the network, i.e., it is a network location instead of a 2-dimensional position. This assumption states that from time to time the on-board computer takes the object position from the GPS and calculates its corresponding network position using a map matching algorithm, as presented in [14], [28], [21], and [31]. The time interval between these observations must be small (less than 1 second, for example). This assumption is important to ensure a good precision on detecting route changes and support precisely the update policies explained above.
- Every object has a maximum speed, which is stored as an attribute of the objects.

Under these assumptions we can now state the geometry of the movements. When an update in the DBMS is triggered by an object, four possibilities can occur:

1. The object is new to the system and started its movement;
2. the object changed routes (ITLU);
3. the object position deviation exceeded the threshold ξ (DTTLU); or
4. the object speed deviation exceeded the threshold ψ (STTLU).

In cases 2, 3, and 4 a new piece of the trajectory will be added to the database from the position in the last update to this current one. It is important to note that these pieces of trajectories represent only movements inside the same route, which is ensured by the ITLU update policy. For case 1, the first information about the moving object is stored in the database.

Let us assume that the measurement points are at positions p_1 and p_2 of the specified route r taken at times t_1 and t_2. At time t_1, the position is precise and equals to p_1 with speed v_1. After that, the object can travel at least with $v_1-\psi$ and at most at $v_1+\psi$ or its maximum speed v_{max}, i.e. $min\{v_1+\psi, v_{max}\}$. Let us assume, without loss of generality, that the object is traveling along the route side where positions are increasing, i.e., $p_1 \le p_2$. Thus, the minimum position of the object at some time t, $t_1 < t < t_2$, $p_{min}(t)$ is

$$\begin{aligned} p_{min}(t) = \max\Big\{ & p_1 + \max\{v_1-\psi, 0\}\,(t-t_1)\,, & (1)\\ & p_1 - \xi + \frac{p_2-p_1}{t_2-t_1}\,(t-t_1)\,, & (2)\\ & p_2 - \min\{v_1+\psi, v_{max}\}\,(t_2-t)\Big\} & (3)\end{aligned}$$

Analogously, the maximum position of the object at some time t, $p_{max}(t)$ is

$$\begin{aligned} p_{max}(t) = \min\Big\{ & p_1 + \min\{v_1+\psi, v_{max}\}\,(t-t_1)\,, & (4)\\ & p_1 + \xi + \frac{p_2-p_1}{t_2-t_1}\,(t-t_1)\,, & (5)\\ & p_2 - \max\{v_1-\psi, 0\}\,(t_2-t)\Big\} & (6)\end{aligned}$$

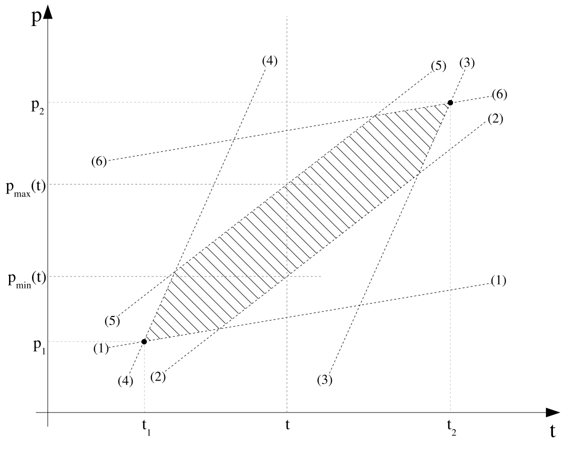

Figure 1: The uncertain geometry of a moving object between two measurement points.

These two equations give us the geometry shown in Figure 1 of a moving object between these two measurement points p_1 and p_2 at times t_1 and t_2, respectively.

4. THE ALGEBRA

In this section we extend the framework in [16] to support uncertainty providing the data types $\underline{uncertain}(\underline{gpoint})$, or $\underline{ungpoint}$ for short[1], and its moving counterpart $\underline{mungpoint}$ that represents static and moving points in networks with uncertainty (Section 4.1). Later, (in Section 4.2) we extend the operations (and semantics) over these new data types.

4.1 Data Types

The data types $\underline{ungpoint}$ and $\underline{mungpoint}$ are inserted into the type system of [16] as can be seen in Table 2. The $UNCERTAIN$ kind with $\underline{uncertain}$ type constructor is added in order to support uncertain data types. As one can note in this type constructor, we also need to propose an extension of the base ($BASE$) and spatial ($SPATIAL$) types to support uncertainty. The need for that will become clear in Section 4.2 where some operators use the data types as arguments and/or return values.

Let us start with the base uncertain data types. For the $\underline{unbool}$ we use the same approach as in [24, 10] where the *maybe* value is added. The carrier set for the $\underline{unbool}$ data type is

$$D_{\underline{unbool}} = \{F, M, T\}$$

For the other uncertain data types we use a possible area with a probability function as proposed in [23] and also used in [2]. The $\underline{unint}$ and $\underline{unreal}$[2] are represented as a finite set of disjoint, non-adjacent intervals ($\underline{range}$ [18]) of $\underline{int}$ and $\underline{real}$ types respectively, assuming the uniform probability distribution function.

The carrier set of the uncertain integer and real numbers is then

$$D_{\underline{un\alpha}} = \underline{range}(\underline{\alpha}),\ \ \alpha \in \{\underline{int}, \underline{real}\}$$

The advantage of representing these data types as described here instead of as in [23, 25] is that no storage for the probability distribution function is necessary, because uniformity is assumed. Moreover, assuming uniformity, the computation becomes simpler. Another important advantage is that holes are possible in this representation.

For the uncertain spatial types $\underline{unpoint}$, $\underline{unpoints}$, $\underline{unline}$, and $\underline{unregion}$ we use the same concept as for the $\underline{int}$ and $\underline{real}$ data types, extended to two dimensions, which means that every uncertain spatial data type will become a (crisp) region that covers all possible positions. We also assume the uniform probability distribution function. In this way, the carrier sets of the uncertain spatial types can be represented as

$$D_{\underline{un\alpha}} = D_{\underline{region}},\ \ \alpha \in SPATIAL$$

[1]We name every uncertain data type starting with the characters 'un'. We do not use only 'u' to avoid conflict with temporal unit names (see [11]).

[2]The data type $\underline{unstring}$ is less important and will be omitted in this framework

Table 2: The extended type system of [GAD05] supporting uncertain data types.

	$\rightarrow BASE$	<u>int</u>, <u>real</u>, <u>string</u>, <u>bool</u>
	$\rightarrow SPATIAL$	<u>point</u>, <u>points</u>, <u>line</u>, <u>region</u>
	$\rightarrow GRAPH$	<u>gpoint</u>, <u>gline</u>
	$\rightarrow TIME$	<u>instant</u>
$BASE \cup SPATIAL \cup GRAPH$	$\rightarrow UNCERTAIN$	<u>uncertain</u>
$BASE \cup SPATIAL \cup GRAPH \cup UNCERTAIN$	$\rightarrow TEMPORAL$	<u>moving</u>, <u>intime</u>
$BASE \cup TIME$	$\rightarrow RANGE$	<u>range</u>

Now we can state the carrier set for the uncertain graph point (<u>*ungpoint*</u>). Given the assumptions in Section 3, an uncertain graph point belongs to a network and the uncertainty is represented only inside the same route. Given a set $N = \{N_1 = (R_1, J_1), ..., N_k = (R_k, J_k)\}$ containing all the networks in the database, the <u>*ungpoint*</u> data type is represented as

$$\begin{aligned} D_{\underline{ungpoint}} = \; & \{(i, rid, side, pos)\} \cup \{\perp\} \mid \\ & 1 \leq i \leq k, \\ & pos \in \underline{unreal}, \\ & \exists (rid, len, cc, kind, sm) \in R_i \text{ such that} \\ & \quad kind = simple \Leftrightarrow side = none \wedge \\ & \quad \forall p \in pos, 0 \leq p \leq len \end{aligned}$$

For representing the geometry of a moving uncertain graph point as a <u>*mungpoint*</u> data type we propose two approaches. Both use the strategy in [11], i.e., representing a moving object as a set of slices, called *temporal units*. Within each slice, the development of the value can be represented by a temporal function. For example, for the <u>*mreal*</u> data type a quadratic polynomial function or the square root of such is used, and for the <u>*mpoint*</u> data type just a simple linear function.

The temporal functions are represented by the generic function ι that evaluates the unit function at a given time instant. Given a non-temporal type α, its corresponding unit type $D_{u\alpha} = Interval(Instant) \times S_\alpha$, where S_α is a suitably defined set and an $Interval(T)$ is an interval over a set $(U, <)$ with a total order with the following definition

$$\begin{aligned} Interval(U) = \{(s, e, lc, rc) \mid s, e \in U, lc, rc \in \underline{bool}, \\ s \leq e, (s = e) \Rightarrow (lc = rc = T)\} \end{aligned}$$

where s and e define the boundaries of the interval and lc and rc define whether the interval is right and/or left-closed.

The function ι_α (or simply ι) is defined as

$$\iota_{\underline{\alpha}} = S_{\underline{\alpha}} \times Instant \rightarrow D_{\underline{\alpha}}$$

In this paper we will use a slightly modified version of the ι function that receives also as argument the time interval of the unit, i.e. the whole unit. The new ι function is then represented as

$$\iota_\alpha = D_{u\alpha} \times Instant \rightarrow D_\alpha$$

The use of the ι function should become clearer when we instantiate it in the two approaches to represent the moving graph point object. The first approach tries to store the whole geometry described in Section 3 inside a temporal unit. Thus, the carrier set of the uncertain graph point temporal unit data type (<u>*uungpoint*</u>) is represented by

$$\begin{aligned} D_{\underline{uungpoint}} = & Interval(Instant) \times \\ & \{(i, rid, side, p_1, p_2, v_{max}, \xi, \psi) \mid \\ & \quad 1 \leq i \leq k, \\ & \quad (rid, p_1, side), (rid, p_2, side) \in RLoc(N_i), \\ & \quad v_{max}, \xi, \psi \in \underline{real}\} \end{aligned}$$

where $ri = (rid, p_1, p_2, side)$ represents a route interval from the begin to the end of the unit time interval, ξ the position deviation threshold, ψ the speed deviation threshold, and v_{max} the maximum speed of the moving object. The evaluation of the temporal unit function (ι) at some time t can be defined as

$$\iota(\; (t_1, t_2, lc, rc), (i, rid, side, p_1, p_2, v_{max}, d) \; , t) = \begin{cases} \perp & (t < t_1) \vee \\ & (t = t_1 \wedge lc = F) \vee \\ & (t > t_2) \vee \\ & (t = t_2 \wedge rc = F) \\ (i, rid, side, \{(p_{min}(t), p_{max}(t), T, T)\}) & \text{otherwise} \end{cases}$$

where the functions $p_{min}(t)$ and $p_{max}(t)$ from Section 3 were used for simplicity of presentation, to avoid repetition.

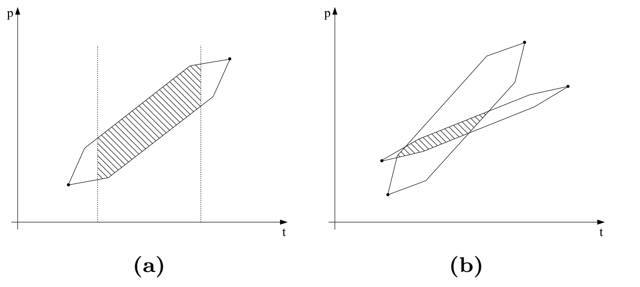

Figure 2: Returning example values of the (a) atperiods and the (b) intersection operators.

This representation is probably the best for predicate processing, but is not suitable for the operations that change the objects' representation, for example **atperiods** and **intersection**. Figure 2 shows the result of these operators applied to sample temporal units, respectively.

The result of these operators cannot be represented by the <u>*uungpoint*</u> data type using the representation described so far. If one chooses this representation, then only predicate operators could be available.

To solve this problem, we propose a second approach where the geometry shown in Figure 1 is divided into pieces and each piece is stored in a different temporal unit. This representation can be seen in Figure 3.

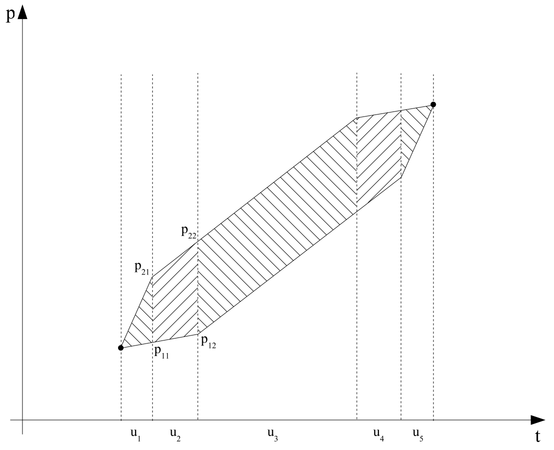

Figure 3: The piece representation of the trajectory of a moving object.

With this representation, a unit now contains a set of disjoint and non-adjacent intervals (<u>*range*</u>) with the interval boundaries in linear movements. One should note that in Figure 3, each unit is composed of only one interval and not by a set of intervals. The need for a set of intervals in the <u>*uungpoint*</u> representation will become clear when we present the operators in Section 4.2, for example, the **minus** operator. In Figure 4 one can see that the result of the **minus** operator can be a set of intervals at some time instants. In fact, at time t', in both examples (a) and (b), the representation of the resulting uncertain point contains two intervals.

The carrier set for the <u>*uungpoint*</u> is then

$$\begin{aligned}D_{\underline{uungpoint}} =& Interval(Instant)\times\\ & i, rid, side, \{(s_{00}, s_{01}, e_{00}, e_{01}, lc_0, rc_0), \ldots,\\ & \quad (s_{n0}, s_{n1}, e_{n0}, e_{n1}, lc_n, rc_n)\} \quad |\\ & 1 \leq i \leq k,\ n \text{ is finite}, n > 0\\ & rid \in \underline{int}, side \in Side,\\ & s_{j0}, s_{j1}, e_{j0}, e_{j1} \in \underline{real}, 0 \leq j \leq n\end{aligned}$$

where s_{j0}, s_{j1}, e_{j0}, and e_{j1} are the coefficients for the set of linear equations of the movements of the intervals' start and end positions, and lc_j and rc_j are the flags that indicate whether the corresponding interval is right and/or left closed, respectively. The temporal function ι is consequently defined as

$$\begin{aligned}\iota(\ & (t_1, t_2, lc, rc), (i, rid, side, \{(s_{00}, s_{01}, e_{00}, e_{01}, lc_0, rc_0), \ldots,\\ & (s_{n0}, s_{n1}, e_{n0}, e_{n1}, lc_n, rc_n)\})\ \ , t) =\\ & (i, rid, side, \{(s_{00} + s_{01}(t - t_1), e_{00} + e_{01}(t - t_1), lc_0, rc_0), \ldots,\\ & (s_{n0} + s_{n1}(t - t_1), e_{n0} + e_{n1}(t - t_1), lc_n, rc_n)\})\end{aligned}$$

Another advantage of this second approach is that it is generic enough to represent moving uncertain graph points without assuming that the ITLU, DTTLU, and STTLU policies presented in Section 3 are used. This means that different combinations of these policies or even other update policies, leading to different geometries than the one in Figure 1, could be used and the representation of the moving uncertain graph points is still applicable.

4.2 Operators

In this section we extend the set of operations in [16] to support the uncertain data types <u>*ungpoint*</u> and its moving counterpart <u>*mungpoint*</u>.

To describe the semantics of the operators, we will use the same approach as in [18], where $u, v, \ldots$ represent values of single valued types, and $U, V, \ldots$ values of set valued types, where u or U represent the first argument of the operator, v or V the second, and so on. We use μ for values that range over moving objects and t and T for values that range over instants and period values. We also make use of the set $\rho(S)$ that represents the set of all (infinite) values that a set valued type S can assume.

We must first state the boolean operators containing the uncertain boolean type, namely **not**, **or**, and **and**. Their truth tables can be seen in Table 3 (a), (b), and (c), respectively.

Table 3: The truth table for the logic operators (a) not, (b) or, and (c) and.

u	**not** u
T	F
M	M
F	T

(a)

or	F	M	T
F	F	M	T
M	M	M	T
T	T	T	T

(b)

and	F	M	T
F	F	F	F
M	F	M	M
T	F	M	T

(c)

4.3 Operations on Non-Temporal Types

The first set of operations are the operations to access a <u>*ungpoint*</u> value. These operations can be seen in Table 4.

Table 4: Operations to access values of a <u>*ungpoint*</u> data type.

Signature		Result	Operator
<u>*ungpoint*</u>		$\rightarrow$ <u>*int*</u>	**route**
Semantics:	if $U \neq \perp$ then $U.rid$ else $\perp$		
<u>*ungpoint*</u>		$\rightarrow$ <u>*unreal*</u>	**pos**
Semantics:	if $U \neq \perp$ then $U.pos$ else $\perp$		
<u>*ungpoint*</u>		$\rightarrow$ <u>*int*</u>	**side**
Semantics:	if $U \neq \perp$ then $U.side$ else $\perp$		
<u>*ungpoint*</u> $\times$ <u>*int*</u>[*routeId*]		$\rightarrow$ <u>*bool*</u>	**inside**
Semantics:	if $U \neq \perp \wedge v \neq \perp$ then $U.rid = v$ else $\perp$		
<u>*network*</u> $\times$ <u>*int*</u>[*routeId*]$\times$ <u>*int*</u> $\times$ <u>*unreal*</u>		$\rightarrow$ <u>*ungpoint*</u>	**ungpoint**
Semantics:	$(U.id, v, w, x)$		

The main changes from [16] are the operators **pos** and **ungpoint**. The first one returns the position of the uncertain graph point inside a route as an uncertain real value (<u>*unreal*</u>). The operator **ungpoint** is used to create an uncertain graph point and it receives an uncertain real value (<u>*unreal*</u>) as argument for the relative position of the object inside a route. These operators are examples for the need of the <u>*uncertain*</u> type constructor applied to the base types, more specifically the <u>*real*</u> data type.

Table 5: Generic predicate operations.

Unary		
$\underline{ungpoint}$	$\rightarrow \underline{bool}$	**isempty**
Semantics:	$U = \perp$	
Binary		
$\underline{gpoint} \times \underline{ungpoint}$	$\rightarrow \underline{unbool}$	$=$
Semantics:	if $u.pos \notin \rho(V.pos)$ then F else if $V.pos = \{u.pos\}$ then T else M	
$\underline{ungpoint} \times \underline{ungpoint}$	$\rightarrow \underline{unbool}$	$=$
Semantics:	if $\rho(U.pos) \cap \rho(V.pos) \neq \emptyset$ then F else if $\exists u \in \underline{real} : U.pos = \{u\} \wedge$ $\exists v \in \underline{real} : V.pos = \{v\} \wedge u = v$ then T else M	
$\underline{gpoint} \times \underline{ungpoint}$	$\rightarrow \underline{unbool}$	$\neq$
Semantics:	if $u.pos \notin \rho(V.pos)$ then T else if $V.pos = \{u.pos\}$ then F else M	
$\underline{ungpoint} \times \underline{ungpoint}$	$\rightarrow \underline{unbool}$	$\neq$
Semantics:	if $\rho(U.pos) \cap \rho(V.pos) \neq \emptyset$ then T else if $\exists u \in \underline{real} : U.pos = \{u\} \wedge$ $\exists v \in \underline{real} : V.pos = \{v\} \wedge u = v$ then F else M	
$\underline{ungpoint} \times \underline{gline}$	$\rightarrow \underline{unbool}$	**inside**
Semantics:	if $\rho(U.pos) \subset \rho(V.pos)$ then T else if $\rho(U.pos) \cap \rho(V.pos) \neq \emptyset$ then M else F	

Table 6: Generic operations.

Set operations		
$\underline{gpoint} \times \underline{ungpoint}$	$\rightarrow \underline{pair}(\underline{gpoint})$	**intersection**
Semantics:	if $V.pos = \{u.pos\}$ then (u, u) else if $u.pos \in \rho(V.pos)$ then $(\perp, u)$ else $(\perp, \perp)$	
$\underline{ungpoint} \times \underline{ungpoint}$	$\rightarrow \underline{pair}(\underline{ungpoint})$	**intersection**
Semantics:	if $\rho(U.pos) \cap \rho(V.pos) \neq \emptyset$ then $(\perp, \perp)$ else if $\exists u \in \underline{real} : U.pos = \{u\} \wedge$ $\exists v \in \underline{real} : V.pos = \{v\} \wedge u = v$ then (U, U) else $(\perp, (U.i, U.rid, U.side, \rho(U.pos) \cap \rho(V.pos)))$	
$\underline{ungpoint} \times \underline{gline}$	$\rightarrow \underline{pair}(\underline{ungpoint})$	**intersection**
Semantics:	if $\rho(U.pos) \subseteq \rho(V.gl)$ then (U, U) else if $\rho(U.pos) \cap \rho(V.gl) \neq \emptyset$ then $(\perp, \rho(U.pos) \cap \rho(V.gl))$ else $(\perp, \perp)$	
$\underline{gpoint} \times \underline{ungpoint}$	$\rightarrow \underline{pair}(\underline{gpoint})$	**minus**
Semantics:	if $V.pos = \{u.pos\}$ then $(\perp, \perp)$ else if $u.pos \in \rho(V.pos)$ then $(\perp, u)$ else (u, u)	
$\underline{ungpoint} \times \underline{gpoint}$	$\rightarrow \underline{pair}(\underline{ungpoint})$	**minus**
Semantics:	if $U.pos = \{v.pos\}$ then $(\perp, \perp)$ else if $v.pos \in \rho(U.pos)$ then $(\perp, U)$ else (U, U)	
$\underline{ungpoint} \times \underline{ungpoint}$	$\rightarrow \underline{pair}(\underline{ungpoint})$	**minus**
Semantics:	if $\rho(U.pos) \cap \rho(V.pos) \neq \emptyset$ then (U, U) else if $\exists u \in \underline{real} : U.pos = \{u\} \wedge$ $\exists v \in \underline{real} : V.pos = \{v\} \wedge u = v$then $(\perp, \perp)$ else $(\perp, \underline{(U.i, U.rid, U.side, \rho(U) \backslash \rho(V))})$	
$\underline{ungpoint} \times \underline{gline}$	$\rightarrow \underline{pair}(\underline{ungpoint})$	**minus**
Semantics:	if $\rho(U.pos) \subseteq \rho(V.gl)$ then $(\perp, \perp)$ else if $\rho(U.pos) \cap \rho(V.gl) \neq \emptyset$ then $(\perp, \rho(U.pos) \backslash \rho(V.gl))$ else (U, U)	
Aggregation		
$\underline{ungpoint}$	$\rightarrow \underline{gpoint}$	**single**
Semantics:	if $\exists u \in \underline{real} : U.pos = \{u\}$ then $(U.i, U.rid, u, U.side)$ else $\perp$	
Distance		
$\underline{gpoint} \times \underline{ungpoint}$	$\rightarrow \underline{unreal}$	**distance**
Semantics:	$\{dist_N(u, (V.i, V.rid, v, V.side)) \mid v \in \rho(V.pos)\}$	
$\underline{ungpoint} \times \underline{ungpoint}$	$\rightarrow \underline{unreal}$	**distance**
Semantics:	$\{dist_N((U.i, U.rid, u, U.side), (V.i, V.rid, v, V.side)) \mid u \in \rho(U), v \in \rho(V)\}$	
$\underline{ungpoint} \times \underline{gline}$	$\rightarrow \underline{unreal}$	**distance**
Semantics:	$\{dist_N((U.i, U.rid, u, U.side), v) \mid u \in \rho(U.pos), v \in V.gl\}$	

In the sequel, for sake of clearness, we will omit the tests for undefined values ($\perp$). We are able now to present the generic operations. Tables 5 and 6 show these operations.

Some comments can be stated. First, one can note that the semantics of the operators $=$ and $\neq$ are consistent with the truth table for the **not** operator of the $\underline{unbool}$ data type (Table 3 (a)), i.e., $a \neq b \Leftrightarrow \neg(a = b)$.

Some operations, like **intersection** and **minus** for example, return objects as result values. The M (*maybe*) result for predicates must be somehow represented in terms of objects. For the **intersection** operation, for example, when two objects u and v possibly intersect each other, it returns two possible values: $\perp$ and $u \cap v$. We represent this kind of resulting object as a pair of objects using the **pair** type constructor. Three possibilities can occur: the pair is simply undefined, which is represented semantically by $(\perp, \perp)$; an object o and the undefined value are contained in the pair, represented by $(\perp, o)$; and the pair contains a single object o, represented by (o, o). These three possibilities correspond to the F, M, and T returning values for predicates, respectively.

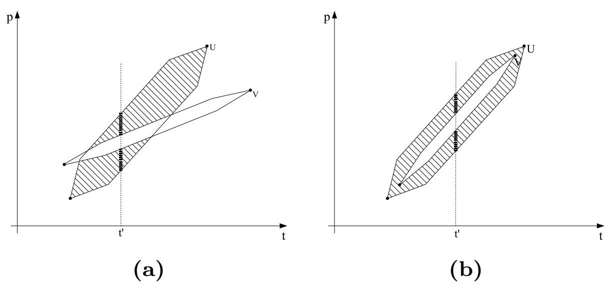

Figure 4: Returning example values (a) and (b) of the minus operator (U minus V).

The operator **minus** is responsible for the representation of the uncertain graph point ($\underline{ungpoint}$) to be a set of intervals containing possible positions inside a route. Figure 4 shows two examples of the **minus** operation's result applied to two units U and V, where the operation is U **minus** V. We can see in this figure that, at some time instants (time t' for example), the representation of the resulting uncertain graph point is not a simple continuous interval, but a set of disjoint and non-adjacent ones.

The aggregate operator **single** retrieves the crisp graph point ($\underline{gpoint}$) if there is no uncertainty in the uncertain graph point.

Finally, the operation **distance** returns the distance from two objects as an uncertain real, containing all possible (minimum network) distances, represented by $d_N(p_1, p_2)$, where p_1 and p_2 are graph points. Since we modeled the network as a directed graph, the set of all possible distances can be discontinuous. For that we need the representation of the $\underline{unreal}$ data type to be able to support discontinuity.

4.4 Operations on Temporal Types

Let us proceed with the operations on temporal types. This set of operations always contain a type belonging to the kind $TEMPORAL$, and the most important ones are those that are applied to moving objects, e.g. the $\underline{mungpoint}$.

First, all operations defined so far are subject to *lifting.* For lifting, we mean that an operation with signature

$$\alpha_1 \times \ldots \times \alpha_n \rightarrow \beta$$

can be lifted in such a way that operations with signature

$$\alpha'_1 \times \ldots \times \alpha'_n \rightarrow \underline{moving}(\beta)$$

are possible. Here, $\alpha'_i \in \{\alpha_i, \underline{moving}(\alpha_i)\}$, which means that each of the argument types may become time-dependent making the result time-dependent as well. Figure 2(b) shows an example of the lifted version of the **intersection** operation with signature

$$\underline{mungpoint} \times \underline{mungpoint} \rightarrow \underline{mungpoint} \quad \textbf{intersection}$$

The semantics of lifting is described in detail in [18]. As examples, one can define the following operations:

$\underline{mungpoint} \times \underline{gline}$	$\rightarrow \underline{munbool}$	**inside**
$\underline{mungpoint} \times \underline{mgline}$	$\rightarrow \underline{munbool}$	**inside**
$\underline{gpoint} \times \underline{mungpoint}$	$\rightarrow \underline{munreal}$	**distance**
$\underline{mungpoint} \times \underline{mungpoint}$	$\rightarrow \underline{munreal}$	**distance**

The second set of operations are the generic operations, which can be seen in Table 7.

The domain operator **deftime** returns the times when the moving uncertain graph point is defined, where **trajectory** projects its value into the network space. The operators **inst** and **val** just return the two components of the $\underline{intime}$ value.

The movement can be restricted to a single time instant with the **atinstant** operator or to a set of time intervals with the **atperiods** operator. The operators **initial** and **final** just return the first and last value pairs, respectively. Operation **present** checks if the moving value exists at a given time instant or at a given set of time intervals. The function $f_{\textbf{atperiods}}$ evaluates the **atperiods** operator's semantics.

In the framework in [18, 16], the operator **at** restricts the movement to the times when its value lies within the second argument. This means evaluating a boolean function, equality for example, and to restrict the movement to when this function is T. With uncertainty, the function that needs to be evaluated returns an uncertain boolean value, and therefore we should also be able to return in the **at** operator the times where this function is evaluated to M. We then use the same approach presented in [27] using the terms *possibly* and *definitely.* The **at** operator will be divided into **possibly_at** and **definitely_at**. The same approach applies to the **when** operator. The **passes** operator checks whether the moving value ever assumed one of the values passed as a second argument.

Table 7: Generic temporal operations.

Projection to domain and range		
$\underline{mungpoint}$	$\rightarrow \underline{periods}$	**deftime**
Semantics:	$dom(\mu)$	
$\underline{mungpoint}$	$\rightarrow \underline{gline}$	**trajectory**
Semantics:	$rng(\mu)$	
$\underline{intime}(\underline{ungpoint})$	$\rightarrow \underline{instant}$	**inst**
Semantics:	t, where $u = (t, V)$	
$\underline{intime}(\underline{ungpoint})$	$\rightarrow \underline{ungpoint}$	**val**
Semantics:	V, where $u = (t, V)$	
Interaction with domain and range		
$\underline{mungpoint} \times \underline{instant}$	$\rightarrow \underline{intime}(\underline{ungpoint})$	**atinstant**
Semantics:	$(t, \mu(t))$	
$\underline{mungpoint} \times \underline{periods}$	$\rightarrow \underline{mungpoint}$	**atperiods**
Semantics:	$(t, y) \in \mu \| t \in T$	
$\underline{mungpoint}$	$\rightarrow \underline{intime}(\underline{ungpoint})$	**initial**
Semantics:	$\lim_{t \rightarrow inf(dom(\mu))} \mu(t)$	
$\underline{mungpoint}$	$\rightarrow \underline{intime}(\underline{ungpoint})$	**final**
Semantics:	$\lim_{t \rightarrow sup(dom(\mu))} \mu(t)$	
$\underline{mungpoint} \times \underline{instant}$	$\rightarrow \underline{bool}$	**present**
Semantics:	$\mu(t) \neq \perp$	
$\underline{mungpoint} \times \underline{periods}$	$\rightarrow \underline{bool}$	
Semantics:	$f_{\underline{atperiods}}(\mu, T) \neq \emptyset$	
$\underline{mungpoint} \times \underline{gpoint}$	$\rightarrow \underline{mungpoint}$	**definitely_at**
Semantics:	$\{(t, y) \in \mu \mid f_{=}(y, v) = T\}$	
$\underline{mungpoint} \times \underline{ungpoint}$	$\rightarrow \underline{mungpoint}$	
Semantics:	$\{(t, y) \in \mu \mid f_{=}(y, V) = T\}$	
$\underline{mungpoint} \times \underline{gline}$	$\rightarrow \underline{mungpoint}$	
Semantics:	$\{(t, y) \in \mu \mid f_{inside}(y, V) = T\}$	
$\underline{mungpoint} \times \underline{gpoint}$	$\rightarrow \underline{mungpoint}$	**possibly_at**
Semantics:	$\{(t, y) \in \mu \mid f_{=}(y, v) \neq F\}$	
$\underline{mungpoint} \times \underline{ungpoint}$	$\rightarrow \underline{mungpoint}$	
Semantics:	$\{(t, y) \in \mu \mid f_{=}(y, V) \neq F\}$	
$\underline{mungpoint} \times \underline{gline}$	$\rightarrow \underline{mungpoint}$	
Semantics:	$\{(t, y) \in \mu \mid f_{inside}(y, V) \neq F\}$	
$\underline{mungpoint} \times \underline{gpoint}$	$\rightarrow \underline{unbool}$	**passes**
Semantics:	if $f_{\textbf{definitely\_at}}(\mu, v) \neq \emptyset$ then T else if $f_{\textbf{possibly\_at}}(\mu, v) \neq \emptyset$ then M else F	
$\underline{mungpoint} \times \underline{ungpoint}$	$\rightarrow \underline{unbool}$	
Semantics:	if $f_{\textbf{definitely\_at}}(\mu, V) \neq \emptyset$ then T else if $f_{\textbf{possibly\_at}}(\mu, V) \neq \emptyset$ then M else F	

5. IMPLEMENTATION ISSUES

In this section we present some implementation techniques for the main data types in the framework, namely $\underline{ungpoint}$ and $\underline{mungpoint}$. We also address an indexing approach for the $\underline{mungpoint}$ data type to speed up query processing.

5.1 Data Types

The uncertain data types are implemented as a new algebra in the Secondo DBMS ([5, 17]). Secondo offers a specific concept for the implementation of persistent attribute data types, where they are represented as a *root record* and can contain some *database arrays.* Database arrays are basically persistent arrays of elements with fixed size, and are implemented on top of a concept called FLOB (Faked Large Object) described in [6], which means that they are automatically either represented *inline* with the tuple representation, or outside in a separate list of pages, depending on their sizes.

The $\underline{ungpoint}$ is represented as follows:

```
ungpoint: record {
  i: int;
  rid: int;
  side: {up, down, none};
  pos: DBArray of Interval(real)
};
```

where $Interval(\underline{real})$ is represented by the tuple (r_1, r_2, lc, rc).

For representing the *<u>mungpoint</u>* we use the sliced representation presented in Section 4.1. We then need to state the representation of the corresponding temporal unit *<u>uungpoint</u>*.

```
uungpoint: record {
  interval: Interval(Instant);
  rid: int;
  side: {up, down, none};
  pos: DBArray of UInterval(real)
};
```

where $UInterval(\underline{real})$ is represented by the tuple (s_0, s_1, e_0, e_1, lc, rc).

Having described the unit data type, the *<u>mapping</u>* type constructor can be directly applied creating a structure that contains a database array (*DBArray*) of units, a *deftime* field (of type *<u>periods</u>*) and some additional information. For more details about the implementation of the *<u>mapping</u>* type constructor, see [3].

5.2 Indexing

For indexing the movements' trajectories, we propose an extension in the MON-Tree presented in [4]. The index structure of the MON-Tree contains a top-level R-Tree indexing the routes, with leaf nodes pointing to bottom-level R-Trees indexing the movements of the objects in the corresponding route. The movement is represented by the position interval (p_1, p_2) and a time interval (t_1, t_2), where $0 \leq p_1, p_2 \leq 1$. These two values p_1 and p_2 store the relative position of the objects inside of the route's polyline at times t_1 and t_2, respectively.

The storage approach that divides the geometry in Figure 3 into pieces can be represented by the time interval (t_1, t_2) and four points $(p_{i11}, p_{i12}, p_{i21}, p_{i22})$ that represent the trapezoid of each disconnected part i of the temporal unit.

In Figure 3, these points are represented in the unit u_2. In this figure, all units are represented by only one part, which makes the subscript i not necessary. One should note that some of these points can be equal, for example in the Figure 3, $p_{11} = p_{21}$ at u_1 and $p_{12} = p_{22}$ at u_5.

It is not necessary to store the MBR of their geometry, since it is very easy to calculate it given the four points. Thus, the leaf nodes of the bottom R-Trees contain the following information: $\langle t_1, t_2, p_{i11}, p_{i12}, p_{i21}, p_{i22}, mopt\rangle$, for every i-th disconnected part of the temporal unit, where *mopt* represents a pointer to the moving object complete representation.

The algorithm for the range query proposed in [4] remains untouched, except for the last test of intersection in the entries of the bottom R-Tree leaf nodes. This test first checks whether the query rectangle set contains the region represented by $(p_{i11}, p_{i12}, p_{i21}, p_{i22})$ and (t_1, t_2) and returns T if so. Otherwise, if they intersect, it returns M, and F for all other cases. It is straightforward to adapt this algorithm for other predicates, such as *inside* for example.

6. CONCLUSIONS

In this paper we presented an extension to the framework in [18, 16] to support uncertainty mainly in the moving point objects data type. The geometry of the moving objects with movement restricted to networks is presented using new update policies recently presented in [7, 8]. Implementation issues are presented to show that the model proposed in the paper is feasible to be implemented as an extension to the previous one in [16].

7. REFERENCES

[1] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):1112–1127, 2004.

[2] R. Cheng, S. Prabhakar, and D. Kalashnikov. Querying imprecise data in moving object environments. In *Proc. of the IEEE International Conference on Data Engineering (ICDE)*, pages 723–725, 2003.

[3] J. A. Cotelo Lema, L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. Algorithms for moving objects databases. *The Computer Journal*, 46(6):680–712, 2003.

[4] V. T. de Almeida and R. H. Güting. Indexing the trajectories of moving objects in networks. *GeoInformatica*, 9(1):33–60, 2005.

[5] S. Dieker and R. H. Güting. Plug and play with query algebras: SECONDO - a generic DBMS development environment. In *Proc. of the Intl. Database Engineering and Applications Symposium (IDEAS)*, pages 380–392, 2000.

[6] S. Dieker, R. H. Güting, and M. R. Luaces. A tool for nesting and clustering large objects. In *Proc. of the 12th Intl. Conf. on Scientific and Statistical Database Management (SSDBM)*, pages 169–181, 2000.

[7] Z. Ding and R. H. Güting. Managing moving objects on dynamic transportation networks. In *Proc. of the 16th Intl. Conf. on Scientific and Statistical Database Management (SSDBM)*, pages 287–296, 2004.

[8] Z. Ding and R. H. Güting. Uncertainty management for network constrained moving objects. In *Proc. of the 15th Intl. Workshop on Database and Expert Systems Applications (DEXA)*, pages 411–421, 2004.

[9] M. Erwig, R. H. Güting, M. Schneider, and M. Vazirgiannis. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *GeoInformatica*, 3(3):269–296, 1999.

[10] M. Erwig and M. Schneider. Vague regions. In *Proc. of the 5th Intl. Symp. of Advances in Spatial Databases (SSD)*, pages 298–320, 1997.

[11] L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, volume 29(2), pages 319–330, 2000.

[12] E. Frentzos. Indexing objects moving on fixed networks. In *Proc. of the 8th Intl. Symp. on Spatial and Temporal Databases (SSTD)*, pages 289–305, 2003.

[13] H. Gowrisankar and S. Nittel. Reducing uncertainty in location prediction of moving objects in road networks (extended abstract). In *Proc. of the 2nd Intl. Conf. on Geographic Information Science (GIScience)*, 2002.

[14] J. S. Greenfeld. Matching GPS observations to locations on a digital map. In *Proc. of the 81th Annual Meeting of the Transportation Research Board*, 2002.

[15] R. H. Güting. Second-order signature: A tool for specifying data models, query processing, and

optimization. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 277–286, 1993.

[16] R. H. Güting, V. T. de Almeida, and Z. Ding. Modeling and querying moving objects in networks. *VLDB Journal*, 2005. To appear. Also available as Technical Report 308, Fernuniversitat Hagen, Fachbereich Informatik at `http://www.informatik.fernuni-hagen.de/import/pi4/papers/PaperMon.pdf`.

[17] R. H. Güting, T. Behr, V. T. de Almeida, Z. Ding, F. Hoffmann, and M. Spiekermann. SECONDO: An extensible DBMS architecture and prototype. Technical Report 313, Fernuniversität Hagen, Fachbereich Informatik, 2004. Available at `http://www.informatik.fernuni-hagen.de/import/pi4/papers/Secondo04.pdf`.

[18] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM Transactions on Database Systems (TODS)*, 25(1):1–42, 2000.

[19] D. Pfoser and C. S. Jensen. Indexing of network constrained moving objects. In *Proc. of the 11th Intl. Symp. on Advances in Geographic Information Systems (ACM-GIS)*, pages 25–32, 2003.

[20] D. Pfoser and C. S. Jensen. Capturing the uncertainty of moving-object representations. In *Proc. of Advances in Spatial Databases, 6th Intl. Symp. (SSD)*, pages 111–132, 1999.

[21] M. A. Quddus, W. Y. Ochieng, L. Zhao, and R. B. Noland. A general map matching algorithm for transport telematics applications. *GPS Solutions Journal*, 7(3):157–167, 2003.

[22] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Querying the uncertain position of moving objects. In *Temporal Databases: Research and Practice*, volume 1399, pages 310–337. LNCS, 1998.

[23] E. Tøssebro and M. Nygård. An advanced discrete model for uncertain spatial data. In *Proc. 3rd Int. Conference on Web-Age Information Management (WAIM)*, pages 37–51, 2002.

[24] E. Tøssebro and M. Nygård. Uncertainty in spatiotemporal databases. In *Proc. 2nd Biennial Int. Conference on Advances in Information Systems (ADVIS)*, pages 43–53, 2002.

[25] E. Tøssebro and M. Nygård. A medium complexity discrete model for uncertain spatial data. In *Proc. 7th Int. Database Engineering and Applications Symposium (IDEAS)*, pages 376–384, 2003.

[26] G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing uncertainty in moving objects databases. *ACM Trans. Database Syst.*, 29(3):463–507, 2004.

[27] G. Trajcevski, O. Wolfson, F. Zhang, and S. Chamberlain. The geometry of uncertainty in moving objects databases. In *Proc. of the 8th Intl. Conf. on Extending Database Technology (EDBT)*, pages 233–250, 2002.

[28] 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.

[29] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendez. Cost and imprecision in modeling the position of moving objects. In *Proceedings of the 14th Intl. Conf. on Data Engineering*, pages 588–596, 1998.

[30] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7(3):257–387, 1999.

[31] H. Yin and O. Wolfson. A weight-based map matching method in moving objects databases. In *Proc. of the 16th Intl. Conf. on Scientific and Statistical Database Management (SSDBM)*, pages 235-244, 2004.