Approximate Order-k Voronoi Cells over Positional Streams

Kostas Patroumpas
School of Electrical and Computer Engineering
National Technical University of Athens, Hellas
kpatro@dbnet.ece.ntua.gr

Theofanis Minogiannis
School of Electrical and Computer Engineering
National Technical University of Athens, Hellas
el01248@mail.ntua.gr

Timos Sellis
School of Electrical and Computer Engineering
National Technical University of Athens, Hellas
timos@dbnet.ece.ntua.gr

ABSTRACT

Handling streams of positional updates from numerous moving objects has become a challenging task for many monitoring applications. Several algorithms have been recently proposed for providing exact answers particularly to continuous range and k-nearest neighbor queries against current object positions. In this work, we introduce a processing technique for efficiently maintaining an *approximate order-k Voronoi cell* around a certain point of interest when all objects continuously change their locations. This heuristic can easily provide a fairly reliable estimate of the k-nearest neighbors for any query point found inside the constructed cell. We further extend our method to handle positional updates that are not received concurrently for all objects, but instead remain valid for a specific time interval according to a sliding window model. Extensive experimental analysis over synthetic datasets confirms the robustness and scalability of this approach offering near real-time cell maintenance with acceptable error margins.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*spatial databases and GIS*

General Terms

Algorithms, Performance, Experimentation

Keywords

Approximation, Data Streams, Nearest Neighbors, Voronoi Cell, Moving Objects

1. INTRODUCTION

There has been a growing public interest in modern positioning applications that monitor movement of objects, like vehicles in fleet management systems, endangered species in natural parks, people for location-based services, etc. From a data management perspective, this phenomenon has led to increasing amounts of geospatial information flowing through networks, effectively taking the form of *positional data streams* [7]. Ideally, massive streams of frequent location updates from numerous moving objects should be processed online, in order to provide timely responses to multiple *continuous queries*. For instance, a commercial application that provides notification to subscribers according to their registered preferences, may issue messages to users whenever they are passing close to specific locations, informing them about shopping opportunities, tourist attractions, cultural events etc.

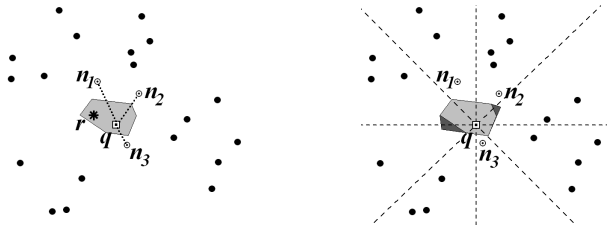

Figure 1: (a) Area of influence for a focal point q. (b) Exact vs. approximate order-k Voronoi cell.

Searching for the k closest locations to a particular query point q is among the most important spatiotemporal continuous queries, commonly referred to as *k-nearest neighbor search*. In the aforementioned location-based setting, notifications could be sent only to those $k = 20$ customers that are currently closest to a store (i.e., the actual point of interest). With respect to streaming locations from numerous moving objects, it is a demanding issue to continuously report the k locations nearest to a given stationary or moving point q, assuming that future movement is not predictable at query time and does not follow a known pattern. Several attractive solutions have been proposed thus far [9, 15, 16], providing exact answers about the current k-nearest neighbors.

In this paper, we turn our focus into a variant of k-nearest neighbor search. Assuming a *focal point* of interest q, we want to maintain its *area of influence* with respect to the k streaming locations closest to q (Fig. 1a). This area is effectively captured by the *order-k Voronoi cell* that is associated to focal point q. Thus, if a query point r asks for its k-nearest neighbors and is found within that cell, it can get response immediately. In other words, this problem reduces to construction and fast maintenance of such polygons when object locations get continuously updated at high rates.

For example, imagine taxis moving in a large metropolitan area. A constellation of smart service centers has been arranged to cover the city, each one maintaining an area affected by its k nearest taxis. When a customer r requires a service, the system could locate her within the cell of such a center q and then notify a currently free taxi in close proximity. As another motivation scenario, consider soldiers fighting in a battlefield. An officer q is in charge of his

nearest troops and has direct contact to headquarters. In a contingency plan, he may call for a helicopter r that can easily locate its nearby soldiers who should evacuate the threatened area at once.

For performance reasons, we have designed an evaluation strategy that calculates *approximate order-k Voronoi cells*. Our method works in main memory and partitions incoming locations into *radial sectors* around focal point q so as to identify quickly a minimal set of positions that influence most the shape of the cell (Fig. 1b). This partitioning of locations into smaller groups reduces computation cost at the expense of accuracy, since search for points affecting the cell is limited within each sector, hence avoiding examination of all possible combinations. Our algorithm only requires knowledge of the k closest streaming locations to q, as well as one additional *sentinel* location per sector. By suitably choosing tight bisectors between sentinel points and k-nearest neighbors to q, we are able to conveniently control the shape of the cell at the direction of each radial sector. Furthermore, as object locations get updated, it is likely that the minimal set of influencing positions is gradually modified, so we expect only local geometric changes at that cell.

We investigate two variations of the problem over the Euclidean plane. First, we assume that all objects send their current location at regular time intervals. For this setting that involves *concurrent updates* from moving objects, the cell must be recomputed periodically. Alternatively, in a *sliding window* model, we consider that the contribution of any single location remains valid for some time and then expires, just before the respective object sends its new position. In this case, cell maintenance takes into account locations received during the most recent time interval.

Our contributions can be summarized as follows:

- We present a simple and fast technique for representing the influence area around a focal point of interest with respect to its current k-nearest object locations. For even faster retrieval of relevant locations, we utilize indexing of object positions against a grid partitioning.
- We provide a variant of the cell computation method that is able to handle positional updates received during a recent time period, assuming that objects do not relay their locations simultaneously.
- We conduct a comprehensive performance evaluation using two synthetic datasets of objects with different movement patterns. Our results indicate that the proposed algorithms can efficiently cope with scalable numbers of moving objects at the expense of acceptable error margins.

The rest of this paper proceeds as follows. In Section 2, we survey related work and in Section 3 we outline fundamental notions concerning Voronoi cells. In Section 4, we describe our technique for constructing an approximate order-k Voronoi cell when all moving objects update their location concurrently. We extend this algorithm in Section 5, for handling asynchronous positional updates using a time-based sliding window. In Section 6, we discuss performance results using synthetic datasets. Finally, Section 7 offers concluding remarks and directions for future research.

2. RELATED WORK

Voronoi diagrams are among the handiest structures in Computational Geometry and have been used in geography, hydrology, biology, computer graphics, etc. In geographic applications they are usually called *Thiessen polygons* and they act as models for spatial processes (e.g., optimal subdivision into service areas, weighted averages over rainfall etc.). A comprehensive survey of Voronoi diagrams and their applications can be found in [2]. Several algorithms have been proposed for efficient construction of Voronoi diagrams, while their mathematical properties have also been extensively studied [4, 10].

In spatial databases, ordinary Voronoi diagrams can be used to answer nearest neighbor queries, by simply identifying the particular cell where a given query point actually lies, e.g., in spatial networks [8] or location-dependent queries [18]. However, these methods are not usually recommended for arbitrary $k > 1$ nearest neighbor search, because this would involve maintenance of order-k Voronoi diagrams with increasing computational and space cost. To avoid such complications, exact order-k Voronoi *cells* were proposed [17] for dealing with moving nearest neighbor queries over static sites (e.g., facilities, restaurants etc.). In our approach, an approximate order-k Voronoi cell is constructed with respect to a static or moving point of interest q, while we assume frequent updates in moving object locations that may continuously influence the shape of this cell.

Approximation of Voronoi cells in d-dimensional space was proposed in [1], offering an ϵ-approximate neighborhood AVN_ϵ of a given site q. This method locates a sample of ϵ-approximate Voronoi neighbors for q, such that AVN_ϵ remains a convex polyhedron with a small number of facets that entirely contains the exact cell. By choosing neighbors more densely when they are found close to q, the boundary of this approximate cell is adapted locally according to the distribution of the neighboring sites, provided that the exact cell is already known. However, it is not straightforward how the sampling criteria of this method can be modified to handle order-k Voronoi cells, as we consider in this work.

Our approximation technique extends the approach proposed in [12] for handling strictly order-1 Voronoi cells. In that algorithm, the well-known heuristic [3, 5, 7] of a planar subdivision into λ sectors originating from the query point q was also applied. Thus, the site closest to q in each sector could be easily determined, as well as the respective bisectors that delineate the cell. Therefore, only a subset of the current object locations from the stream is considered each time, achieving $O(\lambda)$ cost per positional update. A variant of the algorithm was also shown capable to handle only the most recently received locations [12], according to a sliding window model based on their arrival order. Further, the same concepts were utilized in a sensor network setting [13], in order to assist in spatial aggregation of the received measurements. In our approach, apart from the important generalization of the problem to $k > 1$ neighbors, we facilitate searching for closest sites in each sector by utilizing a fixed grid partitioning of the plane. Not only does this indexing structure improve performance even for increased number of objects, but it also enables concurrent execution of the approximation algorithm for several points of interest. Besides, our interpretation of sliding window is based on time indications, hence allowing multiple objects to relay their new location with identical timestamp values.

3. GEOMETRIC BACKGROUND

Voronoi diagrams (also known as *Dirichlet tessellations*) have been used mostly for identifying influence areas around certain points of interest. Given a set P of point *sites* on the plane, their ordinary Voronoi diagram is the unique subdivision of the plane into polygon *cells*, such that each cell is associated with one corresponding site. The important property of this partition is that cell $V(p_i)$ assigned to site p_i contains those points on the plane that have p_i as their *nearest neighbor* among all sites in P:

$$V(p_i) = \left\{ p \mid d(p, p_i) < d(p, p_j),\ \forall\, p_j \in P \setminus \{p_i\} \right\} \quad (1)$$

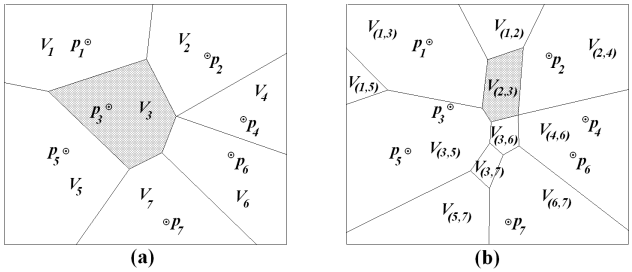

Figure 2: (a) Order-1 and (b) Order-2 Voronoi diagrams for sites on 2-d plane.

where function $d(p, q)$ returns the distance between a pair of points p, q on the plane. Figure 2a shows the Voronoi diagram constructed for seven sites $p_1, p_2, \ldots, p_7$. For instance, any point inside shaded cell V_3 is closest to its associated site p_3 than any other site.

An important observation is that *bisectors* between a given p_i and each p_j of the remaining sites clearly determine the area of influence for p_i. Indeed, each such bisector splits the plane into two half-planes. Half-plane $H(p_i, p_j)$ that contains p_i clearly includes all points closer to p_i than to p_j. It is easily verified that cell $V(p_i)$ can be generated by taking the common area of all such half-planes $H(p_i, p_j)$ defined by site p_i. Notice that some cells are completely bounded by edges, while others are unbounded and extend to infinity. However, each Voronoi cell is convex and its edges are parts of the bisectors defined with the sites of its adjacent cells. Voronoi diagrams can be generalized for point sets at higher dimensional spaces or according to the metric function d used (e.g., Manhattan, Euclidean or weighted distance) [4].

However, another partition of the plane is also possible. Instead of determining the regions closest to a single site, one can perform a partition based on proximity to $k > 1$ sites and thus obtain a higher *order-k* Voronoi diagram [10]. Each cell $V({P_i}^{(k)})$ generated this way, simply contains points that are nearest to the same combination of k sites ${P_i}^{(k)} = \{p_{i1}, p_{i2}, ..., p_{ik}\}$. In practice, each point within the cell is closer to the most distant (i.e., k-th) neighboring site of ${P_i}^{(k)}$ than any other site $p_j \notin {P_i}^{(k)}$. Formally:

$$V({P_i}^{(k)}) = \Big\{ p \mid \max_{p_g} \Big\{ d(p, p_g),\ \forall\, p_g \in P_i^{(k)} \Big\} \leq \min_{p_j} \Big\{ d(p, p_j),\ \forall\, p_j \in P \setminus P_i^{(k)} \Big\} \Big\} \quad (2)$$

Sites constituting subset ${P_i}^{(k)}$ are often called *generators* of that particular cell [10]. In this context, order signifies the fixed number k of generators for each cell of the diagram. An order-2 Voronoi diagram is illustrated in Fig. 2b, constructed for the same sites as in Fig. 2a. Evidently, any point inside shaded cell $V_{(2,3)}$ has the same two nearest neighbors (order $k = 2$), namely the generators p_2 and p_3 of that cell. Note that each order-k cell is convex, but it may contain less than k or even none of its generator sites.

Like standard order-1 Voronoi cells, an order-k cell can be constructed using bisectors between its generators and the rest of the sites. Thus, an equivalent definition of order-k Voronoi cell is:

$$V(P_i^{(k)}) = \bigcap_{p_j \in P \setminus P_i^{(k)}} [H(p_{i1}, p_j) \cap \ldots \cap H(p_{ik}, p_j)] \quad (3)$$

Yet another variant of these diagrams dictates that generating sites be sorted, so that the resulting *ordered order-k* Voronoi cells are characterized with an ordering of the k-nearest neighbors associated to them. This arrangement can distinguish among the first, the second, and up to the k-th nearest neighboring site for each cell. In this paper, we deal with singleton order-k Voronoi cells (i.e., not entire diagrams) without maintaining their generators sorted.

4. APPROXIMATE ORDER-K VORONOI CELL COMPUTATION

Creation of the *exact* order-k Voronoi cell concerning a specific focal point q of interest is a computationally intensive process when repeatedly applied over streaming locations of numerous moving objects. The reason is that we may potentially need to examine all bisectors between every incoming location with each of the k-NN sites closest to q, in order to verify whether they contribute to the shape of the cell. The fact that the set of k-NN points may also be time-varying with respect to a stationary or moving focal point q, makes matters even worse. Next, we attempt to create an *approximate* order-k Voronoi cell by considering only a small fraction of the incoming object locations.

We assume a set $P = \{o_1, o_2, \ldots, o_N\}$ of N objects moving over the Euclidean plane. Each object sends tuples $\langle o_i, p_i, \tau \rangle$ to a central processing engine, where p_i represents the current position of object o_i at timestamp value τ. The basic idea behind our algorithm is to bound the cell associated to focal q, by suitably taking representative bisectors at all directions around q. A straightforward way to achieve this, is with a set of λ vectors $l_1, l_2, \ldots, l_\lambda$ originating from q and uniformly arranged with an angle $\theta = 2\pi/\lambda$ between each pair of successive vectors. In effect, these vectors create a planar partition into a set S of λ similar *radial sectors* $S = \{S_1, S_2, \ldots, S_\lambda\}$ having q as their common vertex (Fig. 3a). This construct has already been applied for other similar geometric problems over spatial streams, like approximation of diameter [5], convex hulls [7], and order-1 Voronoi cells [12, 13].

In this section, we present a cell approximation technique assuming *simultaneous* positional updates from all monitored objects at each time instant τ. In Section 5, we consider availability of a limited portion from the unbounded data stream of timestamped positions, taking into account only its most recently arrived items.

4.1 Handling Concurrent Location Updates

In general, the algorithm for computing order-k Voronoi cells has two main steps: (i) a *filtering stage* for detecting a minimal set of influence points that will be further examined during cell construction, and (ii) a *bounding stage* that uses a limited number of bisectors to determine the shape of the approximate cell. Next, we describe each phase in detail.

4.1.1 Filtering stage

Having applied a radial partitioning of the plane, each incoming location p can be assigned to the sector S_i it falls within. Among all positions received at the current timestamp, each S_i eventually retains up to $k + 1$ points only, while it filters out the rest. This *candidate set* $L(S_i)$ includes at most $k+1$ locations in S_i, i.e., the closest to focal point q in ascending distance. Clearly, this is a conservative strategy, because it may occur that all k-nearest neighbors could be found in the same sector, whereas it suffices one additional point in that sector to bound the cell, as explained in Section 4.1.2. Overall, it is guaranteed that the unified set of candidate points from all $L(S_i)$ does not miss any of the correct k-nearest neighbors to q (*Function UpdateSectors* in Algorithm 1).

Consequently, from candidate sets $L(S_i)$ we must identify the current k-NN locations. Since the number λ of sectors is usually a small integer, we adopt a naïve policy that visits all sectors in succession and marks each location eligible for becoming one of k-nearest neighbors (*Function GetInfluenceSets* in Algorithm 2).

Algorithm 1 ***Concurrent Updates***

Function *UpdateSectors* (point q, int λ, int k)

1: Take focal point q as center and divide Euclidean plane E into λ equiangular sectors $S_1, S_2, \ldots, S_\lambda$
2: $L(S_i) \leftarrow \emptyset$
3: **while** ($p \leftarrow$ next location currently in P) $\neq$ NULL **do**
4: $S_i \leftarrow$ sector s.t. $p\ inside\ S_i$.
5: **if** $|L(S_i)| < k+1$ **then**
6: Insert p in ordered sequence $L(S_i)$
7: **else** $v \leftarrow (k+1)$-th point $\in L(S_i)$
8: **if** $dist(p,q) < dist(v,q)$ **then**
9: Insert p in ordered sequence $L(S_i)$
10: Discard v from $L(S_i)$
11: **return** $L(S_i), i = 1, \ldots \lambda$

Algorithm 2 ***Order-k Voronoi Cell Approximation***

Function *GetInfluenceSets* (point q, int λ, int k, points L)

1: $kNNList \leftarrow \emptyset$, $MList \leftarrow \emptyset$
2: **for each** sector $S_i, i = 1, \ldots \lambda$
3: **while** ($p \leftarrow$ next point $\in L(S_i)$) $\neq$ NULL **do**
4: **if** $|kNNList| < k$ **then**
5: $kNNList \leftarrow kNNList \cup p$
6: Mark p as chosen at sequence $L(S_i)$
7: **else**
8: **while** ($r \leftarrow$ next point $\in kNNList$) $\neq$ NULL **do**
9: **if** $dist(p,q) < dist(r,q)$ **then**
10: Insert p prior to r into $kNNList$
11: Shift all subsequent locations in $kNNList$
12: Mark p as chosen at sequence $L(S_i)$
13: **if** ($v \leftarrow (k+1)$-th point $\in kNNList$) $\neq$ NULL
14: **then** Discard v from $kNNList$
15: Unmark v at its respective sequence $L(S_j)$
16: **else break**
17: **for each** sector $S_i, i = 1, \ldots \lambda$
18: $m(S_i) \leftarrow$ the first *unmarked* point in sequence $L(S_i)$
19: $MList \leftarrow MList \cup m(S_i)$
20: **return** $kNNList$, $MList$

Function *CreateCell* (point q, points $kNNList$, points $MList$)

1: $V'(q) \leftarrow$ universe E
2: **for each** point $m(S_i) \in MList$
3: $B_i \leftarrow \emptyset$
4: **for each** point $n_j \in kNNList$
5: Insert into B_i the bisector between $m(S_i)$ and n_j
6: $b_i \leftarrow$ choose bisector from B_i that is closest to q
7: Crop $V'(q)$ by b_i
8: **return** $V'(q)$

We make use of a $kNNList$ that maintains locations ordered by increasing distance from q. Starting from sector S_1, we take its k locations closest to focal point q. If S_1 has less than k points available, we continue with S_2, until $kNNList$ is filled up with k points. Afterwards, at every subsequent sector S_i in counterclockwise order, we check if any of its retained locations is closer to q than a point currently stored in $kNNList$. In that case, we insert this new location at the proper node of $kNNList$ and evict the superfluous location at the tail of $kNNList$. The discarded location is then unmarked at the candidate set $L(S_i)$ where it belongs. At the end of this cycle, all sectors will have been examined and the correct set of k locations closest to q is stored in $kNNList$.

In addition, our algorithm makes use of a *sentinel location* m_i at each sector S_i. The chosen sentinel is the point in S_i that is closest to q and is not a member of the k-nearest neighbors. Formally:

$$d(q, m_i) = \min \Big\{ d(q,t) \mid t \in L(S_i) \wedge \wedge\, d(q,t) > \max\{d(q,n) \mid n \in kNNList\} \Big\}$$

Since each $L(S_i)$ maintains locations in ascending distance from q, the respective sentinel m_i can be found trivially, after excluding points definitely taken in $kNNList$. Set $MList$ collects all such sentinels (Lines 17-19 in *Function GetInfluenceSets*).

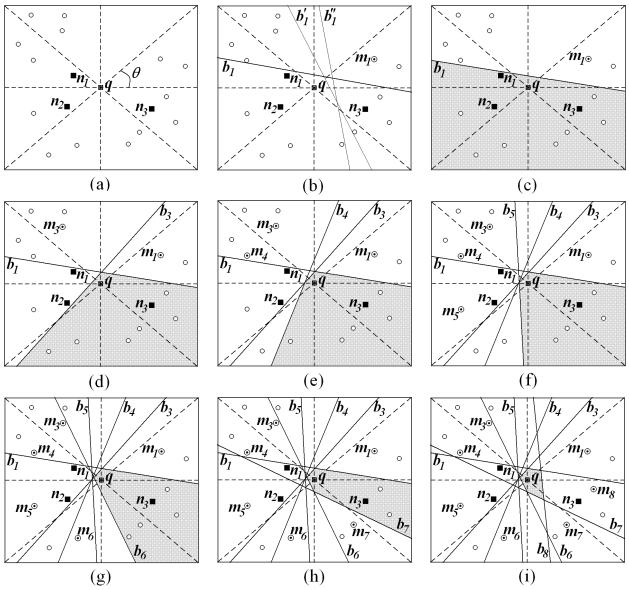

Figure 3: Approximate order-3 Voronoi cell construction.

When it comes to cell construction, the sentinel point acts as a delegate for all locations in its sector. Instead of delineating all possible bisectors between the set of k-NN points and the set of numerous locations in each sector, we are contented to only take bisectors related to sentinel points. Hence, we relax expectations regarding cell accuracy, but we gain considerably in processing time.

4.1.2 Bounding stage

Once the filtering stage has concluded, all k-nearest neighbors and λ sentinel points are known. These locations will be used for approximating the cell at current time τ (*Function Create Cell*).

Initially, we assume that approximate cell $V'(q)$ is equal to E, the entire universe of interest where any possible location is being recorded. For each sentinel m_i we delineate all its k bisectors with the k-NN points. But, for reducing processing cost even further, we choose bisector b_i that is closest to q and thus bounds the cell most tightly (Fig. 3b). Accordingly, by taking the most *restrictive bisector* b_i for each sector S_i in counterclockwise order, we can gradually crop E and obtain the approximate $V'(q)$ (Fig. 3c-3i). In case a sector does not contain any sentinel points (e.g., sector S_2 in Fig. 3c), it does not affect $V'(q)$ at all.

Overall, the cost of the entire strategy is $O(1)$ *per incoming location*, since testing for inclusion in a sector has standard cost. Besides, the result of filtering contains at most $\lambda \cdot (k+1)$ points, whereas up to λ bisectors contribute to the final shape of the cell.

4.2 Facilitating Retrieval of Locations

The aforementioned technique provides a fast approximation of order-k Voronoi cells, but it is able to handle a *single* focal point q. In case we need to handle *multiple* focal points, we have to prepare a separate partitioning into radial sectors for each one of them. What's more, we have to reexamine all $O(N)$ incoming locations independently for each such partitioning. In a typical monitoring application, such a policy would fail to provide timely results, since it performs an exhaustive search among object positions, no matter whether they actually contribute to cell construction or not.

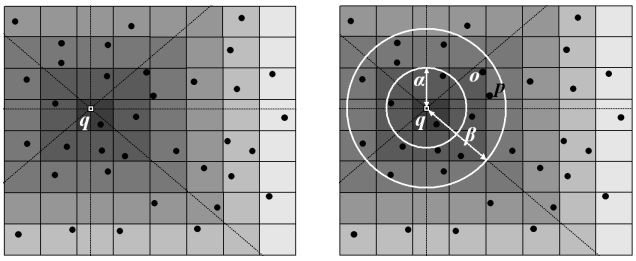

Figure 4: Rings of grid squares around focal site q.

In order to handle multiple focal points and scalable numbers of streaming positions, we make use of a simple indexing scheme [6] based on a uniform partitioning into $c \times c$ *grid squares*[1]. This partitioning G is static and covers the entire area of interest E where objects move (Fig. 4a). In contrast to radial partitioning in sectors, G is independent of the current position of any focal point q.

Typically, each incoming location is hashed against the grid and it is assigned to a square. Thus, each grid square maintains a list of point locations currently found inside its area. This preprocessing stage takes place at each new timestamp τ, involving all positional updates received at τ. Regarding Voronoi cell construction, this indexing of locations reduces drastically the cost of the filtering stage. Indeed, it suffices that each focal point q actually examines locations from just a few grid squares in its neighborhood. Since distribution of streaming positions may be varying, each focal point may need to retrieve differing numbers of squares, while each square may contain a fluctuating count of object positions. However, these squares may be searched in waves, organized in levels $r_0, r_1, \ldots$ according to their distance from focal point q. Each wave involves a *ring* of squares, all of them equally away from the square containing q. In Fig. 4, successive rings of square cells around q are depicted in graduated gray color.

Given a focal point q, the algorithm initially retrieves locations stored at the grid square (level r_0) where q currently lies. If this square does not contain enough locations for securely specifying the k-nearest neighbors and λ sentinels needed for creating Voronoi cell $V'(q)$, then we retrieve the ring of 8 squares at next level r_1. This process continues at successive ring levels, until the minimal set of influence locations has been finalized.

Yet, there is a subtlety concerning how influence point sets are determined. As a criterion, we utilize the circle centered at q with the maximum radius α that does not exceed the boundary of current ring r_j (Fig. 4b). If all $k+\lambda$ influence points are completely within this *inner circle*, then we can be sure the detected influence points are correct. Otherwise, it may occur that a location p in a square at next level r_{j+1} could be closer to focal site q than a presumed influence point o. In this case, we have to continue searching in squares at successive ring levels. These additional rings are covered by the *outer circle*, also with center q and radius the distance β of the furthest vertex of ring r_j from q. Since higher-level rings include more squares, this incurs retrieval of increasing numbers of object locations. Search for influence points continues until all squares covered by the outer circle are checked. As we verified empirically, it should be expected that retrieval ends up after examining a very small number of rings, depending on grid granularity.

4.3 Properties of Approximate Cells

Our methodology is also applicable for standard Voronoi cells (i.e., $k = 1$). Although this solution has close similarity to the one in [12], the setting is different. We allow a moving point of interest, and our technique also involves identification of the nearest neighbor before proceeding to construct its order-1 Voronoi cell.

Higher order-k Voronoi cells created approximately using influence set points have some interesting properties. As already noted, construction of order-k Voronoi cell helps in k-nearest neighbor search for query points found within the cell. In addition:

LEMMA 4.1. *Approximate $V'(q)$ always contains exact cell $V(q)$.*

PROOF. Cell $V'(q)$ is constructed from a subset of bisectors that construct $V(q)$. For contradiction, suppose an internal point $p \in V(q)$, but $p \notin V'(q)$. Then, a vertex r of $V'(q)$ is found closer to focal site q than point p. But then, vertex r should be the intersection point of bisectors that have not been considered during construction of the exact $V(q)$, which does not hold. □

Given that our method provides a *superestimation* of the exact cell, it follows that construction of approximate order-k cells for a set of sites does not create a partitioning of the entire plane, because these cells are not mutually disjoint. Besides, the shape of the computed cell is compact and always remains a convex polygon with a limited number of edges (λ at most), as stated below:

LEMMA 4.2. *Approximate cell $V'(q)$ is convex.*

PROOF. $V'(q)$ is constructed from the intersection of $\lambda \geq 2$ semi-planes defined by the respective bisectors (Fig. 3). Since these semi-planes are always convex and the intersection of convex polygons is also convex, the result follows. □

In each radial sector S_i, all bisectors taken between sentinel location m_i and each of the k-NN points always intersect S_i. To verify this, assume that a bisector b_j does not intersect any of two vectors defining sector S_i. Then focal point q would lie closer to m_i than to k-NN point n_j that controls bisector b_j (contradiction). This observation directly leads to the following:

LEMMA 4.3. *Focal point q is always contained within its associated approximate order-k Voronoi cell.*

Note that approximate order-k Voronoi cell is primarily controlled by the set of k-nearest neighbors. Although focal site q clearly determines those k locations, bisectors are taken between this k-NN set and the group of sentinels further away from q.

5. APPROXIMATE ORDER-K VORONOI CELL OVER SLIDING WINDOWS

In this section, we discuss a variant of the cell approximation technique in order to handle a more realistic situation when not all objects send updates concurrently, e.g., vehicles moving in a city. In particular, at each timestamp τ we assume that a varying portion of objects moves to another location p', invalidating their previous position p. Each recorded location remains valid during a time interval $\Delta\tau$, which may differ among objects. This displacement occurs asynchronously and gradually affects all objects. At any instant τ, the system has to deal with locations that arrived at different times, but at most one location is available for each object.

As is typical in streaming applications [5, 11, 12], not all information is given for processing, but only a portion received during a recent interval ω. Continuous query evaluation repeatedly applies this *sliding window* of fixed size ω in order to fetch the most recent data items, ignoring obsolete recordings. Next, we describe an algorithm for incrementally maintaining approximate order-k Voronoi cells over time-persistent positions from moving objects.

[1]We preferred term *grid squares* instead of *grid cells* to avoid confusion with Voronoi cells.

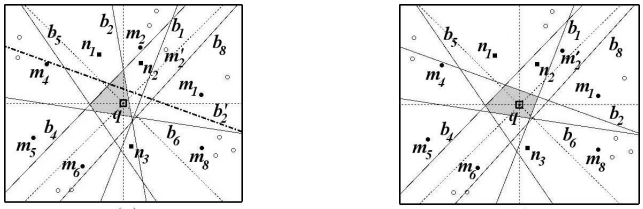

Figure 5: Location expiring from a sliding window.

5.1 Handling Time-Persistent Locations

Suppose that focal point q remains fixed at any single time instant τ. From a geometric perspective, the algorithmic concept remains identical to the one proposed in Section 4. Again, we need to identify the k-nearest neighbors to a focal point q, as well as a sentinel location for each of the λ radial sectors originating from q. The challenge in this case is (*i*) how to choose influence points at each iteration and (*ii*) how to maintain the shape of the approximate Voronoi cell, avoiding construction from scratch as much as possible. Note that cell maintenance is not feasible for concurrent updates, because relative distances between influence points may change unpredictably between successive timestamps.

5.1.1 Filtering stage

For approximately determining order-k Voronoi cells, we take into consideration positional updates that arrived within the latest ω timestamps from current time τ_{NOW}. As the applied window proceeds in pace with time evolution, fresh positions at τ_{NOW} are taken in, at the expense of locations that expire because they had been received more than ω time units earlier. Hence, the lifetime of each particular location within the window can be at most ω time units. For ease of presentation, we assume that $\omega \leq \Delta\tau$, so that we need not check whether an object has reported its location more than once within interval ω. Of course, we can also allow multiple locations for the same object within the current window extent, without any modification to the algorithm presented next.

For each radial sector S_i, we intend to retain the $k+1$ proximal locations to focal point q at each $\tau \in (\tau_{NOW} - \omega, \tau_{NOW}]$. As explained in Section 4.1, this conservative approach guarantees that the minimal set of influence points is always available. More importantly, we should also anticipate that, in case a location m_2 expires, another location m_2' with newer timestamp must be available in the same sector, in order to bound the cell (Fig. 5).

Radial sectors are examined for identifying all points that may possibly affect the approximate cell (*Function UpdateSectors* in Algorithm 3). A new iteration of the algorithm starts at each new timestamp value τ_{NOW}. Initially, all expiring locations should be removed from memory, as they represent positional updates older than ω time instants from τ_{NOW} (Lines 2-5). Afterwards, for each new position p, we find out its respective radial sector. If this sector S_i contains less than $k+1$ points for the current τ_{NOW}, then p is inserted into its set $L(S_i)$ of candidate points (Lines 7-9). In case S_i has already stored more than $k+1$ points at τ_{NOW}, we should verify whether new location p invalidates an existing candidate v. This can only happen if, compared with p, position v is further away from focal point q and it has an older timestamp value (Lines 10-13). Intuitively, by no means can this older candidate v influence the cell in the future, because p will always contribute a more tight bisector and will certainly expire later.

Because of similar invalidations in the candidate sets of sectors, the expected total number of retained positions at any iteration can-

Algorithm 3 ***Sliding Window Updates***

Function *UpdateSectors* (point q, int λ, int k, interval ω)
1: Take focal point q as center and divide Euclidean plane E into λ equiangular sectors $S_1, S_2, \ldots, S_\lambda$
2: **for each** $L(S_i), i = 1, \ldots \lambda$
3: **for each** $o_j \in L(S_i)$
4: **if** $o_j.\tau \leq \tau_{NOW} - \omega$ **then**
5: Discard o_j from $L(S_i)$
6: **while** ($p \leftarrow$ next location at τ_{NOW} in P) $\neq$ NULL **do**
7: $S_i \leftarrow$ sector s.t. p *inside* S_i.
8: **if** $|\{m \in L(S_i)$ with $m.\tau = \tau_{NOW}\}| < k+1$ **then**
9: Insert p in ordered sequence $L(S_i)$
10: **else** $v \leftarrow$ point $\in L(S_i)$ with max distance from q
11: **if** $dist(p,q) < dist(v,q)$ **and** $v.\tau \leq \tau_{NOW}$ **then**
12: Insert p in ordered sequence $L(S_i)$
13: Discard v from $L(S_i)$
14: **return** $L(S_i), i = 1, \ldots \lambda$

Function *MaintainCell* (point q, points $kNNList$, points $MList$)
1: **if** any of the points in $kNNList$ has changed **then**
2: Call *CreateCell*(q, $kNNList$, $MList$)
3: **else**
4: **for each** sector $S_i, i = 1, \ldots \lambda$
5: **if** $m(S_i)$ was replaced by $m'(S_i) \in MList$ **then**
6: $B_i \leftarrow \emptyset$
7: **for each** point $n_j \in kNNList$
8: Insert into B_i the bisector between $m'(S_i)$ and n_j
9: $b_i' \leftarrow$ choose bisector from B_i that is closest to q
10: Discard existing bisector b_i produced by $m(S_i)$
11: Adjust existing V_q by b_i'
12: **return** V_q

not exceed $M = \lambda \cdot \omega \cdot (k+1)$. As we evaluated empirically, this maximum limit is never reached, since practically only a smaller fraction of positional updates is maintained (60% for the worst case shown in Fig. 13). This also depends on the distribution of incoming positions with respect to focal site q.

Premature invalidations also justify why partitioning into grid squares cannot be applied (as in Section 4.2), since locations may be discarded well before their expiration time, in case a more fresh position is found closer to q.

5.1.2 Bounding stage

Having found the required candidate points $L(S_i)$ for each sector, we must determine the set of k-nearest neighbors to focal site q and λ sentinel points that jointly influence the cell. This is performed with routine *GetInfluencePoints* presented in Section 4.1.

The algorithm for cell maintenance is shown in routine *MaintainCell* (Algorithm 3). Before proceeding to adjust the shape of the cell produced at the previous iteration of the algorithm, we must verify that the set of k-NN points remains the same. If not, the existing cell refers to different k neighbors to focal site q, so it is invalid. In that case, we should make use of *Function CreateCell* (Section 4.1) to construct a new cell utilizing the current influence points. If the k-NN set remains intact, then we simply attempt to delineate bisectors with respect to each new sentinel point. As soon as we choose the most restrictive bisector for each affected sector, we replace the existing boundary line and adjust the Voronoi cell accordingly (Lines 3-12 in *Function MaintainCell*).

In summary, the total cost of the sliding window algorithm is $O(1)$ *per incoming location*. This is evident, since the filtering stage returns at most $\lambda \cdot \omega \cdot (k+1)$ points, while all other steps have costs similar to the case of concurrent object updates.

5.2 Extensions

In case of a moving focal site q, the set of influence points may

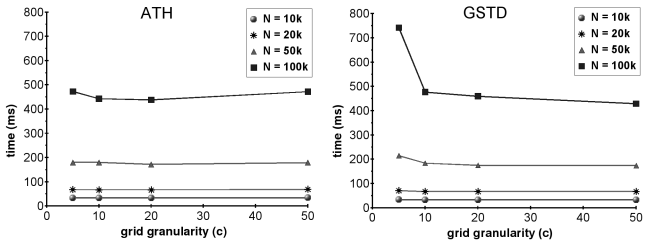

Figure 6: Indexing cost.

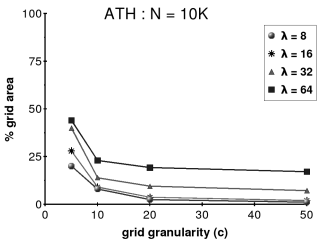

Figure 7: Rings visited.

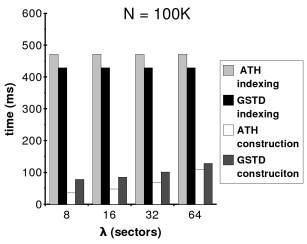

Figure 8: Processing time.

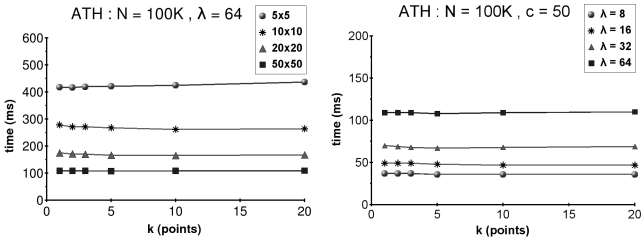

Figure 9: Cell construction time.

change arbitrarily at each iteration. However, we may still take advantage of the existing candidate locations in order to get an initial rough cell. Indeed, as soon as a new radial partitioning around the new position of q is settled, we can redistribute valid candidate locations into the new sectors. We henceforth consume all newly arrived locations for the current timestamp, exactly as if q were fixed. For skewed distribution of retained candidate points, it may occur that all of them fall into certain sectors or even into a single one. Nevertheless, starting from τ_{NOW}, each sector will be filled with current locations, so eventually it will not be empty.

A spatial analog of a *tumbling window* [11] can also be applied, where each window instantiation contains disjoint locations and any single point affects the cell only once. In fact, we get an approximate order-k Voronoi cell every ω time instants, considering only locations received during period ω. Thus, Voronoi cell is computed periodically at each interval ω, so it suffices to maintain $k+1$ locations that are closest to q anytime during this period.

6. EXPERIMENTAL EVALUATION

In this section, we report indicative results from an experimental validation of order-k Voronoi cell approximation for concurrent location updates as well as for the sliding window case.

All experiments were performed on an Intel CeleronM 1.6GHz CPU running GNU/Linux Ubuntu with 512 MB of main memory. We generated two types of synthetic datasets. The first class (denoted as ATH) represents trajectories of up to 100000 objects moving at various speeds along the road network of greater Athens. By calculating shortest paths between random nodes across the network and then taking a sample at 200 timestamps from each such route, we obtained point sets for network-constrained movement concerning varying numbers of objects (N = 10k, 20k, 50k, 100k).

We also utilized $GSTD$ framework [14] to generate sets of objects originally concentrated in a small area that subsequently diverge towards all directions under a uniform distribution. To allow both fast and slow movement, we generated separately two initial point sets of 50000 objects each, using different speed parameters, and then we merged the results. Finally, we produced subsets for varying number N of objects, always for 200 time units.

6.1 Concurrent Updates

As explained in Section 4.2, evaluation of single focal points needs to examine exhaustively all current locations. This was also verified experimentally, showing impractical response times for scaling number of objects. Due to lack of space, we present results for the more efficient variant that utilizes a grid partitioning. We ran simulations concerning a single moving focal point for 200 time units and calculated averages for the measured parameters.

The first set of experiments concerns selection of the appropriate grid granularity. Figure 6 plots the cost for hashing incoming locations against grids of $c \times c$ squares. It is easily seen that indexing depends solely on input size and a 50×50 grid is superior than any other granularity for all datasets. Figure 7 illustrates the fraction of grid area being searched for identifying influence points when $N =$ 10k. The finer the granularity becomes, the less area is examined. With 50×50 squares the percentage of the area searched never exceeds 25%, for any choice of radial sectors. Even for such a dataset of sparse locations, the influence set is found using a limited number of squares, thus avoiding retrieval of superfluous points.

Construction times are depicted in Figure 9. Again, we observe that the finest grid granularity is better and is not sensitive to the number k of nearest neighbors. The situation is even better for greater number of objects, so we henceforth utilize the 50×50 granularity in all subsequent experiments. As illustrated in Fig. 9(right), absolute construction times depend on the number λ of sectors. This pattern also holds for decreasing number of monitored objects, although at reduced costs.

Figure 8 plots the breakdown of processing time for both datasets, considering N =100K objects for varying radial partitioning. First, we notice that indexing cost is almost constant for both datasets. Further, $GSTD$ incurs a slightly worse construction cost due to uniform distribution of object locations within the rings, so the algorithm has to carry out additional checks for determining the k-NN points. Obviously the cost of indexing is disproportional compared to cell construction cost, but it can be compensated when multiple focal points get evaluated against the same grid partitioning, as mentioned in Section 4.2.

We assessed approximation quality by measuring the additional area attributed to the approximate Voronoi cell compared to the exact one. Error is generally expected to reduce with the increase of λ, but grows for more neighbors (k). Indeed, a finer radial partitioning (i.e., greater λ values) offers more bisectors, thus bounding the cell more tightly. In contrast, a greater k increases the number of bisectors ignored during cell construction. Since for each sector we examine k bisectors but we finally choose only one, it is evident that we loose in accuracy. This is reflected in Fig. 10, where relative error is higher than expected for larger k. The balance between the two parameters k and λ is not clear-cut, because it is heavily affected by the arrangement of moving objects in the plane, with respect to focal point q. This becomes more evident in Fig. 11,

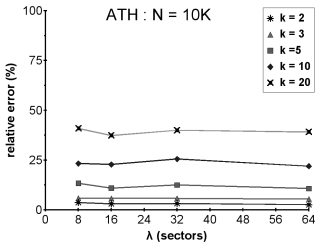

Figure 10: Relative error.

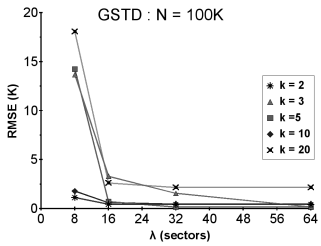

Figure 11: RMSE.

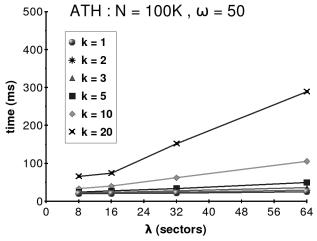

Figure 12: Response time.

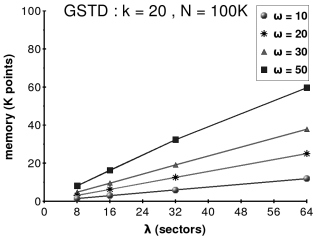

Figure 13: Memory consumption.

where uniform distribution of dataset $GSTD$ causes considerable drop of the RMSE index for subdivisions into more sectors.

6.2 Sliding Window

In Figure 12 we illustrate the average response time of cell approximation for the worst case tested. Evidently, time deteriorates for growing number of neighbors and finer radial partitioning. Besides, as mentioned in Section 5.1, the number of candidate points is directly proportional to k, λ and ω. This effect is clarified in Fig. 13, where the cardinality of candidate points escalates linearly with increasing parameter values. The initial k-NN search is done extensively involving all stored points, so processing time gets worse as more points need examination. However, for smaller window size ($\omega = 10, 20$ timestamps), we observed far better response times.

Regarding approximation quality (Fig. 14), relative error (again based on areas) appears greater for $GSTD$ due to existence of points in every sector. In that case, more bisectors are neglected compared to ATH, where the linearity of trajectory samples makes some sectors completely empty. Nonetheless, one of these bisectors may crop a considerable portion of the cell area during construction. The possibility to ignore any such bisector is unknown and cannot be estimated beforehand. Moreover, each chosen bisector determines the shape of the approximate cell only locally, but it is mainly the arrangement of bisectors in the plane that finalizes the cell. Although we choose the best bisector for each sector, its combination with the rest may not be beneficial. Despite this limitation, approximation error for ATH never exceeds 30% even for $k = 20$, which is fairly acceptable for such a great number of objects.

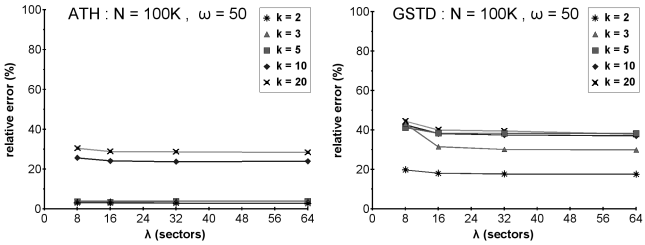

Figure 14: Relative error (sliding window).

7. CONCLUDING REMARKS

In this paper, we propose an approximation technique for calculating order-k Voronoi cells for designated points of interest. The general concept can be applied to handle positional updates concurrently arriving from multiple moving objects, as well as time-persistent locations examined over a recent time period. We have empirically verified that both variants have very small response time for scalable numbers of objects at diverse motion patterns. In addition to small memory footprint, our method also achieves acceptable error with respect to the expensive exact cell.

In the future, we intend to study the effect of taking multiple bisectors to bound the cell at each radial direction. It is expected that this policy may reduce approximation error, although it probably incurs additional cost for cell construction. Besides, investigation of analytical error bounds with respect to radial partitioning and the number of nearest neighbors is also a challenging research topic.

8. REFERENCES

[1] S. Arya and A. Vigneron. Aproximating a Voronoi Cell. *Technical Report HKUST-TCSC-2003-10*, Hong Kong University of Science and Technology, 2003.

[2] F. Aurenhammer. Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys*, 23(3): 345-405, September 1991.

[3] G. Cormode and S. Muthukrishnan. Radial Histograms for Spatial Streams. *Technical Report*, DIMACS *TR:2003-11*, 2003.

[4] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. 2nd edition, Springer-Verlag, Berlin, 2000.

[5] J. Feigenbaum, S. Kannan, and J. Zhang. Computing Diameter in the Streaming and Sliding-Window Models. *Algorithmica*, 41(1):25-41, October 2004.

[6] V. Gaede and O. Günther. Multidimensional Access Methods. *ACM Computing Surveys*, 30 (2): 170-231, 1998.

[7] J. Hershberger and S. Suri. Adaptive Sampling for Geometric Problems over Data Streams. In *ACM PODS*, pp. 252-262, June 2004.

[8] M.R. Kolahdouzan and C. Shahabi. Voronoi-based k-Nearest Neighbor Search for Spatial Network Databases. In *VLDB*, pp. 840-851, September 2004.

[9] 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.

[10] A. Okabe, B. Boots, K. Sugihara, and S.N. Chiu. *Spatial Tesselations, Concepts and Applications of Voronoi Diagrams*. 2nd edition, John Wiley & Sons Ltd., 2000.

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

[12] M. Sharifzadeh and C. Shahabi. Voronoi Cell Computation on Geometric Data Streams. *Technical Report TR-04-835*, University of Southern California, March 2006.

[13] M. Sharifzadeh and C. Shahabi. Utilizing Voronoi Cells of Location Data Streams for Accurate Computation of Aggregate Functions in Sensor Networks. *GeoInformatica*, 10(1): 9-36, March 2006.

[14] Y. Theodoridis, J.R.O. Silva, and M. Nascimento. On the Generation of Spatiotemporal Datasets. In *SSD*, pp. 147-164, July 1999.

[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, N. Koudas. Monitoring k-Nearest Neighbor Queries Over Moving Objects. In *ICDE*, pp. 631-642, April 2005.

[17] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based Spatial Queries. In *ACM SIGMOD*, pp. 443-454, June 2003.

[18] B. Zheng and D. L. Lee. Semantic Caching in Location-Dependent Query Processing. In *SSTD*, pp. 97-113, July 2001.