

Managing Trajectories of Moving Objects as Data Streams

Kostas Patroumpas

Timos Sellis

School of Electrical and Computer Engineering
National Technical University of Athens
Zographou 15773, Athens, Hellas
{kpatro, timos}@dbnet.ece.ntua.gr

Abstract

The advent of modern monitoring applications, such as location-based services, presents several new challenges when dealing with continuously evolving spatiotemporal information. Frequent updates in the positions of moving objects, unexpected fluctuations in data volume and the requirement for real-time responses to continuous spatiotemporal queries indicate the limitations of traditional database systems. We attempt to model management of moving objects with the underlying assumption that their trajectories are essentially continuous, time-varying and possibly unbounded data streams. We propose a basic framework for managing trajectory streams, and suggest the introduction of enhanced constructs for advanced query capabilities. Our first experience with querying moving objects in two data stream prototype systems is promising for the feasibility and extensibility of this approach.

1 Introduction

In monitoring applications, like sensor networks, financial tickers, Web search engines etc., data arrives into the system from multiple sources as *online data streams*. This persistent data flow through a network requires real-time responses to users' requests. Multiple *continuous queries* remain active for long and they must be evaluated incrementally, keeping up with the arrival rate of the data. The typical pull-based model for transactions and queries utilized in the conventional DBMS paradigm gives way to a push-based architecture, which has been adopted by several prototype systems currently being developed for data stream management [ACC+03, BBD+02, CCD+03].

Applications for *location-based services* (e.g. automatic

vehicle location) have also greatly benefited from recent advances in the fields of telecommunications and Global Positioning System (GPS). The possibility to locate objects in continuous movement (e.g., vehicles or humans) in real-time and with improved accuracy has boosted up the trend for developing moving objects databases, and consequently, has assisted research in the broader area of spatiotemporal phenomena. In particular, topics such as representation, indexing and querying moving objects have attracted much research effort, mainly towards the foundation of an appropriate model for the efficient management of their *trajectories*.

We believe that the research fields of data streams and management of moving objects can naturally come together. In particular, ideas and techniques originally proposed for data streams can also apply to trajectories:

- Why not maintain trajectory elements in memory for immediate processing, rather than just storing them in a spatiotemporal database for offline management?
- Is it possible to formulate continuous queries over trajectories and provide their results incrementally?
- If the current status of movement is what matters most, would it then be reasonable to ignore the details concerning remote parts of trajectories and instead focus on windows of the most recent features?

We think that the answer to these indicative questions is affirmative according to the model we propose.

Indeed, by sampling the trajectories of moving objects and thus compiling their successive positions across time, a data stream of spatiotemporal features can be created. Sampling rates may fluctuate, data could get lost during transmission to the system, or even superfluous (i.e. very dense) measurements might be observed at locations. Apparently, a DBMS cannot store data in its entirety, since the arrival rate is unpredictable and the expected amount of data rather high, as tuples are piling up continuously into the processing mechanism from their sources. Possible variations at the arrival rate of incoming data might impose excessive requirements in system resources, especially with respect to memory allocation. It is also likely that the system cannot cope with a sudden burst of data and thus become unable to provide responses

Copyright held by the author(s).

Proceedings of the Second Workshop on Spatio-Temporal Database Management (STDBM'04), Toronto, Canada, August 30th, 2004

to queries in time; techniques such as *load shedding* [ACC+03] or *approximations* [BBD+02] have been devised in an attempt to remedy the problem. Clearly, a traditional DBMS, yet enhanced with special constructs for spatiotemporal data management, would prove inadequate to this purpose, especially in terms of online processing of queries that remain active for a long time.

Our fundamental assumption about streams of trajectory elements paves the way towards the formulation of a query language for manipulating *trajectory streams*. Noticeably, spatial information is not simply some invariable coordinates, as in the case of sensor networks covering an area. On the contrary, it is precisely the continuous movement that is of great interest, as well as any spatiotemporal interactions among moving objects. Hence, this language should bring together the clarity and the processing functionality of a stream query language along with operations particularly designed to capture interesting spatiotemporal interrelationships, e.g. “objects crossing the boundaries of a polygon area”.

This paper describes a novel approach to the management of trajectories of moving objects, considering them as online data streams. To the best of our knowledge, there is no prior work making out the case for utilizing a stream query language so as to express continuous queries over dynamic spatiotemporal information. We believe that a declarative language is the most appropriate for the trajectory stream model proposed here. The semantics of an SQL-like language can be comfortably modified so as to apply to streams, whereas spatiotemporal operations may be incorporated with proper adjustments or additions. New language constructs, namely several types of *windows*, are worth introducing so as to facilitate managing and querying trajectories as data streams.

The remainder of this paper is structured as follows. Section 2 presents a brief overview of previous work in the research areas of data streams and moving objects databases. In Section 3, we introduce elements of a model for representing trajectories as data streams, which we call trajectory streams. In Section 4, we briefly report on formulating indicative spatiotemporal continuous queries over trajectories in two data stream prototype systems. In Section 5, we discuss the necessity for new language constructs, mainly special-purpose window operators for trajectory streams. Finally, Section 6 concludes the paper and suggests ideas for future research.

2 Related Work

Several data stream prototypes are already at an advanced implementation stage, introducing a lot of new concepts with regard to the architecture of such a data management system. In parallel, several suggestions have been made for a stream query language. Most researchers tend to adopt a declarative SQL-like language. A *Continuous Query Language* (CQL) is being developed for the STREAM prototype [STR04], supporting management of

dynamic streams and static relations, as well as mappings between them [ABW03]. In addition, CQL provides the means for query optimization, a crucial issue when numerous continuous queries are active at the same time. Similarly, TelegraphCQ introduces *StreaQuel* [TEL04], which extends semantics of relational operators to streams and allows transformation between streams. There is also support for windows that specify portions of the infinite stream. Conversely, AURORA [AUR04] follows the data flow paradigm, and makes use of a built-in query algebra *SQuAl* in order to construct a network of operators. These primitive operators take part in the global execution plan maintained for all active continuous queries.

Concerning management of spatiotemporal information, there are two different perspectives regarding time. From a historical point of view, an algebra for moving objects has been proposed [KSF+03] with data types, operators and predicates over trajectories. These constructs adhere to SQL and they may be implemented as an extension to a traditional DBMS [EGSV99]. In contrast, other models attempt to determine the current status of objects or to predict their future positions. To this goal, a *Future Temporal Logic* language was introduced [SWCD97], enhanced with spatiotemporal constructs for suitable query specification. Indexing techniques for moving objects or trajectories abound, with scalability being the main challenging task. Indeed, the larger the number of moving objects, the more expensive the maintenance of indexing structures due to frequent position updates.

Some recent papers focus on evaluating continuous queries over continuously moving objects. Most of them deal with specific query types, such as *k-nearest neighbors* [ISS03] or *range* queries [GL04]. However, the streaming behavior of trajectories is all but implicit. For instance, in [MXA04], shared query execution is based on hashing with a predefined decomposition of space. Evaluation of spatiotemporal queries is outside the scope of this paper.

Finally, Stream Query Repository [SQR04] offers some example queries involving coordinates; nonetheless, spatial references are used as stationary information only, without examining potential interactions among the corresponding entities.

3 Modeling Trajectories as Data Streams

The scope of our model is currently restricted to point objects only, thus spatial entities are considered to have no extent: the shape of objects is reduced to their centroid. A *trajectory* is obtained by recording the successive positions a point object takes across time. This continuous data flow from a number of sources (e.g. GPS receivers in cars) to a central processing system can be suitably modeled as a *trajectory stream*. So trajectories are viewed from a historical perspective, not dealing at all with predictions about the future position of the moving objects. On the other hand, the model conveniently enables calculation of derived attributes, such as speed or acceleration.

3.1 Basic Entities

The entities appearing in the trajectory stream model can be drawn from the following principal components:

- *Relations*, which can be updated from time to time, thus allowing several temporal versions. Of particular interest are *spatial relations* that need be supported for stationary entities (e.g. area boundaries). These are stored in attributes of a primitive data type (point, line, and region). Each such relation must contain features of the same spatial data type, avoiding by all means any mixture of dissimilar spatial entities.
- Pure *data streams*, modeled as ordered multisets of tuples. The ordering attribute (timestamp) is based on valid time values registered at the data sources. An identifier needs to be included in the schema of attributes to distinguish the origin of the incoming tuples, as the union of those tuples creates the stream.
- *Trajectory streams* can be considered a specialized class of streams, which models the continuous movement of spatial point entities. The spatial reference evolves with time, following the successive positions taken by each moving object. Therefore, there is a close connection between spatial and temporal coordinates, which coexist simultaneously and create a unique spatiotemporal reference. This duality between space and time is very significant in formulating queries, since a combination of temporal and spatial predicates may be required.
- *Derived streams*, produced when operations and predicates are applied to the contents of *base streams* [ABW03] originating from data sources. Apart from local views used in continuous queries (as will be demonstrated in subsection 4.1), the principal form of derived information is *summaries* or *synopses* that approximate the contents of the (trajectory) stream, possibly by sampling or aggregating specific attributes. These can be produced after a query or a subquery is applied over the tuples of a base stream.
- *Query language*, for expressing continuous queries on the contents of streams and relations in a declarative manner. Obviously, this language can be implemented according to the algebraic structures prescribed for operations applicable to streams.

Informally speaking, if data evolves with time, then it is modeled as a data stream; if spatial information also varies with time, a trajectory stream is produced. Permanence in space and time is captured with relations. Observe that trajectory streams may collapse to data streams after projecting out the spatial reference attribute. So, it is possible that a data stream can be derived from a trajectory stream, but not vice versa. For example, a data stream containing the speed of moving objects can be calculated for every position, retaining only timestamp information (but not spatial features) in the resulting tuples. Consequently, by projecting out the timestamp

attribute as well, a relation is derived. Relations may even include temporal and/or spatial attributes; however, as long as order cannot be defined, they may be considered as degenerate (static) streams or trajectory streams.

As in the relational model, we may define:

Definition 1 (Schema of tuples): The *tuple schema* E of the data is represented by a set of elements (e_1, e_2, \dots, e_N) . Each element e_i is termed *attribute*, it has a name A_i and its values are drawn from a –possibly infinite– atomic data type domain D_i . The finite number N of the attributes is called the *arity* of the schema. A *tuple* is an instance of the schema and it is described by the values of the respective attributes. \square

3.2 Temporal Modeling

Time is represented as an ordered sequence of distinct moments. A *timestamp* is attached to the set of coordinates (and other attributes relative to the measurement) at data sources. As a result, each timestamp marks the actual time the item was recorded and refers to what is termed *valid time* in temporal databases. Another option is to introduce for each tuple a second timestamp based on *transaction time* (hence marking the time instants that tuples enter into the system for processing) [BBD+02]. Timestamps based on valid time need to be included in the schema of tuples (*explicit timestamps*), since they are indispensable to the calculation of derived attributes. More formally:

Definition 2 (Time Domain): The *Time Domain* T is regarded as an ordered, infinite set of discrete values (*time instants*) $\tau \in T$. A *time interval* $[\tau_1, \tau_2] \subset T$ consists of all distinct time instants of T between τ_1 and τ_2 . \square

For clarity [ABW03], T may be considered similar to the set of natural numbers N . For each timestamp, a *Data Stream* S is an unbounded multiset (bag) of items (i.e. allowing duplicates as in the definition at [ABW03]):

Definition 3 (Data Stream): A *Data Stream* S is a mapping $S: T \rightarrow 2^R$ from the Time Domain T to the powerset of the set R of tuples with schema E . One of the attributes A_τ is designated as the *timestamp* of each tuple, taking its ever-increasing values τ from T . \square

From a historical perspective, a Data Stream may be regarded as an *ordered sequence of tuple values evolving in time*, whereas an instance of the stream at a specific time (e.g., NOW or even in the future) is a finite set of tuples. Only one attribute is used as timestamp, i.e., a unique time reference for the entire tuple. As further explained in subsection 3.4, each tuple maps to exactly one timestamp, but more than one tuples can have identical timestamp values. Timestamps cannot be assigned a NULL value.

3.3 Spatial Modeling

As mentioned before, the proposed model deals only with *point entities* moving in two spatial dimensions. Despite

this simplification, interactions are allowed between moving objects and other shapes with extent (i.e. lines or regions), which remain stationary over time. Therefore, the model attempts to capture spatiotemporal relationships among trajectories or between trajectories and static spatial entities. It is also envisaged that the model is extensible to higher dimensionality of spatial entities.

Definition 4 (Point Domain): The *Spatial Point Domain* P contains all possible (hence infinite) pairs of values $\langle x, y \rangle$, with real planar coordinates $x, y \in \mathbb{R}$. Thus, P may be regarded similar to \mathbb{R}^2 . \square

Concerning spatial data types, those originally introduced in [GBE+00] are deemed sufficient. More specifically, `point`, `points`, `line` and `region` with their obvious interpretation may appear in relations, whereas only type `point` is allowed in trajectory streams.

We then make use of several primitive *predicates* for expressing topological relations. Each of these binary spatial predicates applies to a discrete instance of the trajectory of a moving point in conjunction either with another moving point or with a stationary spatial feature:

- a *point* (predicate `MEET` when the two points coincide),
- a *directed line* (so that predicates `LEFT` and `RIGHT` can be defined for the moving point, or `MEET` when the point is along the line), and
- a *region*, with predicates `INSIDE` for moving points that fall within its interior (U°) and `MEET` for those touching its boundary (∂U).

Complex predicates like `ENTER`, `LEAVE`, `CROSS` and `BYPASS` [PJT00] can be constructed from primitive ones.

Spatial *operators* are also needed for the formulation of queries. We designate four basic ones (note that the first three of them refer to a single object):

- `length` as the Euclidean distance between two arbitrary point positions of the same trajectory,
- `direction` as the angle between the vector of the object's movement and the horizontal axis,
- `heading` which denotes the general orientation of the movement ($\{E, NE, N, NW, W, SW, S, SE\}$), and
- `distance` for calculating the Euclidean distance between two distinct moving point objects at the same time instant.

Additionally, when formulating queries, simple shapes might be necessary to predicates, e.g. for defining the area of interest for a range query. We mention four basic *type constructors* for spatial entities:

- `Point(x, y)`,
- `Circle(P, r)`, where P is an instance of the `Point` type (the center) and $r \in \mathbb{R}$ the radius,
- `Rectangle(P1, P2)`, where $P1, P2$ are instances of `Point` (the edge-points of a diagonal) and
- `Triangle(P1, P2, P3)`, where $P1, P2, P3$ are the three vertices defining the triangle.

Notice that the *uncertainty* caused from positional inaccuracies should be taken into account (with a *fault tolerance*) when applying the predicates and operators mentioned above.

Finally, we extend the concept of data streams by defining a *Point Data Stream* S as an unbounded bag of elements with varying spatiotemporal properties:

Definition 5 (Point Data Stream): A *Point Data Stream* S is a mapping $S: T \times P \rightarrow 2^R$ from Time Domain T and Point Domain P to the powerset of the set R of tuples with schema E . In addition to an attribute A_τ used for *time-stamp values* $\tau \in T$, another attribute A_p acts as the *spatial reference* for the represented entity and obtains its values $p \in \mathbb{R}^2$ exclusively from the Point Domain P . Pair $\langle p, \tau \rangle$ may be regarded as a composite *space-time-stamp*. \square

3.4 Spatiotemporal Modeling

Movement in 3D-space (one temporal and two spatial dimensions) can be regarded as a sequence of discrete observations of positions across time. Once coherence in time domain is retained (i.e., no vacuums when recording locations), then contiguity in spatial domain for each object is also likely to be preserved. Redundancies in the collected data are inevitable, if sampling rate is too high and speed too low. On the other hand, it is not at all certain that an object's movement remains linear between two successive observations, as the model suggests. As a result, the trajectory itself should be considered as an error-prone approximation of the actual movement.

Definition 6 (Trajectory Stream): Let O denote the set of continuously moving point objects. The changing position of each distinct object $o \in O$ is actually a function of time $p_o: T \rightarrow P$. The Point Data Stream S composed of the successive tuple values obtained by monitoring the movement of the points from O is a *Trajectory Stream*. \square

Therefore, a trajectory stream for point objects may be viewed as an *ordered sequence of tuple values concurrently evolving in space and time*. An instance of this sequence is a set of tuples with positional and temporal indications; the former are simply some 2D-coordinates $\langle x, y \rangle$ in a common reference system, whereas the latter are instants drawn from a common Time Domain. Spatial and temporal references cannot be assigned a `NULL` value. This *space-time-stamp* uniquely determines the status of each object in space and time, and is explicitly included in the schema of tuples E . This is essential for calculations concerning derived attributes, such as speed. Updates to older data are not allowed, so trajectory streams are considered to be composed of *append-only* tuples. We emphasize the difference between point and trajectory streams through the following example.

Example 3.1. Imagine N mobile stations, which measure air pollution in a city. Collected values are transmitted to a control center along with their location and the time instant they were recorded. Obviously, incoming tuples

are compiled into a *trajectory stream*, as every item captures an instance of an object’s movement. It is likely that a user might submit a continuous query that retrieves tuples referring to the k out of N stations where the higher concentrations of a specific gas (e.g., CO) are observed at any time instant. As one might expect, these *top-k* stations will vary across time, so the selected tuples will form a data stream with changing spatial references for possibly different objects. Therefore, this derived information is a *point data stream*, not a trajectory stream; the notion of a sequence of object transitions from one location to another does not hold anymore in that case. \square

Although *total ordering* among tuples of a trajectory stream cannot be achieved based on their space-time-stamps, a *temporal total order* among tuples belonging to the same trajectory can be defined according to their timestamp values only. Obviously, elements referring to the same trajectory should have advancing timestamps, even if the object stands still (i.e. identical spatial positions, assuming no errors in measurement). Formally:

Definition 7 (Ordering): A *temporal ordering* f_O is defined as a many-to-one mapping from the *type domain* D_S of the tuples s of the trajectory stream S to the *Time Domain* T , with the following properties:

- $\forall s \in S, \exists \tau \in T$, such that $f_O(s) = \tau$.
- $\forall s_1, s_2 \in S$, instances of the trajectory stream S , and their values $\tau_1, \tau_2 \in T$ at timestamps $s_1.A_T$ and $s_2.A_T$ respectively, if $\tau_1 \leq \tau_2$, then $f_O(s_1) \leq f_O(s_2)$. \square

The first property of the definition above states that a timestamp need be associated to every tuple of the stream. Many tuples may map to the same timestamp; however, there may exist time instants where no tuples are recorded, e.g. for a particular time period no positional updates were received from a GPS. The second property establishes *monotonicity*, i.e. tuples of the trajectory stream are in increasing temporal order as time advances. Hence, tuples referring to the same trajectory can be ordered by simply comparing their time indications. This inherent and valuable characteristic of the time dimension has a subtle effect: after a while, older tuples may be considered as obsolete, so they may be either purged from memory completely freeing space or stored in synopses. Observe that the latter property holds for any two stream tuples (i.e. not necessarily belonging to the same trajectory), provided that time indications have identical granularities with values drawn from a common Time Domain T .

Note that an attribute of the schema may be designated as the identifier *id* of each point object. This can be proven useful in case the contents of the trajectory stream need to be split in disjoint *substreams*, one for each point object. We are currently investigating potential inclusion of velocity vectors in the model. Intuitively, it is the mobility of data sources in both space and time that produces a trajectory stream; this is best captured by a vector (like velocity or acceleration) that emphasizes change.

3.5 Types of Query Operators

As we will demonstrate in the following section, we make use of several types of operators in the framework of two stream prototypes, applying their semantics to trajectories. These constructs can either be adapted from the relational or data stream models, or may be introduced especially for managing trajectories (as we suggest in Section 5):

- *Relational*, such as selection or projection, modified properly to apply on trajectory streams as well.
- *Windowing*, for determining specific parts of the streams and converting them to relations for subsequent processing. The *scope* of the windows can be fixed, variable or sliding.
- *Transform*, for conveniently mapping relations (e.g. tuples produced by other operators) to streams.
- *Spatial*, for determining interactions between moving points and other stationary spatial entities, e.g. points, lines, regions.
- *Temporal*, for processing timestamps, which apply either on a time-point basis (individual timestamps) or for intervals (range of consecutive timestamps).
- *Spatiotemporal*, for effectively capturing evolving properties and interactions between moving points.

4 Querying Trajectories as Data Streams

In this section we briefly report our first experience in formulating continuous queries over trajectories in a SQL-like declarative language. Due to lack of space, we give just a few indicative queries submitted to the trial versions of two stream prototypes (STREAM, TelegraphCQ) that have recently been made publicly available. Despite variations at the idiom of SQL utilized, as well as certain limitations in expressing certain types of queries, our overall impression is that the streaming paradigm is powerful enough for managing moving objects. Our focus was on expressiveness, leaving aside query performance.

A synthetic dataset was used, containing positions of several types of vehicles (taxis, ambulances, private and police cars) moving in the road network of the greater area of Athens. After applying a shortest path algorithm at 1000 origin-destination pairs and sampling each route every second, in total 3.6 million records were created. Some adjustments were necessary to the schema of tuples, in order to properly represent spatiotemporal information, according to the restrictions imposed by each prototype.

4.1 Continuous Trajectory Queries in STREAM

We first give a short overview of the Continuous Query Language [ABW03] that is being developed for the STREAM prototype. Queries in CQL are composed from three classes of operators:

- *stream-to-relation* operators (particularly windows) are applied over streaming tuples and return relations.
- *relation-to-relation* are common SQL operators, such as selection or projection.

- *relation-to-stream* operators take a set of relational tuples as input and provide their output in the form of streams at every time instant. `ISTREAM` streams inserts to its input relation (i.e., tuples that did not exist in the previous time instant), `DSTREAM` streams deletes from its input relation (tuples that ceased to appear in the relation since the previous instant), whereas `RSTREAM` streams all current tuples of its input relation.

Thus, query semantics apply to streams and relations alike. Continuous queries have the following general form:

```
SELECT <select_list>
FROM <relations>, <streams_with_windows>
WHERE <predicates>
GROUP BY <expressions>
```

Windows are always specified for streams (in the `FROM` clause of the CQL query). Supported types include:

- *sliding windows* (`RANGE <time interval>`) that specify the most recent tuples according to their timestamps,
- *tuple-based* (`ROWS <value expression>`) that determine a given number from the most recent stream tuples and
- *partitioned windows* (`PARTITION BY <attribute list> ROWS <value expr>`), which partition the recent portion of a stream based on a subset of its attributes.

Since the current release of the `STREAM` prototype is Web-based, and our focus was not on performance issues, only a small subset of the data was used for convenience. The schema of tuples is `Vehicles (vID, vType, x, y, t, TS)`, where `TS` is used for timestamps attached to all tuples.

Query Q1: For instance, a spatiotemporal *range* query, such as “Find all taxis found within the area of interest (e.g., a rectangle) sometime in the last 10 minutes” can be expressed with a sliding temporal window, as follows:

```
SELECT V1.vID, V1.vType, V1.x, V1.y
FROM Vehicles V1 RANGE 10 MINUTES
WHERE V1.x>=475000 AND V1.x<=480000
AND V1.y>=4204000 AND V1.y<=4208000
AND V1.vType="TAXI" □
```

Note that CQL has no built-in types for any spatial entities, so only simple shapes can be defined (from their coordinates). Nested subqueries are not yet supported, but views may be defined over streams or relations. So, *aggregation* (`GROUP BY`) operations can be performed using views, since the `HAVING` clause is not available:

Query Q2: “Alert when more than 10 police cars are located simultaneously within the area of interest”. This query has to be expressed with two intermediate views:

```
InRegionCnt: SELECT TS, COUNT(vID) AS cnt
FROM Vehicles NOW
WHERE x>=475000 AND x<=480000
AND y>=4204000 AND y<=4208000
AND vType="POLICE"
GROUP BY TS
```

```
PoliceCnt: ISTREAM (SELECT * FROM InRegionCnt)
```

The first view returns a *relation* containing the number of police cars within the area for each time instant, whereas

the second one transforms (`ISTREAM`) these intermediate results into a *stream*. `NOW` is a shortcut for a sliding window that returns only tuples with the current timestamp value. The final query provides the expected results:

```
SELECT * FROM PoliceCnt NOW WHERE cnt>10 □
```

Even *nearest neighbor queries* can be expressed in CQL, but in a complex style with many intermediate views. Things get even more complicated, because only simple arithmetic functions are supported, so geometric ones, like distance, must be simulated in several steps. However, expressiveness of CQL helps in formulating queries for complex spatial predicates, such as `ENTER` into region:

Query Q3: “Find all vehicles entering now into the area of interest”. Spatial operation `ENTER` can be simulated as an `ANTISEMIJOIN`¹ between two temporary relational views. The former (`InsideAreaNow`) stores vehicles located now inside the area; the latter (`InsideAreaRec`) keeps those recorded within the area in the last two time instances, utilizing a partitioning window for each object:

```
InsideAreaNow: SELECT vID, t
FROM Vehicles NOW
WHERE x>=475000 AND x<=480000
AND y>=4204000 AND y<=4208000
```

```
InsideAreaRec: SELECT vID, t + 1 AS t1
FROM Vehicles PARTITION BY vID ROWS 2
WHERE x>=475000 AND x<=480000
AND y>=4204000 AND y<=4208000
```

```
SELECT InsideAreaNow.vID, InsideAreaNow.t
FROM InsideAreaNow ANTISEMIJOIN InsideAreaRec
```

Note that we take advantage of the monotonicity of time, assuming tacitly that positional information is registered at every time instant. □

4.2 Continuous Trajectory Queries in TelegraphCQ

TelegraphCQ is based on PostgreSQL DBMS, with many necessary adjustments to achieve adaptivity of operators in dynamically changing query load or data flow rate. Continuous queries are executed as cursors, which collect resulting tuples and fetch them to users in an incremental fashion. Only *sliding windows* are currently supported for streams; a `WINDOW` clause is specified for streams after standard `SELECT-FROM-WHERE` commands [TEL04]:

```
SELECT <select_list>
FROM <relation_and_<pstream_list>
WHERE <predicate>
GROUP BY <group_by_expressions>
WINDOW stream [time interval], ...
ORDER BY <order_by_expressions>
```

Neither nested subqueries nor self-joins can yet be expressed in TelegraphCQ. However, PostgreSQL offers several built-in *spatial* operators, functions and data types (point, polygon, etc.), which prove particularly valuable in formulating continuous queries over trajectories.

¹ *Antisemijoin* of R and S is defined as the multiset of tuples in R that do not agree with any tuple of S in the attributes common to both R and S .

The schema of tuples used for the test queries was `Vehicles (vID, vType, pos, t, TCQTIME)`, where `TCQTIME` is the attribute reserved for timestamp values. Next, we give some SQL expressions submitted to `TelegraphCQ`, equivalent to the queries of the previous subsection:

Query Q1: A temporal sliding window is specified (as in `STREAM`), but we take advantage of spatial data types and operations (`@` denotes “point `INSIDE` region”):

```
SELECT vID, vType, TCQTIME
FROM Vehicles V1
WHERE (V1.pos @ polygon '(475000, 4204000,
                          480000, 4208000)') = TRUE
AND V1.vType = 'TAXI'
WINDOW V1 ['10 minutes']
```

Query Q2: Aggregation is carried out on timestamp values, but the `HAVING` clause simplifies query expression:

```
SELECT TCQTIME, COUNT(V1.vID) AS cnt
FROM Vehicles V1
WHERE (V1.pos @ polygon '(475000, 4204000,
                          480000, 4208000)') = TRUE
AND V1.vType = 'TAXI'
GROUP BY TCQTIME
HAVING COUNT(V1.vID) >= 10
WINDOW V1 ['1 seconds']
```

However, due to lack of support for nested subqueries or views, other types of queries cannot be expressed at all. This is the case for nearest neighbors or complex spatial predicates (like `ENTER` into region, which can be expressed in `STREAM`, as showed for **Query Q3**). On the contrary, queries involving spatial functions, like *distance* (denoted `<->` in PostgreSQL) are formulated rather simply:

Query Q4: “Find all vehicles that are now within a distance of 500 meters from a point of interest” (this point is specified here by its coordinates):

```
SELECT vID, vType, TCQTIME, (V1.pos <->
                             Point '(475750,4201500)') AS distance
FROM Vehicles V1
WHERE (V1.pos <-> Point '(475750,4201500)') <= 500
WINDOW V1 ['1 seconds']
```

5 Special Structures for Trajectory Streams

Only *time-based*, *tuple-based*, and *partitioned windows* have been implemented for `STREAM` and `TelegraphCQ`. As we have just demonstrated, these structures are also valid for the trajectory stream model. In addition, we advocate for the adoption of four special-purpose window operators, with semantics applied to trajectories.

5.1 Window Specification Types

i. *Binary windows.* Sliding windows of size two time units may be needed to examine changes in motion, e.g., an object crossing the boundary of a stationary area. Window constructs proposed for data streams tend to ignore the *order* of tuples once the contents of the window have been specified. In contrast, when dealing with trajectories, the current and the oldest

tuple representing the boundaries of the temporal window are of particular interest. Based on this information, several interesting entities can be derived, such as velocity or acceleration. Further, *window edge functions* may be utilized in `SELECT` clauses, such as `First_Value` and `Last_Value` (used in SQL-99). These two functions can take as arguments specific attributes and return the values of the most remote and the most recent tuple within the window, respectively.

ii. *Landmark windows.* The starting edge of this window remains fixed over time, while its end point matches the present time instant. Hence, such a window returns an always-increasing number of tuples. Evidently, this window type can be defined based on time units only; therefore a syntax like `[AFTER t]` seems plausible and reminiscent of that for time-based windows. However, instead of a time interval, a specific time instant `t` is specified, no matter if any tuple exists with exactly this timestamp. A similar window type can be defined even when all tuples *before* a specific time instant `t` are needed, with a syntax like `[BEFORE t]`. Note that window contents remain unchanged as soon as current timestamp `NOW` exceeds value `t`. Interestingly enough, combination of these two language constructs could provide a *band window* that determines a time period with the syntax `[AFTER t1 AND BEFORE t2]` (if $t_1 \leq t_2$).

iii. *Area-based windows.* While any sliding or landmark window refers solely to timestamps, this particular window type applies only to trajectory streams and exploits their spatial contents. Intuitively, an area-based window extracts tuples from a trajectory stream whose positional reference falls inside the interior of the area defined. The area boundaries can be obtained either from a stationary spatial relation or by employing a type constructor (such as `Circle` or `Rectangle`). The former results in a stationary area, whereas the latter enables even a *moving area* to be defined, e.g. a circle of known (or time-varying) radius around a moving point object. A clause such as `[AREA A]` must be used *always* in combination with another window type based on timestamps (*sliding* or *landmark*); or else, it would select all tuples belonging to objects that crossed area `A` anytime in the past.

iv. *Trajectory-based windows.* Clearly, this is a value-based window that examines any but spatiotemporal attributes of a trajectory stream and extracts elements according to the criteria specified. Syntax `[WHERE <trajectory conditional expression>]` may be used so as to isolate qualifying tuples of certain trajectories, e.g. `WHERE id IN (24, 89, 425)`. A possible use of this window type might be to split a trajectory stream into separate *substreams* based on objects’ identities. Such a clause need be combined with time-based, partitioned or area-based (but not

tuple-based) windows, thus limiting the number of tuples returned. In contrast to the `WHERE` clause used in `SELECT` queries, this construct is applied to trajectory streams prior to any other processing for a given query (e.g. joins to another stream or relation).

5.2 Issues in Continuous Queries over Trajectories

We now informally point out certain issues related to query processing of continuous queries on moving objects. The notion of *punctuations* [TMSF02] could prove particularly constructive in query evaluation. For instance, extra tuples may be interleaved in a trajectory stream so as to indicate that a moving object has just crossed the boundary of an area or that it was found very close to another object or a known reference location.

The presence of numerous continuous queries over the same trajectory segments puts forward the need for *multiple query optimization* [Sel88]. In particular, it seems reasonable that expensive spatial operations (e.g. intersections) common to several queries need not be carried out in isolation from each other. Instead, grouping similar predicates or operators together, evaluating them and finally disseminating intermediate results to active queries, could cut execution costs considerably.

Last, but not least, *trajectory sketches* may be worth introducing. Due to the large volume of positional data flowing into the system, a compressed synopsis could provide an acceptable approximation for each trajectory (using sampling, wavelets or other summary structures).

6 Conclusions and Future Work

This paper considers streams as first-class concepts in a data management system and presents a new approach in modeling moving point objects by representing their continuous motion as trajectory streams. Basic predicates and operators are identified, under the assumption that movement takes place in one temporal and two spatial dimensions. Formulation of continuous queries over trajectories turns out to be feasible and intuitive, when window constructs are incorporated in a query language developed for managing data streams. Preliminary attempts to formulate continuous queries on moving objects in two data stream prototypes has showed encouraging results, and some challenging issues as well.

We believe that this approach is a promising area for future research. We plan to further study modeling of moving objects, introducing algebraic constructs for windows and proposing syntax rules for query language. Indexing trajectories by utilizing summaries as auxiliary structures in the presence of continuous positional updates is also important for improved query evaluation. Finally, shared execution and multiple query optimization of various spatiotemporal predicates is considered a major challenge in such a dynamic environment.

Acknowledgements. We greatly appreciate availability of data stream prototype systems, STREAM from Stanford Uni-

versity and TelegraphCQ from UC Berkeley. Comments by anonymous reviewers have improved the presentation of this paper.

References

- [ACC+03] D.J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a New Model and Architecture for Data Stream Management. *VLDB Journal*, 12(2):120-139, August 2003.
- [ABW03] A. Arasu, S. Babu, and J. Widom. An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations. In *Proceedings of the 9th Int'l Conference on Data Base Programming Languages*, Berlin, Germany, Sept. 2003.
- [AUR04] AURORA Project: Website available at <http://www.cs.brown.edu/research/aurora/>
- [BBD+02] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *Proceedings of the 21st ACM Symposium on Principles of Database Systems*, pp.1-16, Madison, Wisconsin, May 2002.
- [CCD+03] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S.R. Madden, V. Raman, F. Reiss, M.A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proceedings of the 1st Conference on Innovative Data Systems Research (CIDR'03)*, Asilomar, California, January 2003.
- [EGSV99] M. Erwig, R.H. Güting, M. 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.
- [GL04] B. Gedik, and L. Liu. MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System. In *Proceedings of the 9th Int'l Conference on Extending Database Technology*, Heraklion, Greece, March 2004.
- [GBE+00] 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*, 25 (1): 1-42, 2000.
- [ISS03] G. S. Iwerks, H. Samet, and K. Smith. Continuous k-Nearest Neighbor Queries for Continuously Moving Points with Updates. In *Proceedings of the 29th Int'l Conference on Very Large Data Bases*, Berlin, Germany, September 2003.
- [KSF+03] M. Koubarakis, T. Sellis, A. Frank, S. Grumbach, R.-H. Güting, C. Jensen, N. Lorentzos, E. Nardelli, B. Pernici, Y. Manolopoulos, B. Theodoulidis, N. Tryfona, H.-J. Schek, and M. Scholl (eds.) *Spatiotemporal Databases: The CHORONOS Approach*, LNCS vol. 2520, Springer, 2003.
- [MXA04] M.F. Mokbel, X. Xiong, W. G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In *Proceedings of 23rd ACM SIGMOD Int'l Conference on Management of Data*, Paris, June 2004.
- [PJT00] D. Pfoser, C. Jensen, and Y. Theodoridis. Novel Approaches in Query Processing for Moving Objects. In *Proceedings of the 26th Int'l Conference on Very Large Data Bases*, Cairo, Egypt, September 2000.
- [SWCD97] A. Prasad Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Proceedings of IEEE Int'l Conference on Data Engineering (ICDE'97)*, pp. 422-432, 1997.
- [Sel88] T. K. Sellis. Multiple-Query Optimization. *ACM Transactions on Database Systems*, 13 (1): 23-52, 1988.
- [SQR04] SQR – A Stream Query Repository. Available at <http://www-db.stanford.edu/stream/sqr>.
- [STR04] STREAM Project: Website available at <http://www-db.stanford.edu/stream/>
- [TEL04] TelegraphCQ Project: Website available at <http://telegraph.cs.berkeley.edu/telegraphcq/v0.2/>
- [TMSF02] P. Tucker, D. Maier, T. Sheard, and L. Fegaras. Enhancing Relational Operators for Querying over Punctuated Data Streams. In *Proceedings of the 28th Int'l Conference on Very Large Data Bases*, Hong Kong, China, 2002.