Semantics of Spatially-aware Windows over Streaming Moving Objects

Kostas Patroumpas Timos Sellis
School of Electrical and Computer Engineering
National Technical University of Athens, Hellas
{*kpatro, timos*}*@dbnet.ece.ntua.gr*

Abstract

Several window constructs are usually specified in continuous queries over data streams as a means of limiting the amount of data processed each time and thus providing real-time responses. Current research has mostly focused on tackling the temporal volatility of the stream, overlooking other inherent features of incoming items. In this paper, we argue that novel window types, other than strictly temporal, can also prove adequate in providing finite portions of multidimensional streams. We systematically examine the particular case of spatiotemporal streams generated from moving point objects and we introduce a comprehensive classification of window variants useful in expressing the most common operations, such as range or nearest-neighbor search. Our investigation also demonstrates that composite windows, combining temporal and spatial properties, can effectively capture the evolving characteristics of trajectories and assist significantly in query specification.

1 Introduction

Spatiotemporal streams generated from moving point objects are of particular importance in location-based services, fleet management, commodity tracking, etc., mostly for answering continuous range queries or nearest-neighbor search. The incessant arrival of fresh positional updates (occasionally at very high rates) and the necessity to process data online are the most important requirements to be addressed by a processing mechanism. Recent research has focused on efficiently maintaining query answers related to the current status of objects [10, 16], rather than *trajectories* composed from traces of their location over a period of time. Most of the processing techniques suggested thus far assume that each update received from a monitored object actually invalidates its previous position and triggers evaluation of the relevant *now-related* continuous queries.

Meanwhile, the notion of *windows* has been rather overlooked, despite its central role in prototype stream systems and processing algorithms [1, 3, 5]. Windows take advantage of semantics inherent in queries, as users are mainly interested in recent stream items rather than stale data. In terms of execution, windows get evaluated before any other operator (join, aggregation, etc.) in order to provide a bounded stream portion and thus enable respective queries to return timely and incremental results.

Likewise, we believe that adequate windowing constructs over streaming locations could prove a valuable tool for specifying and evaluating both *coordinate-based* and *trajectory-based* queries [13]. Indeed, window specification might act as a means of limiting the amount of data given for processing, by filtering out any irrelevant incoming locations (e.g., those beyond a city district of interest). On the other hand, windows would provide an ideal abstraction of the recent history of trajectories, as they could return extensive portions of their routes (e.g., objects moving within city center all along during the past 30 minutes) for detecting similarity, periodicity or other movement patterns.

Therefore, we introduce composite window types that in their specification integrate spatial conditions by means of a *coverage* function, in conjunction with the omnipresent temporal *scope*. Apart from the benefit of better expressiveness in query formulation, this offers prospects for reduced processing overhead. To the best of our knowledge, neither *spatially-aware windows* have ever been suggested so far nor any generalized notion of windowed operators in multiple dimensions (other than the temporal one).

In this paper, we extend our earlier viewpoint concerning windows in stream processing [12], by providing a concise algebraic representation of spatiotemporal streams. We also describe precise semantics of certain novel space-based window variants useful to efficiently manage the current or past locations of streaming positional updates. We demonstrate the expressiveness of such constructs through their use in SQL-like renditions of several representative queries. We further examine windows with respect to management of streaming trajectories and exemplify their use in continuous trajectory joins as a convenient means of discovering similarities in objects' movement.

The remainder of this paper is organized as follows: Section 2 explains the motivation behind our work. Section 3 describes basic notions on streams generated from moving point objects. Section 4 introduces semantics of spatially-aware windows and presents most typical variants, whereas Section 5 discusses application of windows over streaming trajectories. Related work is surveyed in Section 6. Finally, Section 7 offers conclusions and future research directions.

2 Rationale

Windows based solely on temporal constraints can certainly be applied over streaming locations, e.g., a sliding window extracting items of last hour [11]. Yet, inherent spatial characteristics would remain largely unexploited, despite the fact that most requests usually focus on restricted areas of interest. This seems inevitable for massive datasets, since applications cannot keep track of object interrelationships for long. Even worse, evaluation of spatial predicates would be deferred to subsequent stages in execution plans, causing considerable delay to timely responses. By introducing spatially-aware windows, it is possible to bound incoming locations according to joint spatial and temporal criteria. Such constraints get applied early in execution plans, offering also great opportunities for multiple query optimization (e.g., handling together overlapping regions).

For motivation, suppose a monitoring application used for traffic surveillance and driver notification in a city that is separated in certain administrative districts. The following indicative examples illustrate the potential benefits of spatially-aware windows to query specification:

Q1. For each vehicle now travelling within the city center, identify every 10 seconds its nearest one that also moves inside this designated region.

Q2. Report every minute the number of cars of each type located in the city center anytime during past quarter.

Q3. Notify a privileged customer for special offers and discounts whenever located among the 50 users currently closest to any service site (e.g. fueling stations).

Q4. Monitor average speed per vehicle type based on the 1000 most recent locations recorded in each district.

Q5. Identify delivery trucks that follow similar routes during past half-hour. In case of a car failure, ask another driver nearby to take charge of pending customers.

Q6. Detect pairs of vehicles having close enough locations (even not concurrently recorded) in the last 10 minutes.

The common characteristic of these requests is that a *spatial constraint* (points or zone of interest) should be applied continuously over the dynamic locations, effectively reducing the amount of data that needs to be handled each time. Furthermore, continuity in time and contiguity in space are preserved in every instance of the extracted locations, as well as the sequential pattern in trajectory subtraces. In contrast, notice how query Q1 would give semantically different results in case the search for nearest neighbors was performed before window (range) filtering.

When applying a spatially-enabled window over a stream of positional updates, resulting tuples create compact subsequences of the original routes in both temporal and spatial domain, because window evaluation takes precedence over all other operators. Besides, constraints in window specifications are not necessarily fixed (as in typical selection predicates) but may vary over time and space, keeping in pace with the evolving nature of positional streams. This latter feature offers a powerful means to express a wide range of continuous spatial queries, as we explain later on.

3 Handling Positional Streams

3.1 Fundamentals on Streaming Locations

In general, moving objects have spatial extent (shape), which might be varying over time. Our principal concern is to capture movement characteristics of objects, rather than other changes (appearance, disappearance etc.) occurring during their lifetime. So, we restrict our interest to streaming point entities over the 2-dimensional Euclidean plane:

[Point Domain]: *Planar Point Domain* $\mathbb{P}$ contains all possible (hence infinite) pairs of values $\langle x, y \rangle$, where $x, y \in \mathbb{R}$ are interpreted as point coordinates on 2-d Euclidean plane.

For any moving object, its locations are usually recorded at specific time instants. Hence, each recorded location is accompanied with a *timestamp* value drawn from a domain common for all monitored objects:

Time Domain $\mathbb{T}$ is an ordered, infinite set of discrete time instants $\tau \in \mathbb{T}$.

Note that successive positions of an individual object are assigned ever-increasing timestamp values that denote the registration time of each recording or its sequence number. Thus, a unique *ordering* reference is established among all such elements. In fact, $\mathbb{T}$ may be considered similar to the domain of natural numbers $\mathbb{N}$, whereas $\mathbb{P}$ is similar to $\mathbb{R}^2$.

Now consider a "cloud" of point locations recorded over $\mathbb{P}$ at a specific instant $\tau \in \mathbb{T}$. This dataset can be generated after taking a snapshot of the current positions for numerous moving objects, e.g., GPS-equipped vehicles on a road network. In case such information is being continuously monitored over a long period of time, it takes the form of a spatiotemporal data stream consisting of append-only tuples (i.e., no deletions allowed) from positional updates:

[Point Data Stream]: A *Point Data Stream* S is a mapping $S : \mathbb{T} \times \mathbb{P} \rightarrow 2^R$ from Time Domain $\mathbb{T}$ and Point Domain $\mathbb{P}$ to the powerset of the set R of tuples with common schema E of attributes. In addition to an attribute A_τ denoting timestamp values $\tau \in \mathbb{T}$, another attribute A_p acts as the spatial reference for each point entity and obtains its values $p \in \mathbb{R}^2$ exclusively from $\mathbb{P}$. Pair $\langle p, \tau \rangle$ is considered the composite *space-time-stamp* for a stream element $s \in S$.

As is typical in data stream modelling, we assume that a possibly large, but always finite number of positional updates arrive for processing at each timestamp $\tau_i \in \mathbb{T}$. This bounded multiset actually constitutes the *current stream instance* $S_I(\tau_i) = \{s \in S : s.A_\tau = \tau_i\}$. Besides, *current stream contents* $S(\tau_i) = \{s \in S : s.A_\tau \leq \tau_i\}$ signify an ordered sequence of all data elements accumulated thus far, practically representing the historical movement of objects.

3.2 Specifying Queries with Coverages

Most continuous queries over streaming point locations refer to *range search* ("identify objects contained in a specific region"), *k-nearest neighbor* ("find k objects currently closest to particular locations"), *aggregates* (e.g., "calculate count or density of objects in designated areas") etc. Such *now-related* queries usually specify a spatial predicate (by means of a polygon, a set of query points, etc.) that current positions should meet in order to qualify for the answer.

At an abstract level, we may consider the spatial entity involved in this search as the respective *coverage* of the query, which implies its presence over the Euclidean plane and varies according to the degree of its geometry (point, line or region). This approach can simplify reasoning about continuous queries, since a coverage essentially conveys the notion of the area (or points) of interest that appears in user requests. In other words, we implicitly set a "mask" over the streaming locations received, hence practically restricting the set of features that will subsequently be considered in query evaluation. Next, we distinguish four typical cases:

i) Spot coverages. A set of distinct points (or multipoints) on the plane occupies certain locations, which may be collectively regarded as a *spot coverage* (shown as encircled spots in Fig. 1a). Intuitively, this construct specifies a group of points of interest where certain phenomena should be observed, e.g., appearance or disappearance of monitored objects at particular crossroads, mine fields, wells, etc.

More formally, let a finite set of k distinct points $P_k = \{\langle x_i, y_i \rangle \in \mathbb{P}, i \in [1..k]\}$. These locations define a subset of the plane $\mathbb{P}$ according to the following coverage function:

$$cover_{\text{SPOT}}(P_k) = \{\langle x, y \rangle \in \mathbb{P} : \langle x, y \rangle \in P_k\}$$

ii) Surface coverages. The interior of a region (polygon, rectangle, circle etc.) naturally defines a *surface coverage* that is typical for range queries. Conversely, the exterior of such a region also defines a coverage, taken as the complement of its interior with regard to the entire plane $\mathbb{P}$. Next, we exemplify the case for two simple shapes (Fig. 1b):

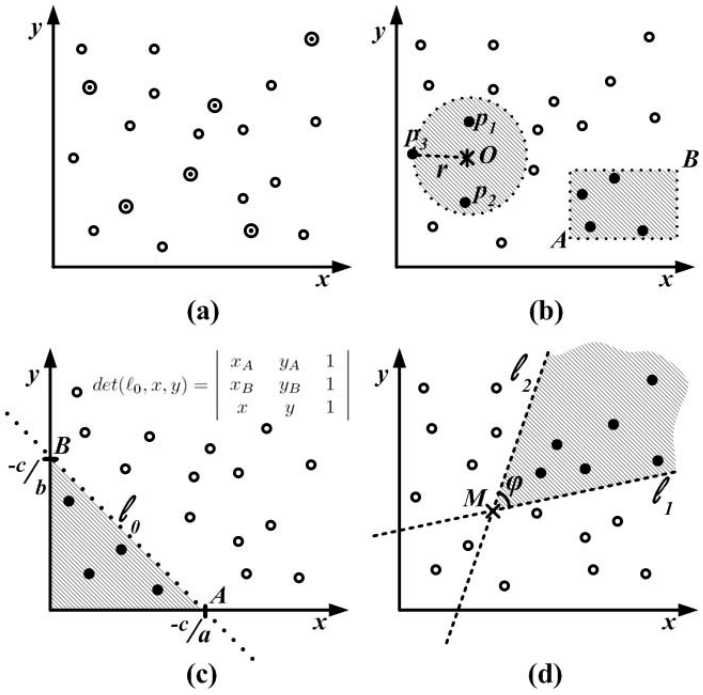

Figure 1. (a) A spot coverage. (b) Typical surface coverages. (c) A border coverage on the left of a straight line. (d) Band coverage formed by two straight lines.

The surface covered by a rectangle L with bottom left $A(x_A, y_A)$ and top right corner $B(x_B, y_B)$ is trivial:

$$cover_R(L) = \{\langle x, y \rangle \in \mathbb{P} : x_A \leq x \leq x_B \wedge y_A \leq y \leq y_B\}$$

Similarly, the coverage of a circle $C(O, r)$ on $\mathbb{P}$ with center $O(x_o, y_o) \in \mathbb{P}$ and radius $r \in \mathbb{R}$ is defined as:

$$cover_C(O, r) = \{\langle x, y \rangle \in \mathbb{P} : (x - x_o)^2 + (y - y_o)^2 \leq r^2\}$$

For other common shapes, like triangles, ellipses etc., the surface coverage also has a closed formula, but for irregular polygons this function gets more complex. For a convex polygon, its coverage can be stated as a conjunction of constraints $ax + by + c \leq 0$, i.e., an intersection of half-planes in 2-d plane. The coverage for a non-convex polygon (or disjoint regions) can be expressed as a union of convex polygons, i.e. disjunction of conjunctive spatial constraints.

iii) Border coverages. Suppose that an arbitrary direction is assigned to a line curve over $\mathbb{P}$. Then, $\mathbb{P}$ is separated in two disjoint half-planes that represent the left- and right-side coverages, while the *border coverage* consists of points along that curve. This construct may be useful in identifying

topological relationships, e.g., objects moving across a river or crossing the border between two states.

In case of a straight line $\ell_0 : ax + by + c = 0$ on plane $\mathbb{P}$, we may assert a direction by choosing two distinct points A, B along the line, e.g., its intersections with the axes (Fig. 1c). For any point $\langle x, y\rangle \in \mathbb{P}$, we can easily decide its position relative to ℓ_0 by the sign of determinant $det(\ell_0, x, y)$. For instance, the coverage on the left of line ℓ_0 is:

$$cover_{\mathsf{LEFT}}(\ell_0) = \{\langle x, y\rangle \in \mathbb{P} : det(\ell_0, x, y) < 0\}$$

Curves of arbitrary shape can be approximated by complex functions (e.g., splines) or consecutive line segments.

iv) Band coverages. Any such non-closed coverage defines a zone of interest that is partially bounded by at least two curves. Considering the simple case of two straight lines ℓ_1 and ℓ_2 that intersect at point M, their band coverage is anchored at M and corresponds to the *positive convex angle* φ (Fig. 1d). If ℓ_1 and ℓ_2 are parallel, the band is trivially the fixed-width zone between them.

4 Spatial Windows over Streaming Locations

As already mentioned, window semantics in continuous queries over infinite streams help in determining a countable portion of tuples by setting specific temporal constraints. Towards this goal, windows rely on an ordering established among all incoming items, through a *windowing attribute* [8] that usually contains timestamp values. In practice, at each time instant, window's contents (its current *state*) make up a temporary relation of a finite, yet ever-changing set of tuples that fall within its *scope* (e.g., past hour, latest 100 items etc.). Window variants are often classified according to their evolution with time (*sliding*, *tumbling*, *landmark* windows) or through a presumptive succession in arriving items (*count-based* or *partitioned*) [12].

When it comes to streaming point locations, the assertions of the data stream model still hold and typical coordinate-based queries can be expressed [11]. Topological queries are more difficult to express, as they include spatial or spatiotemporal notions (e.g., orientation). Moreover, defining total order in space is not an easy task, especially for non-stationary entities. Our key idea is to exploit total order in time domain, so as to establish an ordering of positional updates and further create sequences of items for each distinct object (its *trajectory*). In addition, spatial criteria may act as enhancements to typical window constructs, in a manner similar to *value-based windows* [3]. The crucial difference is that spatial constraints are not just some extra filtering conditions, but they control window's structure by modifying its extent or shape across time.

We distinguish spatially-aware windows in two main classes. Next, we refer to windows against streaming point objects, i.e. distinct locations considered in isolation from each other. In Section 5, we discuss windows over trajectories by exploiting continuity in each object's movement.

4.1 Abstract Semantics and Properties

In effect, the coverage function is the spatial analog of *scope* that determines the extent of time-based windows, so:

[Spatial Window]: A *spatial window* W_C is specified by a subset C of the plane $\mathbb{P}$ and returns all location items from a Point Stream S that qualify to the coverage of C, as follows:

$$W_C(S) = \{s \in S :\ s.A_p \in cover(C)\}$$

provided that $\mid W_C(S) \mid \leq n$, for any large, but finite $n \in \mathbb{N}$.

This definition resembles to the general description of windows over streams [12], by simply establishing a spatial containment predicate instead of a generic constraint. At this abstract level, no temporal specifications are taken into consideration, assuming that streaming locations are being processed as they come, ignoring any resource limitations for maintaining the growing volume of data items.

By combining spatial and temporal properties of the stream, we may suitably specify composite windows over spatiotemporal streams. As shown in Section 3, the exact shape of a coverage is controlled by the characteristics of its owner entity, and it is expressed as a geometric function. However, a coverage can be *time-varying*, if it is allowed to evolve over time, e.g., modify its location or shape. We distinguish three criteria for detecting coverage changes:

- *Mutability.* Depending on whether a coverage changes location over time, we may characterize it either *stationary* (i.e., immobile) or *moving*.
- *Spatial interaction.* If any two successive states of a coverage intersect, we may speak of *overlapping* coverages. A coverage is *contiguous*, when the boundary of each state touches that of its predecessor. Otherwise, the coverage moves in a *discontinuous* fashion.
- *Variability* is concerned with changes in the shape of a coverage (mostly for areas). If that shape remains intact all along its lifetime, then the coverage is *permanent*. If not, the coverage is *distorted* and takes irregular shapes, e.g., it shrinks or grows over time.

4.2 Typical Spatially-aware Windows

In the following, we present several intuitive variants of spatially-enabled windows that can greatly support query specification in continuous monitoring of moving objects.

4.2.1 Extent-based windows

This most characteristic spatially-aware window simply filters out locations beyond the query area of interest (Fig. 1),

like *range predicates* in spatial databases. Depending on the type of coverage C specified, this construct returns all tuples from a particular portion of the point stream, which fall inside the extent or coincide with spot locations of the coverage. The extent is fixed at each $\tau \in \mathbb{T}$, although C may be stationary (e.g., extent of the central town district) or moving (e.g., 1 km around a moving truck). Formally:

$$W_R(S, \tau, C) = \{\, s \in S(\tau) : s.A_p \in cover(C) \\ \wedge\, s.A_\tau \in scope(\tau)\}$$

All tuples returned by each window state retain their original timestamp values, often an indication $\tau = \texttt{NOW}$ if the current stream instance $S_I(\tau)$ is given as input (i.e., $scope(\tau) = \texttt{NOW}$). In case the query is concerned with locations recorded over a time interval (e.g., a sliding window over the past 20 minutes, a landmark window after 10 a.m. etc.), an adequately parameterized *scope function* can be used to fetch the relevant tuples [12]. Therefore, a composite window variant is formed, effectively combining spatial and temporal constraints over streaming locations.

An interesting variation concerns extent-based windows with a fixed-shape coverage that *shifts* as a compact entity over the 2-d plane by (dx, dy) during time interval dt (expressed in timestamp units). Since step dt remains constant, the window's coverage will move periodically. In terms of query specification, we propose a generic SQL clause like:

[EXTENT $\langle cover \rangle$ RANGE $\langle time \rangle$ SHIFT BY(dx, dy, dt)]

We borrow from CQL [2], keyword RANGE $\langle time \rangle$ to denote the temporal scope of the window. For now-related queries, this can be simplified using shortcut NOW.

Query Q1. Assume that a spatial operator for k-NN search is available (e.g., implemented as proposed in [10]), whereas the polygon representing city center is denoted with parameter center_geom. Then, query Q1 (Section 2) can be expressed in SQL-like syntax as a windowed self-join between pairs of successive window states:

```
SELECT V1.id, V2.id
FROM Vehicles[EXTENT center_geom NOW
              SHIFT BY (0,0,10 SECONDS)] V1,
     Vehicles[EXTENT center_geom NOW
              SHIFT BY (0,0,10 SECONDS)] V2
WHERE NN(V1.location, V2.location, 'k=2')
AND V1.id < V2.id
```

To avoid situations where each object reports itself, we specify k=2 and then eliminate duplicates comparing object id's. From a semantics point of view, it is crucial that extraction of relevant locations (i.e., coverage evaluation) actually precedes execution of k-NN operator. This guarantees that k-NN search is performed against the correct dataset, so qualifying neighbors are moving inside city center for sure.

Query Q2. This aggregation requires a time-based sliding window of 15 minutes that gets refreshed every minute, so:

```
SELECT V.type, COUNT(DISTINCT V.id)
FROM Vehicles [EXTENT center_geom
               RANGE 15 MINUTES
               SHIFT BY (0,0, 1 MINUTE)] V
GROUP BY V.type
```

4.2.2 k-Proximity windows

This window type extracts for every designated point of interest its k closest location items (Fig. 2a), which proves advantageous for situations like that of query Q3 (Section 2). Its computation first involves a k-NN search over the current positional updates in $S_I(\tau)$ for each such *focal point* $q_f \in F \subset \mathbb{P}$. Then, the union of these partial subsets yields the final multiset of location tuples. In algebraic notation:

$$W_P(S, \tau, F, k) = \bigcup_{q_f \in F} \{s \in S_I(\tau) : \\ \exists\, r \in \mathbb{R},\, ((\, r \geq 0 \,\wedge\, s.A_p \in cover_C(q_f, r) \wedge \\ \wedge\, |\, \{s' \in S_I(\tau) :\, s'.A_p \in cover_C(q_f, r)\}\, | \leq k) \\ \wedge\, \forall\, r' \in \mathbb{R},\, (\, r' \geq r \,\wedge \\ \wedge\, |\, \{s'' \in S_I(\tau) :\, s''.A_p \in cover_C(q_f, r')\}\, | > k))\}$$

Semantically speaking, this window is applied over the current stream locations (all referring to $\tau = \texttt{NOW}$), so the chosen nearest neighbors are *synchronized*. Apparently, not only the identities of k-proximal locations vary over time, but also the distances of k-th (remotest) neighbors from their corresponding focal point[1]. Thus, search radius r that determines the circular coverage around a focal point q_f gets adjusted at each τ, depending on current object positions. Note that focal points may not necessarily be stationary, but they could be allowed to move from time to time.

In terms of continuous evaluation, an efficient k-NN algorithm [10, 16] is more adequate to identify proximal locations compared to the naïve policy implied by the semantic definition, i.e., successive circle enlargements until k-th point is reached. With respect to SQL, a clause such as

[NEARBY $\langle spot\_cover \rangle$ ROWS $\langle tuple\_count \rangle$]

could declare a proximity window that repeatedly returns the union of all $\langle tuple\_count \rangle$ current locations nearest to each member of a $\langle spot\_cover \rangle$, as exemplified next.

Query Q3. Assuming that customer classification is available in the stream (otherwise, it could come from a join with a table where this data has been stored), the query syntax is:

```
SELECT V.id
FROM Vehicles [NEARBY (SELECT loc FROM Sites)
               ROWS 50] V
WHERE V.class = 'PRIVILEGED'
```

[1] It may occur that multiple locations qualify as k-th neighbor of focal point q_f, e.g., A, B are equi-distanced from q_3 in Fig. 2a. Like count-based windows in CQL [2], ties may be broken in a non-deterministic fashion, by choosing arbitrarily one of the candidate locations.

The nested subquery after keyword NEARBY serves in determining the list of focal points (sites). Next, a 50-NN search is being performed for each site, providing the current window state over vehicle locations. The selection condition against this temporary dataset returns qualifying customers.

4.2.3 Distance-based windows

In contrast to k-proximity windows, this variant returns locations falling within a specific distance d around any point of interest q_f (taken from a spot coverage F). The number of qualifying locations inside each such "influence" area may be varying over time, due to changing object positions. Additionally, a temporal scope may be specified to retrieve positions recorded during a sliding time interval. Formally:

$$W_P(S,\tau,F,d) = \bigcup_{q_f \in F}\{s \in S(\tau) : s.A_p \in cover_C(q_f,d) \wedge s.A_\tau \in scope(\tau)\}$$

and the respective SQL expression is quite straightforward:

[NEARBY ⟨*spot_cover*⟩ WITHIN ⟨*distance*⟩ RANGE ⟨*time*⟩]

Although this variant looks similar to extent-based windows, a subtle difference exists. Distance-based constructs are intrinsically tied to designated sites that may not be stationary, but instead move at arbitrary directions; hence, they are not shifting as one entity like an explicit compact extent. We see a research challenge in continuous maintenance of such window states with respect to moving circles.

Query Q3′. Identify privileged customers within 1 km distance from service sites anytime during past 5 minutes:

```
SELECT V.id
FROM Vehicles [NEARBY (SELECT loc FROM Sites)
               WITHIN 1000 METERS
               RANGE 5 MINUTES] V
WHERE V.class = 'PRIVILEGED'
```

4.2.4 Tessellated windows

Suppose a *spatial decomposition* D of plane $\mathbb{P}$ into a set of m pairwise disjoint regions $\{d_i, i \in [1..m]\}$, such that:

$$\forall\, i,j \in [1..m]\,,\ d_i \neq \emptyset \ \wedge\ d_i \cap d_j = \emptyset \ \wedge\ \bigcup d_i = \mathbb{P}.$$

There is no restriction that these regions must necessarily be of equal area (i.e., a grid-like partitioning), but they can form an irregular planar *tessellation* (Fig. 2b). Then, each incoming point location falls inside exactly one such region d_k and it is assigned to it. This is essential for applications that have no interest in the exact position of objects, but rather in their distribution over predetermined areas of interest (for instance, city districts, traffic flow zones, restricted-access areas etc.). The returned subset can be used to evaluate density queries or to maintain the N most recent stream locations for each subarea d_k. Each qualifying item retains its original timestamp, so this variant is inherently sliding in time and reminiscent of *partitioned windows* [2], only that here subdivision refers to space. Specifically:

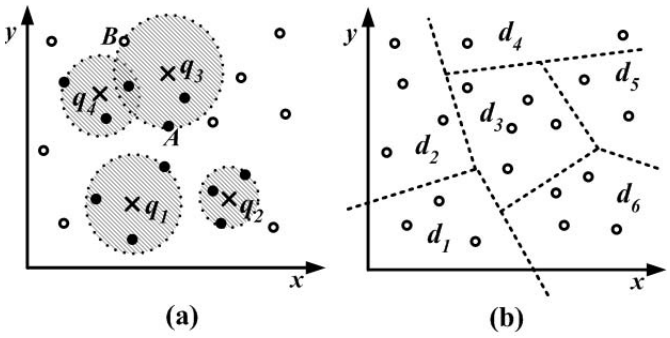

Figure 2. (a) A 3-proximity window for 4 focal points. (b) Irregular tessellation of the plane.

$$W_T(S,\tau,D,N) = \{\langle s,d_i\rangle : \exists\, d_i \in D, s \in S(\tau) \wedge s.A_p \in d_i \wedge \exists\, \tau_1 \in \mathbb{T}\,(\tau_1 \leq \tau \wedge |\, \{\langle s,d_i\rangle :\ \tau_1 \leq s.A_\tau \leq \tau\} \,| \leq N) \wedge \forall\, \tau_2 \in \mathbb{T}\,(\tau_2 < \tau_1 \wedge |\, \{\langle s,d_i\rangle : \tau_2 \leq s.A_\tau \leq \tau\} \,| > N)\}$$

For expressing such windows in queries, an SQL clause

[TESSELLATE BY ⟨*cover*⟩ ROWS ⟨*tuple_count*⟩]

is proposed, where ⟨*cover*⟩ denotes either an attribute or a list containing the regions of the tessellation.

Query Q4. Let a relational table Districts where the geometric subdivision of the city in administrative zones is stored in field region. Then, query Q4 may be written as:

```
SELECT V.type, AVG(V.speed)
FROM Vehicles [TESSELLATE BY (SELECT region
                              FROM Districts)
               ROWS 1000] V
GROUP BY V.type
```

assuming that each vehicle also relays its current speed to the server. Note that aggregation is computed collectively for the unified subsets supplied by the current window state.

4.3 Discussion

Although this collection of window variants is not exhaustive, it addresses frequent user queries, while expressing range and k-NN search is straightforward. What we advocate is pushing down spatial constraints into window filtering, so that evaluation of spatial and temporal constraints could be interchanged. Here we assumed a trivial processing policy that requires reevaluation of window states at every distinct $\tau \in \mathbb{T}$, but more efficient techniques are possible (e.g., buffer "safe" zones [15]). Continuous query optimization involving spatial windows is beyond the scope of this paper and is the topic of our ongoing work.

5 Windowed Joins over Trajectory Streams

Up to this point, the case of spatiotemporal streams of moving points has been examined for continuous queries that collectively involve discrete isolated snapshots of objects' locations. However, the sequential nature in each object's path is significant when we need to inspect movement patterns and interactions between objects (or with stationary entities) over a period of time. More specifically:

Trajectory T of a point object identified as oid and moving over plane $\mathbb{P}$ is a possibly unbounded sequence of its timestamped positions across time, i.e., triplets of values $\langle oid, p_i, \tau_i \rangle$, where position $p_i \in \mathbb{P}$ and timestamp $\tau_i \in \mathbb{T}$.

We further consider a *trajectory stream* of point objects as an ordered sequence of locations concurrently evolving in space and time. It is worth mentioning two characteristics inherent in streaming trajectories. First, *time monotonicity dictates a strict ordering of spatial positions* taken by a moving object all along its trajectory. Second, *locality in an object's movement should be expected*, assuming that its next location will be recorded somewhere close to the current one. Therefore, it is plausible to anticipate consistent object paths in space, save for possible discontinuities in movement due to communication failures or noise.

The general notion of *trajectory join* queries was introduced in [4] as a means of identifying all pairs of trajectories between two datasets that exhibit some kind of user-defined similarity. In spatial databases, *distance-join* queries are often used to identify trajectories that are found within a certain distance from each other during a time interval, e.g., "vehicles following similar routes after 8 a.m.".

Next, we introduce the novel idea of *continuous joins* between streaming trajectories, which effectively reduce to a series of snapshot trajectory joins. More concretely, let two sets R, S consisting of $n, m \in \mathbb{N}$ objects moving on plane $\mathbb{P}$, their respective trajectory sets $\{T^R\}, \{T^S\}$, a possibly time-varying distance threshold $\epsilon(t) \in \mathbb{R}$ for each $t \in \mathbb{T}$, and a non-standard temporal extent $\Delta t \in \mathbb{T}$. Then:

Continuous trajectory join $Q_c(\{T^R\}, \{T^S\}, \epsilon(t), \Delta t, \tau_0)$ at any $\tau_0 \in \mathbb{T}$ returns all pairs of trajectories $\langle T_i^R, T_j^S \rangle, i \in [1..n]$, $j \in [1..m]$, so that $distance(T_i^R(t), T_j^S(t)) \leq \epsilon(t)$ at every time instant $t \in (\tau_0 - \Delta t, \tau_0]$.

For example, in query Q5 a temporal (half-hour) and a spatial frame (similarity expressed as a distance threshold of 100 meters) can be jointly applied, perhaps varying with time. The temporal scope of the window decides to what extent we will consider the historical trace of each trajectory; orthogonally, the spatial coverage determines how far away from each path we will search for potential matches. The special case of *trajectory self-joins* (i.e., sets R, S are identical) can also handle *nearest-neighbor search* on trajectories. In Fig. 3, snapshots of a windowed join query are illustrated as 3-d volumes that interweave time with space. A "twisted" cylindrical envelope of variable radius $\epsilon(t)$ and adjustable height Δt is constructed for each trajectory T_q. A snapshot of this envelope returns trajectories T_i, T_j and T_k that have been moving within distance $\epsilon(t)$ from T_q at every instant t during recent time period Δt. Results at each time instant include only those trajectories completely within the current shape of this cylindroid frame. For instance, T_i is partially outside the frame during Δt_4, so it does not qualify at t_4, while it is included in the answer at t_5.

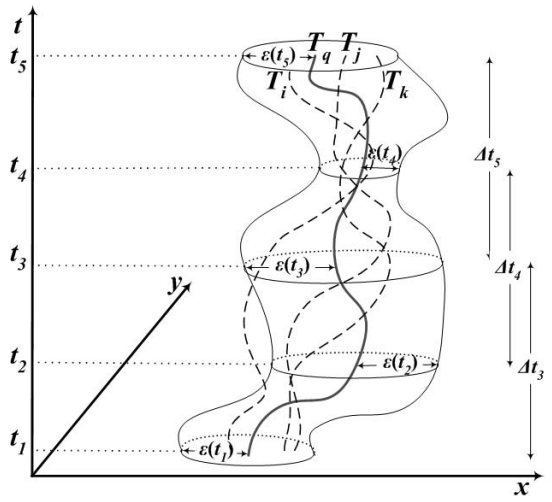

Figure 3. Identifying objects of similar movement pattern with windowed trajectory joins.

We can also relax the requirement of temporal concurrence between matching trajectory segments, to handle cases like query Q6. Such a join finds objects passing from similar places at different times within a sliding interval:

Slacked trajectory join $Q_{sl}(\{T^R\}, \{T^S\}, \epsilon(t), \Delta t, \tau_0)$ at $\tau_0 \in \mathbb{T}$ returns pairs of trajectories $\langle T_i^R, T_j^S \rangle, i \in [1..n]$, $j \in [1..m]$, such that $\exists\, t_i, t_j \in (\tau_0 - \Delta t, \tau_0]$ which have $\mid t_i - t_j \mid < \Delta t$ and $distance(T_i^R(t_i), T_j^S(t_j)) \leq \epsilon(\tau_0)$.

Note that this operation is substantially different from time-relaxed trajectory joins [7], since the latter applies on static datasets and lacks continuous semantics.

With respect to trajectory join evaluation, the stream of locations can be seen as partitioned into several substreams, one for each individual object. Each trajectory is handled separately and a custom window instantiation is applied over its trace to provide the relevant segments. Temporal duration is common for all trajectories (e.g., the past half-hour), so the returned segments are *synchronized*. At each time instant, the respective spatial coverages have equivalent shapes (e.g., circles of equal area around each object).

6 Related Work

There has been a recent surge in research on real-time management of moving objects for monitoring applications. It is the current position of objects that matters most in typical range [9] and nearest-neighbor search [10, 15, 16]. Trajectories have been studied in spatial databases with respect to indexing [13], setting aside their streaming nature. Latest studies examine interaction between trajectories, focusing on joins [4, 7] and nearest-neighbor queries [6].

Several proposals have been presented for a stream query language, mostly leveraging familiar SQL syntax and semantics [14]. In Aurora [1], the main interest is in computing joins and aggregates through sliding and tumbling windows. A declarative Continuous Query Language (CQL) has been developed for STREAM [2], which supports both streams and relations and allows specification of tuple-based and time-based sliding windows. StreaQuel is a SQL-like query language for TelegraphCQ system [5], currently supporting time-based sliding windows only, but plans for more variants (landmark, tumbling) are under way.

Our own investigation [11] has verified that certain now-related continuous spatial queries, such as range search or distance joins, can be answered from stream prototypes like STREAM or TelegraphCQ, by using typical (mostly sliding) window variants. However, these general-purpose stream processing engines offer limited expressiveness for topological queries concerning trajectories, as they cannot maintain the sequential nature of movement. To the best of our knowledge, no suggestion has been made so far for combining spatial constraints into temporal stream semantics, so as to further enhance window specifications.

7 Conclusions and Future Work

In this paper, we introduced spatially-aware windows over streaming locations generated by moving objects. We proposed novel window types and minimal language extensions, demonstrating their superior expressiveness in formulating a rich set of continuous spatial queries. Finally, we presented how such constructs could be used in kinetic operations, like continuous trajectory joins, to discover patterns over streaming timeseries of object positions.

We further intend to study other interesting cases (e.g., directional semantics) and to investigate query rewriting rules in the presence of such windows. Meanwhile, we have begun tackling evaluation issues concerning windowed implementations of spatial operators and continuous trajectory joins. A challenging task is to take advantage of window subsumption at multiple dimensions and we anticipate two opportunities for query optimization; namely, overlaps between successive window states and containment among coverages concurrently applied over distinct trajectories.

References

[1] D. 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.

[2] A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *VLDB Journal*, 15(2):121-142, June 2006.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *ACM PODS*, pp. 1-16, May 2002.

[4] P. Bakalov, M. Hadjieleftheriou, E. Keogh, and V. Tsotras. Efficient Trajectory Joins using Symbolic Representations. In *MDM*, pp. 86-93, May 2005.

[5] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S.R. Madden, V. Raman, F. Reiss, and M.A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR*, Asilomar, California, January 2003.

[6] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis. Nearest Neighbor Search on Moving Object Trajectories. In *SSTD*, pp. 328-345, August 2005.

[7] P. Bakalov, M. Hadjieleftheriou, and V. Tsotras. Time Relaxed Spatiotemporal Trajectory Joins. In *ACM GIS*, pp. 182-191, November 2005.

[8] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. Tucker. Semantics and Evaluation Techniques for Window Aggregates in Data Streams. In *ACM SIGMOD*, pp. 311-322, June 2005.

[9] M. Mokbel, X. Xiong, and W. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatiotemporal Databases. In *ACM SIGMOD*, pp. 623-634, June 2004.

[10] K. Mouratidis, M. Hadjieleftheriou, and D. Papadias. Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring. In *ACM SIGMOD*, pp. 634-645, June 2005.

[11] K. Patroumpas and T. Sellis. Managing Trajectories of Moving Objects as Data Streams. In *STDBM*, pp. 41-48, August 2004.

[12] K. Patroumpas and T. Sellis. Window Specification over Data Streams. In *ICSNW*, Springer LNCS 4254, pp. 445-464, March 2006.

[13] D. Pfoser, C. Jensen, and Y. Theodoridis. Novel Approaches to the Indexing of Moving Object Trajectories. In *VLDB*, pp. 395-406, September 2000.

[14] M. Stonebraker, U. Çetintemel, and S. Zdonik. The 8 Requirements of Real-Time Stream Processing. *ACM SIGMOD Record*, 34(4):42-47, December 2005.

[15] X. Xiong, M. Mokbel, and W. Aref. SEA-CNN: Scalable Processing of Continuous k-Nearest Neighbor Queries in Spatiotemporal Databases. In *ICDE*, pp.643-654, April 2005.

[16] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-Nearest Neighbor Queries Over Moving Objects. In *ICDE*, pp. 631-642, April 2005.