Approximate Voronoi Cell Computation on Geometric Data Streams

Mehdi Sharifzadeh* and Cyrus Shahabi

Computer Science Department
University of Southern California
Los Angeles, California 90089-0781
`[sharifza, shahabi]@usc.edu`

Abstract. Several studies have exploited the properties of Voronoi diagrams to improve variations of the nearest neighbor search on stored datasets. However, the significance of Voronoi diagrams and their basic building blocks, Voronoi cells, has been neglected when the geometry data is incrementally becoming available as a data stream. In this paper, we study the problem of Voronoi cell computation for fixed 2-d site points when the locations of the neighboring sites arrive as *geometric* data streams. We show that the non-streaming solution to the problem does not meet the memory requirements of streaming applications over a sliding window. Hence, we propose AVC and AVC-SW, two approximate streaming algorithms that compute ε-approximations to the actual Voronoi cell in $O(\kappa)$ using $O(\kappa)$ space where κ is their sample size. With the sliding window model, we prove both theoretically and experimentally that AVC-SW significantly reduces the average memory requirements of the classic algorithm, specially when the window size w is large, which is the case in real-world scenarios.

1 Introduction

Different variations of the Voronoi diagrams have been theoretically studied in the field of computational geometry. The Voronoi diagram of a set of points partition the space into a set of convex polygons so that each polygon contains exactly one point of the set. The polygon corresponding to each point p contains the points in the space that are closer to the point p than to any other point in the set. The computational geometry literature refers to the polygons as Voronoi cells, Dirichlet regions, Thiessen polytopes, or Voronoi polygons [5, 13].

In practice, different research areas in databases such as spatial databases, content-based image retrieval, and data mining have exploited the properties of the Voronoi diagrams to address variations of the nearest neighbor search in different spaces [14]. The body of research in these areas is focused on improving the performance of the nearest neighbor queries when the data tuples are stored as a massive dataset. However, the emerging trend of the data stream research exposes several fundamental challenges to this problem when the data in its entirety

* Contact author, phone: 1 (213) 740-2295, fax: 1 (213) 740-5807.

is not available to be processed. This radical change in the assumptions arises when modern applications generate bulky fast-rate data streams. A large class of these applications continuously generate *geometric* data streams. Examples include GPS equipments in today's personal devices such as cell phones, PDAs, laptops and cars that generate streams of geospatial data as well as more special-purpose applications such as sensor networks [3] and immersive environments [15][1].

The adaption of the idea of Voronoi diagrams to various forms of the nearest neighbor search for geometric data streams requires reconsideration of the algorithmic aspects of building these data structures. Although a few research papers have been published recently to revisit classic computational geometry problems in a data stream framework [6, 4, 10], to the best of our knowledge no study has reconsidered the problem of building Voronoi diagrams or related data structures in this framework. In this paper, we study the problem of computing Voronoi cells of fixed points with respect to a *sliding window* over the *data stream* of *2-d site* points. The cells can be used to solve different problems such as reverse nearest neighbor search [17]. As a real-world application, consider the sensor nodes in a sensor network deployed to monitor a physical phenomena such as soil temperature. The average temperature of an area can be computed as the weighted average of the temperature values recorded by each sensor node to provide a seamless average, independent of the density of the network. In this case, each node's weight can be the area of its Voronoi cell [16]. Each immobile node, knowing its fixed location, frequently receives the locations of other mobile/immobile nodes as a geometric data stream and maintains its Voronoi cell with respect to these locations.

We show that with the general *time series* model [12], the classic algorithm for computing *exact* Voronoi cell of a point p simply drops any newly arrived point which is not changing the cell. However, with the sliding window model, where the size of the window is w, the classic algorithm requires to store all the w points (i.e., $O(w)$ space complexity) to continuously maintain the Voronoi cell given the w recent points. The reason is that even though an arriving point may not be changing the cell at the time of the point's arrival through the stream, it may cause a change in future due to the deletion of old points from the window. To be precise, with a fixed w, w insertions and w deletions occur during the lifetime of a point inside the window. Any single deletion in the window may cause *any* point in the window to contribute to the cell while any insertion may end a point's contribution (see the motivating example in Section 3).

Note that in real-world scenarios w is very large because its size is only limited by the amount of available memory. A widely available 1GB of memory space can easily store w=100 million points of size 10 bytes each. From an application perspective, a time-based window of *one day* over 10 streams, with a rate of 100 points/second per stream, contains the same number of points. To reduce the $O(w)$ space complexity of the classic algorithm in sliding window model, we first propose AVC, an approximation algorithm for the time series model which maintains only a sample of the streamed points using the similar sampling technique described for building radial histograms in [4] and maintaining convex hull of data streams

[1] See the related section in *SQR: A Stream Query Repository*, http://www-db.stanford.edu/stream/sqr/immersive.html.

in [10]. AVC employs the classic algorithm to build the Voronoi cell of the sample as an *approximation* to the exact cell. The AVC's time complexity of an update per-point is $O(\kappa)$ where κ is the sample size. The value of κ is determined based on a single parameter k that is independent of the distribution of the points. For the sliding window model, we propose an extension to AVC, AVC-SW, with the same time and space complexities. However, the sample size of AVC-SW is dependent on both the parameter of the algorithm and the distribution of the points in the stream. We show that for a uniform site point distribution, the average size of the sample maintained by AVC-SW is far less than the window size w, specially for large w's. Even though our simple experiments show 80% reduction in memory usage by AVC-SW as compared to the classic algorithm (Section 7), we prove that this reduction is even more when $w \gg k$. For a uniform point distribution, we theoretically compute the sample size of AVC-SW in terms of the window size and its single parameter. We show that the sample size grows very slowly as the ratio of window size w over k increases. For instance, it is less than $20k$ for $w/k \leq 2.5 \times 10^8$.

Our main contribution resides in our analytical proofs and experimental results. We study the properties of our approximation algorithms (Section 5) and compute their approximation error for a general nearest neighbor search through detail analysis. More importantly, we theoretically prove that one can determine the single parameter of AVC (and AVC-SW) based on user's tolerable error ε (Section 6). We show that both AVCs compute an ε-approximation to the Voronoi cell in $O(\kappa)$ per-point computation time and with $O(\kappa)$ memory where κ is the size of their sample. Both AVCs can be incorporated within a general histogram-like sampling framework [4]. Moreover, the size of their approximate Voronoi cell is independent from the complexity of the actual cell and the distributions of the points.

2 Definitions

The Voronoi cell of a point p derived from a given set of points $N \subset \Re^2$ is a *unique convex* polygon which includes all the points in the space $\Re^2$ that are closer to p than to the other points in the set N. Each edge of the polygon is a part of the perpendicular bisector line of the line segment connecting p to one of the points in the set. We call each of these edges a *Voronoi edge* and each of its end-points (vertices of the polygon) a *Voronoi vertex* of p. For each Voronoi edge of the point p, we refer to the corresponding point in N as a *Voronoi neighbor* of p. Furthermore, the set N is usually called the set of *site* points. As an example, Figure 1 shows the Voronoi cell of a point p generated given the set $N = \{n_1, ..., n_4\}$ as a quadrilateral. The points n_1 and v_1, and the edge $\overline{v_1 v_2}$ are the corresponding Voronoi neighbor, vertex and edge of p, respectively. The general definition of the Voronoi cell of a point in the d-dimensional space $\Re^d$ follows:

Definition 1. *If p is a d-dimensional point, N is a set of n points in the d-dimensional space $\Re^d$, and $D(.,.)$ is a distance metric defined in the space, then $V(p)$, the Voronoi cell of the point p given set N, is defined as the* ***unique convex***

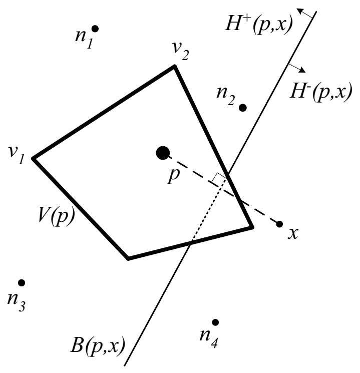


Fig. 1. The Voronoi cell of the point p with respect to $N = \{n_1, ..., n_4\}$. The bisector line corresponding to the new point x intersects with $V(p)$ and excludes all the points in $H^{-}(p,x)$ from $V(p)$.

polygon *which contains all the points in the set* $V_n(p)$ [2]*:*

$$V_n(p) = \{q \in \Re^d \mid \forall\ n \in N, D(q,p) < D(q,n)\}$$

Throughout this paper, we assume that the points are in 2-dimensional space and the distance metric is Euclidian. We use $|pq|$, $\overline{pq}$, and $B(p,q)$ to denote the Euclidian distance between the points p and q, the line segment connecting them, and the perpendicular bisector line of this segment, respectively. We use $\upsilon(p)$ to refer to the set of Voronoi neighbors of p. It is clear that the Voronoi cell $V(p)$ can be computed using $\upsilon(p)$ in $O(1)$ time and vice versa. Hence, we use $V(p)$ and $\upsilon(p)$ interchangeably throughout the paper.

3 The Problem

Assume that we need to compute $V(p)$, the Voronoi cell of a given fixed point p with respect to a set of n site points N. For each point $q \in N$, the line $B(p,q)$ divides the space into two half-planes: $H^{+}(p,q)$ including p and $H^{-}(p,q)$ including q. For any point in $H^{-}(p,q)$, we say that $B(p,q)$ *excludes* the point from $V(p)$ (see Figure 1). The trivial way to find $V(p)$ is to find the common intersection of all n half-planes $H^{+}(p,x)$ for all points $x \in N$. $V(p)$ is the boundary of the intersection. If we use linear programming, the intersection is computed in $O(n \log n)$ time and linear storage [5].

In this paper, we study two variations of the problem. First, consider the case when the points in N become incrementally available as a data stream. That is, at each time instance t, only one point n_t arrives, and we update N to $N \cup \{n_t\}$. The point p could be the fixed location of an immobile sensor node in the motivating example of Section 1. The node incrementally receives the locations of the other nodes (i.e., points in N) through a data stream. Here, the update scheme of N is broadly referred as *time series model* in the data stream literature [12]. Now the problem is to update $V(p)$ (or $\upsilon(p)$) according to the updates to N (i.e, when

[2] We assume that such a bounded polygon exists. Furthemore, we assume that $p \notin N$. While this convention is different from the literature on Voronoi diagrams, the result is the same.

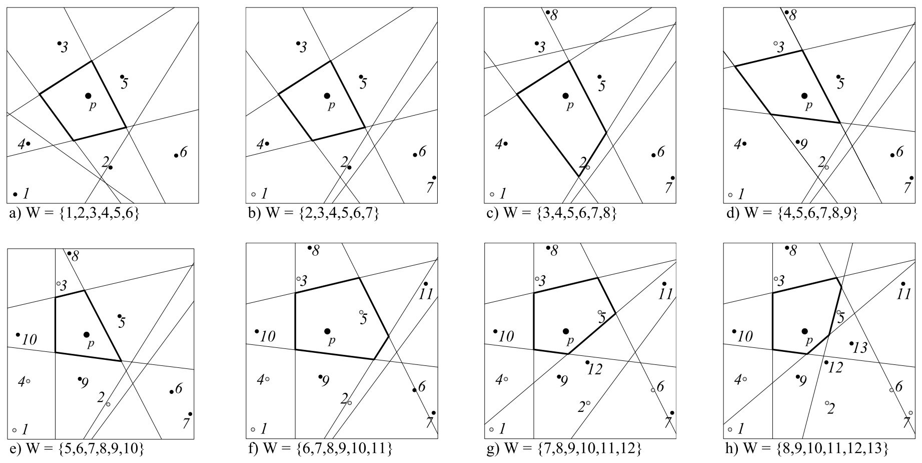


Fig. 2. The Voronoi cell of point p over a sliding window W of size 6 for eight subsequent time instances. The label of the each point shows its arrival order.

a new point x arrives). The classic traditional solution is to find the intersection of $B(p, x)$ with Voronoi edges of p (i.e., edges of $V(p)$) and update $V(p)$ to the intersection of $V(p)$ and $H^+(p, x)$ (see Figure 1).

The time complexity of each update is $O(|\upsilon(p)|)$ where $|\upsilon(p)|$ is the number of current Voronoi neighbors of p. This complexity is $O(|N|)$ for the worst case where any point in N is a Voronoi neighbor of p. Meanwhile, it is $O(1)$ on average as the average number of vertices of a Voronoi cell is less than six [5]. Therefore, the approach meets the common requirement in data stream algorithms which dictates that the complexity of an update per-point must be sub-linear in time. Moreover, the space complexity of the solution is also $O(|\upsilon(p)|)$. The reason is that if $B(p, x)$ does not intersect with the current $V(p)$, we do not need to store the point x. Hence, we only store the Voronoi neighbors of p at each update time and drop the other points.

Even though the optimal algorithm to compute the exact Voronoi cell is cost effective, its complexity is mainly dependent on the distribution of the site points. The number of Voronoi neighbors of the point p depends on the position of the points in the stream. Therefore, the amount of space required to store the cell is not deterministic. This is critical in applications such as sensor networks with memory and power limitations.

Now consider the *sliding window* case when we are only interested in w most recent points. Here, the goal is to maintain the Voronoi cell of the set of points arrived so far in a window W of fixed size. To illustrate, Figure 2 shows the Voronoi cell of a point p over a window of size 6 for eight subsequent time instances. Each point is labelled by its arrival order (or time). The points shown as filled dots are within the current window (i.e., the set W) while empty dots show the others. Each figure snapshot shows only the bisector lines of these points. In Figure 2a, when the point 6 arrives, its corresponding bisector does not intersect with the cell and hence it is not a Voronoi neighbor of p. However, later in Figures 2c and 2f, the point 6 does become a Voronoi neighbor of p. On the other hand, the point

7 never become a Voronoi neighbor of p during any of the time instances when the point 7 is in the current window (Figures 2b-2g). This example shows that the traditional algorithm cannot drop a new point (e.g., point 6) even though its corresponding bisector does not intersect currently with the cell. That is, the space complexity of the algorithm is $O(|W|)$ where $|W|$ is the size of the window [3]. Therefore, the traditional algorithm is not scalable as data stream rate and window size grow. Motivating by this observation, we first propose an algorithm to maintain an approximate Voronoi cell in the general time series model (Section 4). In Section 7, we extend our algorithm to be applicable over a sliding window.

4 The Approximate Voronoi Cell Algorithm (AVC)

We want to maintain an approximation to the Voronoi cell of the point p while the site points in N are arriving as a data stream. The core idea behind our AVC algorithm is to maintain a minimum subset of N including the closest site points to p and compute the Voronoi cell of p with respect to this subset instead of N. This cell is an approximation to the exact Voronoi cell given the entire N.

We divide the 2-d space using k vectors in k different directions. Each vector originates from the point p. Moreover, the angle between each pair of neighboring vectors is $\theta = 2\pi/k$. We will show in Section 6 that the value of k can be determined as a function of the user's tolerance for error. As Figure 3a shows, the vectors partition the space into k identical *sectors*. For each sector S_i, we store a point $m(S_i)$, the closest site point to the point p which is inside S_i. We refer to this point as the *minimum point* of the sector S_i. When a new point x arrives through the stream, first we find the sector S_i containing x. Then, we replace the minimum point of the sector ($m(S_i)$) with x if the point p is closer to the point x than to the point $m(S_i)$.

Now the Voronoi cell of p derived from the set of k minimum points corresponding to k sectors ($M_N = \bigcap_{1\leq i\leq k}\{m(S_i)\}$) is an approximation of the actual Voronoi cell of p using the set N. It is clear that the approximate Voronoi cell of p, contains its actual Voronoi cell. We can compute the approximation in $O(k \log k)$ time and space at any time using the classic algorithm from the scratch. By incrementally updating this approximation on new point arrivals, the per-point computation can be reduced to $O(k)$. Furthermore, the time complexity of the per-point sample update is also $O(k)$ which can be improved to $O(\log k)$ using a searchable data structure [10]. Hence, the per-point update time of AVC including the time for updating both the sample and the approximation is $O(k)$. Therefore, AVC maintains a sample of size $\kappa = k$ to improve in terms of both time and space complexity over the classic algorithm when $k < |v(p)|$.

Throughout this paper, we use AVC(θ) to denote our algorithm with parameter θ in $k = 2\pi/\theta$ specifying the number of sectors. Furthermore, we use $V'(p)$ to refer to AVC's approximation to the Voronoi cell of the point p. Figure 3b shows the exact Voronoi cell of p with respect to the set $N = \{a, b, c, d, e, f, g, h\}$. Figure 3c shows $V'(p)$ created by AVC($\theta = \pi/8$). The filled dots in the figure are minimum points of the sectors while the empty dots are dropped by AVC.

[3] In fact, deciding to drop a new point is significantly expensive for the traditional algorithm (i.e., $O(|W|^2)$). Details are removed due to the lack of space.

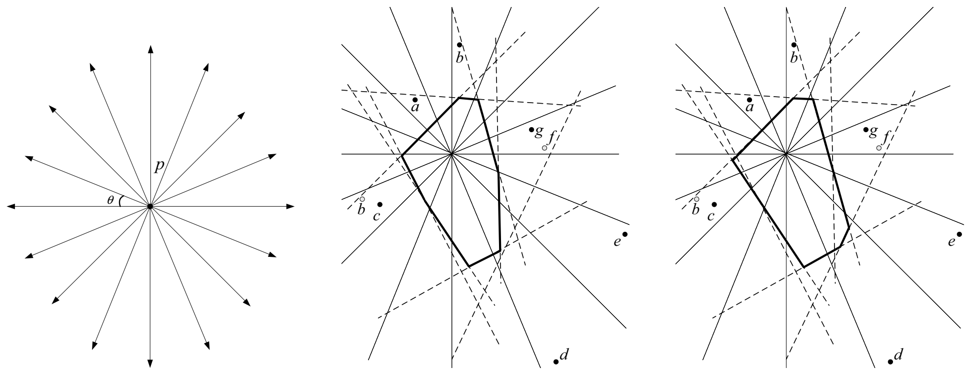


Fig. 3. a) $k = 16$ vectors originating from p divide the space into k identical sectors, b) The Voronoi cell of the point p, and c) The approximate Voronoi cell of p.

5 Properties of AVC

In this section, we study different properties of the approximate Voronoi cell computed by the AVC algorithm. We use these properties to compute the approximation error of the algorithm in terms of the parameter θ.

A primary property of the Voronoi cell of a point p is that it contains none of the site points[4]. We intend to maintain this property for the approximate Voronoi cell. It is trivial that the output of AVC(θ) depends on both the value of θ and the distribution of the site points in N. However, we show that specific values of θ can be used in AVC to make some properties of its output independent from the distribution of the input points.

Lemma 1. *For any point p, $V'(p)$ computed by AVC(θ) does not contain any of the site points in N for any arbitrary set N if and only if θ is less than $\pi/3$.*

Proof. See Appendix A.

Another property of the Voronoi cell of p is that the distance of any point inside (on) the cell to p is less than (equal to) its distance to any site point in N. We show that for any point inside (on) the approximate Voronoi cell of a point p, its distance to its closest site point in N is less than its distance to p by at most a small factor. We define the function $f(q)$ over the set of points q in 2-d space as

$$f(q) = \frac{|qp|}{|qr|} \tag{1}$$

where r is the closest site point to q in N. The property indicates that $f(q)$ is always less than or equal to 1 for the set of points inside or on $V(p)$. Over this set, the function f reaches its upper bound (one) on the points on the boundary of $V(p)$. To study the approximation error of the AVC algorithm, we need to find the upper bound of the function over the set of points inside or on $V'(p)$. In particular, if a point is inside or on $V'(p)$ we find how far its distance to p from its distance to its actual closest point could be. Towards this end, we first locate the points where the maximum of $f(q)$ occurs.

[4] Note that $p \notin N$.

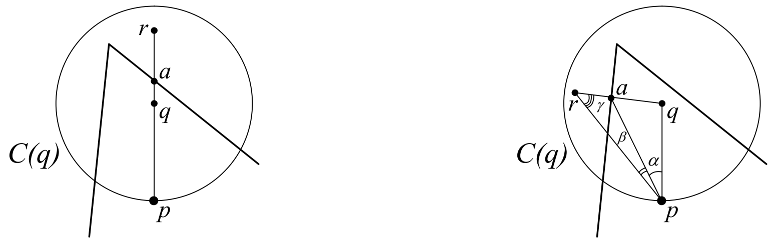


Fig. 4. The point q inside $V'(p)$ and its closest site point r where a) p, q, and r are collinear, and b) they form the triangle $\triangle qpr$.

Lemma 2. *Let q be a point inside or on $V'(p)$, computed by AVC(θ) for a point p, and r be its closest site point in N. If $\theta < \pi/3$, the maximum of $f(q) = \frac{|qp|}{|qr|}$ over all points q occurs for a point on the boundary of $V'(p)$.*

Proof. The proof is by contradiction. Assume that q with the maximum $f(q) = \frac{|qp|}{|qr|}$ is inside $V'(p)$. According to Lemma 1, as θ is less than $\pi/3$, the site point r is not inside $V'(p)$. Therefore, the line segment $\overline{qr}$ intersects with one of the edges of $V'(p)$ at a point a. First, we show that

$$\frac{|ap|}{|ar|} > \frac{|qp|}{|qr|}. \tag{2}$$

The points p, q, and r are either collinear or form a triangle. Figure 4a illustrates the first case. As q is between a and p, and a is between q and r, and all four points are on the same line, we have $|qp| < |ap|$ and $|qr| > |ar|$. Therefore, Equation 2 holds. Figure 4b shows the second case illustrating the triangle $\triangle qpr$. In the figure, we have $\angle qpa = \alpha$, $\angle apr = \beta$, and $\angle qrp = \gamma$. In the triangle $\triangle qpr$, the law of sines yields

$$f(q) = \frac{|qp|}{|qr|} = \frac{\sin\gamma}{\sin(\alpha+\beta)} \tag{3}$$

Meanwhile, using the same law in the triangle $\triangle apr$, we get

$$\frac{|ap|}{|ar|} = \frac{\sin\gamma}{\sin\beta} \tag{4}$$

As r excludes q from $V(p)$, r is inside the circle $C(q)$ centered at q with a radius of $|qp|$. Therefore, in the triangle $\triangle qpr$ we have $\alpha + \beta < \pi/2$. Comparing $\alpha + \beta$ with β results in

$$\beta < \alpha + \beta < \frac{\pi}{2} \Rightarrow \sin\beta < \sin(\alpha+\beta) \tag{5}$$

Comparing Equations 3 and 4, and considering the inequality in Equation 5 shows that Equation 2 holds in the second case too. Let s be the closest point to a in N. We assumed that $|as| < |ar|$, therefore

$$f(a) = \frac{|ap|}{|as|} > \frac{|ap|}{|ar|} > \frac{|qp|}{|qr|} = f(q) \tag{6}$$

Equation 6 contradicts our assumption and shows that the point q with the maximum value for $f(q)$ must be on the boundary of $V'(p)$. □

6 Approximation Error

We prove that the Voronoi cell computed by the AVC algorithm is an ε-approximation to the actual Voronoi cell. More precisely, if a point q is inside the approximate Voronoi cell of a point p, its distance to its closest point in N is less than its distance to p by at most a factor of $1+\varepsilon$ (i.e., $f(q) \leq 1+\varepsilon$). We show that this difference is bounded and find the upper bound of ε for a given θ. Moreover, we prove that for a given ε, one can compute the largest θ for which AVC(θ) results in an approximation of tolerable error ε. To provide a proof, we first showed in Lemma 1 that the maximum of the function f occurs on the edges of the approximate Voronoi cell. In this section, for an arbitrary point q on the approximate Voronoi cell of p but outside its actual Voronoi cell, we consider the set of its possible closest points in N which might have been dropped by the AVC algorithm. We find the maximum of $f(q)$ over the set of all points such as q.

Theorem 1. *If q is a point on the boundary of $V'(p)$ computed by the algorithm AVC(θ) and r is its closest site point in N, a certain ε can be found in terms of θ which*

$$f(q) = \frac{|qp|}{|qr|} \leq 1+\varepsilon \tag{7}$$

Proof. Let q be a point on one of the edges of the approximate Voronoi cell of p and outside the actual Voronoi cell of p (i.e., inside $H^-(p,r)$). That is, the closest point to q in $N \cup \{p\}$ is a point r other than p. The goal is to find the maximum of $|qp|/|qr|$. Towards this objective, we need to find the minimum of $|qr|$ as $|qp|$ is fixed for q. Hence, we locate the closest such a point r to q. It is clear that this point is not among k minimum points corresponding to the sectors (i.e., $r \notin M_N$). The reason is that if it was one of these points, the bisector line corresponding to $\overline{pr}$, $B(p,r)$, would have excluded q from the approximate Voronoi cell of p in the AVC algorithm. However, an exact Voronoi cell computation algorithm causes q to be outside $V(p)$. The locus of the points such as r which their corresponding bisector lines, $B(p,r)$, exclude the point q from $V(p)$ is inside a circle centered at q with a radius of $|pq|$. We call this circle $C(q)$. To illustrate, consider the Voronoi cell of point p showed in Figure 5a. The figure shows a point q inside the Voronoi cell of p and the points x and y inside and outside the circle $C(q)$, respectively. The bisector line $B(p,x)$ intersects with the Voronoi cell causing q to be excluded from the cell. This is while, y which is outside the circle $C(q)$ has no effect on the inclusion of q in the cell.

Now consider all the sectors which intersect with the circle $C(q)$, namely $S_1, ..., S_9$ in Figure 5b. As q is on $V'(p)$ we can infer that for each of these sectors either it includes none of the points in N or its corresponding minimum point is outside the circle $C(q)$. However, because q is outside $V(p)$ the former case cannot be true for all of such sectors. That is, there must be at least one sector containing r with a minimum point outside the circle $C(q)$. This minimum point has caused the point r to be removed from our set of minimum points. Therefore, its corresponding bisector line has not excluded q from $V'(p)$ in the AVC algorithm. To find the minimum value of $|qr|$, we show how this happens and find the closest point r which has been removed because of this minimum point.

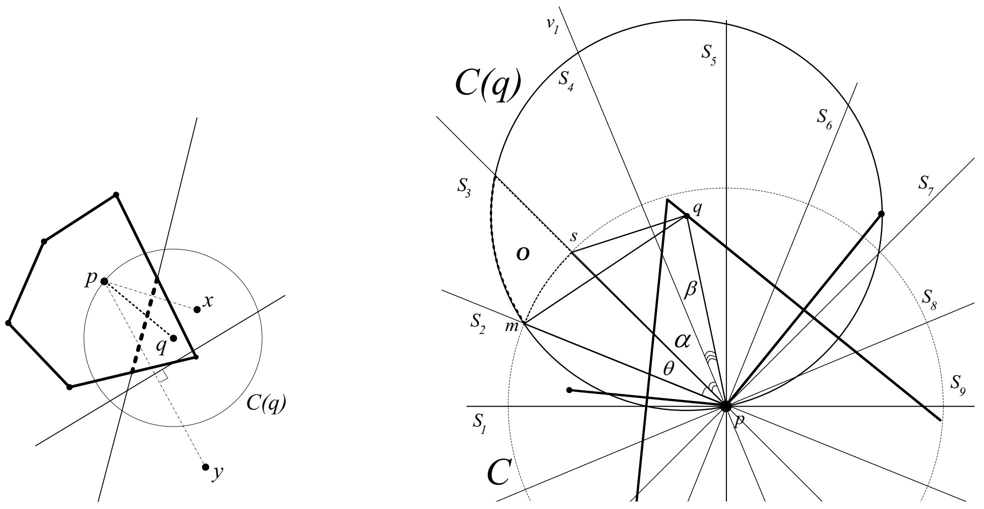


Fig. 5. a) The effect of the two points x and y on the inclusion of q in the Voronoi cell of p. b) The point q on one of the edges of $V'(p)$, the intersection of the sectors with the circle $C(q)$, the hidden point of the sector S_3, the angle β based on the location of q in S_5, and the angle α based on the location of S_3 and S_5.

Consider one of the sectors that intersect with $C(q)$ (S_3 in Figure 5b). Let us locate the intersection point of the boundaries of the sector and the circumference of $C(q)$ which is closer to p (m in Figure 5b). The point m is the closest possible minimum point to p inside S_3 and outside $C(q)$. Each of the points inside the sector and the circle $C(q)$ whose distance to p is more than $|mp|$ has been removed because of m. The locus of these points is the intersection of 1) inside the sector S_3, 2) inside the circle $C(q)$, and 3) outside the circle centered at p with a radius of $|mp|$ (C). Figure 5b marks this intersection as O. The closest point to q in O is the point s. We refer to s as the *hidden* point of the sector S_3 with respect to q and $V'(p)$.

Now consider the hidden point of S_3. Let v_1 be the closest vector to q between $\overline{pq}$ and $\overline{ps}$. We use α and β to refer to the angles which $\overline{ps}$ and $\overline{pq}$ make with v_1, respectively (see Figure 5b). Notice that the location of q determines the value of β while α depends on the location of S_3 with reference to the sector containing q (i.e., S_5). Using the law of cosines in the triangle $\triangle pqs$ we have

$$|qs|^2 = |qp|^2 + |ps|^2 - 2 \cdot |qp| \cdot |ps| \cdot \cos(\alpha + \beta) \tag{8}$$

We have $|ps| = |pm|$ as both s and m reside on the circle C. Moreover, the triangle $\triangle pqm$ is an isosceles triangle as both points p and m are on the circle $C(q)$ ($|qm| = |qp|$). The angle between $\overline{pm}$ and $\overline{ps}$ is equal to θ as they are boundaries of the same sector. Now in the triangle $\triangle pqm$ we have $|pm| = 2 \cdot |qp| \cdot \cos(\alpha+\beta+\theta)$. If we replace $|ps|$ with the value of $|pm|$ in Equation 8, we get

$$|qs|^2 = |qp|^2 + 4 \cdot |qp|^2 \cdot \cos^2(\alpha + \beta + \theta) - 4 \cdot |qp|^2 \cdot \cos(\alpha + \beta) \cdot \cos(\alpha + \beta + \theta) \tag{9}$$

We define $F(\alpha, \beta, \theta)$ as

$$F(\alpha, \beta, \theta) = \sqrt{1 + 4 \cdot \cos^2(\alpha + \beta + \theta) - 4 \cdot \cos(\alpha + \beta) \cdot \cos(\alpha + \beta + \theta)} \tag{10}$$

and replace it in Equation 9 to get

$$|qs| = |qp| \cdot F(\alpha + \beta + \theta) \tag{11}$$

As r is the closest point to q, the point q is closer to the hidden point of the sector containing r (i.e., s inside S_3 in Figure 5b) than to the hidden points of all of the sectors that intersect with the circle $C(q)$. To be precise, for a given point q (with a fixed angle β) the value of $|qs|$ in the sector containing r must be minimum over all other sectors (with different values of α) that intersect with the circle $C(q)$. To exploit the domain of the angle α, observe that as both points m and p are on the circle $C(q)$, the angle $\angle qpm = \alpha + \beta + \theta$ is not greater than $\pi/2$. Therefore we have

$$|qs| = \min_{\substack{0 \leq \alpha \leq \frac{\pi}{2} - \beta - \theta \\ \alpha = i\theta}} (|qp| \cdot F(\alpha + \beta + \theta)) \tag{12}$$

We replace the value of $|qs|$ in $f(q) = |qp|/|qs|$ to obtain

$$\frac{|qp|}{|qs|} = \frac{1}{\min_{\alpha = i\theta} F(\alpha + \beta + \theta)} \tag{13}$$

Now as $|qr| \geq |qs|$ for the sector containing r, we have

$$f(q) = \frac{|qp|}{|qr|} \leq \frac{|qp|}{|qs|} \tag{14}$$

Therefore, the upper bound of $f(q)$ is not greater than the upper bound of $|qp|/|qs|$. The latter is the maximum of Equation 13 over different points q (with different β angles). Hence, for any point such as q

$$f(q) \leq \max_{0 \leq \beta \leq \theta} \left(\frac{1}{\min_{\alpha = i\theta} F(\alpha + \beta + \theta)} \right) \tag{15}$$

Equation 15 shows an upper bound for $f(q)$ as a function of α, β, and θ. In the remainder of the proof, we show that this function is bounded by another function in terms of θ. Let $\alpha_0 = argmin(|qs|)$. Figure 6a plots α_0/θ for different values of $k = 2\pi/\theta$ and β, and Figure 6b shows a slice of this diagram for $k = 36$. As shown in both figures, the value of α_0 that determines the location of the point s depends on both θ and β. It means that if q (and β) changes in Figure 5b, the sector containing r might change to be S_4 (not S_3). In general, for a given θ the polar angle between the sector containing a point inside $V'(p)$ and its closest possible point which can be removed by AVC is not fixed.

Figure 6c plots the maximum of $f(q)$ for different values of θ. As the figure shows, $f(q)$ is less than 2 if k is greater than 16 (i.e., $\theta < \pi/8$). For any value of θ,

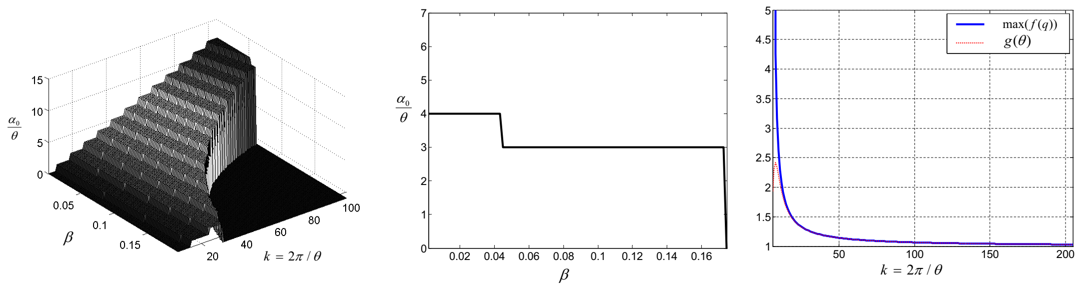


Fig. 6. a) α_0/θ shows the location of the sector containing the closest hidden point to q for different values of β and k (β is in radians), b) α_0/θ for different values of β where $k = 36$ (i.e., $\theta = \pi/18$), c) The upper bound of $f(q)$ for different values of k.

one can extract a certain $\varepsilon = max(f(q)) - 1$ from the Figure 6c so that Equation 7 holds. To find a maximum function in terms of θ we plotted the following function in this diagram:

$$g(\theta) = \frac{1}{\sqrt{3 - 2\sqrt{2} \cdot \cos(\pi/4 - \theta)}} \tag{16}$$

The figure shows that for the values of $k > 24$, $g(\theta = 2\pi/k)$ is always greater than but very close to $f(q)$. Consequently, the approximation error of AVC(θ) is bounded by $g(\theta)$. It implies that Equation 7 holds for $\varepsilon = g(\theta) - 1$. $\square$

The immediate implication of the proof of Theorem 1 is that for a given ε, we can use $g(\theta)$ to compute the smallest k (i.e., the largest θ) to use with the AVC algorithm and result in an approximation of tolerable error ε.

Theorem 2. *For a given error bound ε, the largest θ, using which AVC(θ) can compute an approximate Voronoi cell with a maximum error of ε is computed from the equation $g(\theta) = 1 + \varepsilon$.*

7 AVC over Sliding Windows

In this section, we extend the AVC algorithm to be applicable to the sliding window model. With this model, the goal is to maintain the Voronoi cell of a point p with respect to the set of w recent points. With a window of fixed size w, when the new point n_t is arrived through the data stream, we update the set of site points N to exclude its oldest point and include n_t. We say the oldest point has been *expired*.

In Section 2, we showed that a traditional Voronoi cell computation algorithm cannot be used over a sliding window. The algorithm is not scalable to data stream rate and window size as it costs $O(w)$ memory. Likewise, the AVC algorithm is prone to the same problems. As an example, assume that AVC stores the minimum point m corresponding to the sector S at time t. Assume that according to the window size, m expires at time $t' > t$ (i.e., when we receive w points after t). Consider the set of site points that arrive during the time range (t, t') and reside in the sector S. If p is closer to m than to any of these points, AVC drops all of them and maintains m as the minimum point of S. However, as soon as m expires, the minimum point of S needs to be updated to the point m', the closest point to

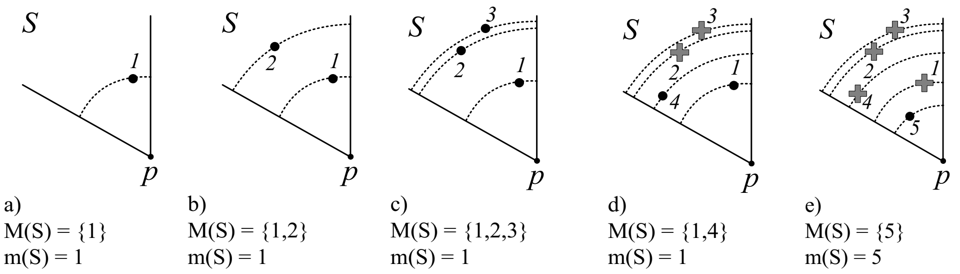


Fig. 7. AVC-SW updates the set $M(S)$ and the minimum point $m(S)$ for each of the 5 arriving points in the sector S. Assume that none of these points expire during this illustration.

p in N_S. Hence, AVC must store m'. Applying the reasoning recursively renders that AVC must store all the points in the window. In the remainder of this section, we extend the AVC algorithm to overcome this shortcoming by storing only the points which might become a minimum point in a future window.

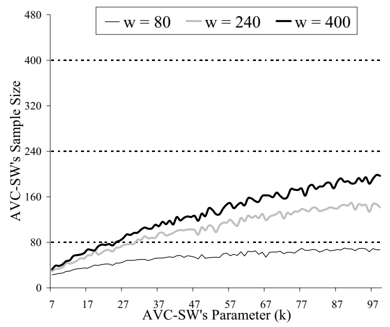
With the AVC algorithm, we require to maintain the minimum points of each sector for the current window. The main idea behind our extension (AVC-SW) is that for each sector, any point arrived earlier than the sector's current closest point to p could not be minimum in any of the future windows. Therefore, we can drop this point. Otherwise, we store it as it might be the minimum point in a future window before it expires.

7.1 The AVC-SW Algorithm

Let w be the window size, and at each time instance t, only one point arrives. We employ the same vectors of AVC to divide the space into sectors (see Section 4). For each sector S_i, we store the minimum point $m(S_i)$ and a set of points $M(S_i)$. This set includes all the points which are likely to be minimum points in the future windows. We initialize $m(S_i)$ and $M(S_i)$ to null for all sectors S_i before we start processing the data stream.

For each new point x, first we find the expired point among members of all $M(S_i)$ sets and delete it. Second, we find the sector S containing x and add x to $M(S)$. Third, we delete any point y in $M(S)$ if $|py| > |px|$. These points will never become minimum point of their sector in a future window. Finally, we set $m(S)$ to the closest point to p in $M(S)$. Similar to AVC, the Voronoi cell of p derived from the set of k minimum points ($m(S_i)$) corresponding to k sectors is the approximation of the actual Voronoi cell of p. The properties of $V'(p)$ and the approximation error analysis discussed in Sections 5 and 6 also hold for the output of AVC-SW. The sample size of AVC-SW is computed as $\kappa = \sum_{i=1}^{k} |M(S_i)|$, where $|M(S_i)|$ is the cardinality of the set $M(S_i)$.

Figure 7 illustrates how AVC-SW maintains the minimum points of the sector S. The figure shows only the times when the new point is inside the sector S. Assume that none of these points expires during these time instances. As shown in Figure 7a, the point 1 is the current minimum point of S. When the point 2 and 3 arrive in Figures 7b and 7c, respectively, AVC-SW adds them to $M(S)$ as they might become the minimum point of S when 1 expires. However, 1 is still the



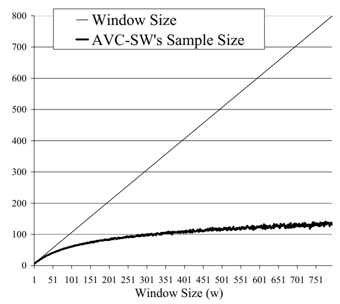


Fig. 8. a) Average number of points stored by AVC-SW (i.e., AVC-SW's sample size) for a) $w = 80$, 240, and 400 and different values of k, and b) different values of w when $k = 36$.

current minimum point of S in the window. In Figure 7d, the point 4 arrives. As 4 is in all future windows in which 2 or 3 exist and p is closer to 4 than to 2 and 3, we delete 2 and 3. Finally, the point 5 arrives in Figure 7e and causes the update to the minimum point and deletion of all the points in $M(S)$.

In general case, the space requirements of AVC-SW is less than $O(w)$ as we drop the portion of the points that are unlikely to be a minimum point. However, in the worst case, when the points of each sector arrive in the increasing order of their distance to p, AVC-SW stores all of them (see Figures 7a-7c). We conducted an experiment to evaluate the average space used by AVC-SW. We synthetically generated data streams of 1000 points uniformly distributed inside a circle. We applied AVC-SW's sampling algorithm on the stream and computed the average number of stored points during 100 runs. Figure 8a illustrates the sample size (κ) of AVC-SW for different values of k when we vary the window size. It shows that for window sizes greater than k, the sample size is far less than w. Figure 8b shows up to 80% reduction in memory requirement for $k = 36$ and large windows.

7.2 Space Requirements Analysis

In this section, we theoretically find the average number of the points stored by AVC-SW. Let w and k be the window size and the number of sectors used by AVC-SW, respectively. For a uniform distribution of the points in the space, each sector includes w/k points. For now, we assume that $n = w/k$ is an integer. Assume that on average, AVC-SW stores $A(i)$ points of i points contained in each sector. In other words, $A(i)$ is the average size of the set $M(S_j)$ for the sector S_j. We compute $A(n)$, the average number of points in a window of size w which AVC-SW stores corresponding to each sector. Hence, the total number of points stored by AVC-SW will be $\kappa = k.A(n)$.

AVC-SW uses the order of the points received in each sector and their distance to the center point p to decide whether they must be stored or dropped. For a sector S, assume that i points of the current window are inside S. We show these points as an i-tuple $P = (p_1, p_2, \ldots, p_i)$. The point p_j is the j-th arrived point which is inside the sector S. We sort these points based on their increasing distance to the point p and rank them accordingly. To break the ties, for two points with the

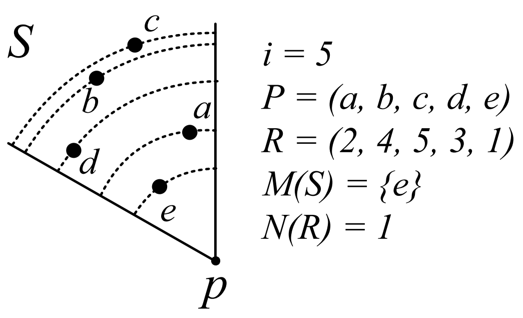


Fig. 9. .

same distance to p, we insert the point with greater polar angle with the x-axis after the one with the smaller angle. The i-tuple $R = (R_1, R_2, \ldots, R_i)$ includes the ranks in which $1 \leq R_j \leq i$ is the rank of the point p_j in P. As P includes no redundant point, R is clearly a permutation of positive integers less than or equal i. Depending on the order of the points in the i-tuple P and their distance to the center point p, R could be any of $i!$ permutations of these numbers. Assume that for the sector S including the points in P ranked as R, AVC-SW stores $N(R)$ points. To illustrate, Figure 9 shows the sector S including five points a, b, c, d, and e. The order of their arrival time is given as the 5-tuple P with a and e as the oldest and the most recent points, respectively. R shows their ranks considering their distances to p. AVC-SW stores only the point e so the value of $N(R)$ is one.

Let T_i be the set of all permutations of positive integers less than $i+1$. Moreover, let $S(i)$ be the sum of values of $N(R)$ for all permutations $R \in T_i$. In other words,

$$S(i) = \sum_{R \in T_i} N(R) \tag{17}$$

Therefore, out of i points inside each sector, the average number of points stored by AVC-SW will be

$$A(i) = \frac{S(i)}{i!} \tag{18}$$

It is clear that $A(1) = S(1) = 1$. We try to find $S(i)$ and $A(i)$ in terms of $S(i-1)$ and $A(i-1)$, respectively. First, notice how T_i is generated using the members of T_{i-1}. Inserting i before each number of a permutation in T_{i-1} or after its last number generates a unique permutation in T_i. For example, for the permutation $(2,1)$ in $T_2 = \{(1,2),(2,1)\}$, we can generate permutations $(3,2,1)$, $(2,3,1)$, and $(2,1,3)$ of T_3 by inserting 3 before 2, before 1 and after 1, respectively. In general, we generate i members of T_i from each permutation in T_{i-1}. We use $\Gamma_{R,i}$ to refer to the set of all permutations generated from R by the above approach. The above generation scheme dictates that $\Gamma_{R,i} \cap \Gamma_{R',i} = \emptyset$ where $R, R' \in T_{i-1}$ and $R \neq R'$. Furthermore, obviously we have $T_i = \bigcup_{R \in T_{i-1}} \Gamma_{R,i}$.

Now, we determine the value of $N(R')$ for each permutation $R' \in T_i$ generated from a permutation $R \in T_{i-1}$ (i.e., $R' \in \mathit{\Gamma}_{R,i}$). Given $R = (R_1, R_2, \ldots, R_{i-1})$, R' is generated by inserting i either before an R_j in R or exactly after R_{i-1}. In the first case, when we insert i before R_j to generate R', it means that the corresponding point q with rank i has arrived before the jth point (i.e., p_j). All R_k's in R are less than i. Recall that each R_j shows how close the jth arrived point of a sector is to the center point p; the smaller R_j the closer p_j to p. Hence, as we have $R_j < i$, the arrival of p_j causes deletion of the point q by AVC-SW. It means that in the first case adding q (i.e., adding i to R) does not change the number of points stored by AVC-SW (i.e., $N(R') = N(R)$). In the second case, when R' is $(R_1, R_2, \ldots, R_{i-1}, i)$, AVC-SW stores the point q with the rank i as it might become the minimum point of the sector in future windows. Subsequently, we have $N(R') = N(R) + 1$. Therefore, we get

$$\sum_{R' \in \mathit{\Gamma}_{R,i}} N(R') = i.N(R) + 1 \tag{19}$$

Using Equation 17 we have

$$\begin{aligned} S(i) &= \textstyle\sum_{R \in T_i} N(R) \\ &= \textstyle\sum_{R \in T_{i-1}} \sum_{R' \in \mathit{\Gamma}_{R,i}} N(R') \\ &= i. \textstyle\sum_{R \in T_{i-1}} N(R) + \sum_{R \in T_{i-1}} 1 \\ &= i.S(i-1) + (i-1)! \end{aligned} \tag{20}$$

Dividing both sides of this equation by $i!$ and considering the definition of $A(i)$ given in Equation 18, we get the following for $i > 1$:

$$A(i) = A(i-1) + \frac{1}{i} \tag{21}$$

As we have $A(1) = 1$, rewriting Equation 21 yields

$$A(i) = \sum_{j=1}^{i} \frac{1}{i} \tag{22}$$

The above is the partial sum of the first i terms of the harmonic series. The sum $A(i)$ is given analytically by the ith harmonic number H_i:

$$H_i = \gamma + \psi_0(i+1) \tag{23}$$

where γ is the Euler-Mascheroni constant and $\psi_0(x)$ is the digamma function. Although the series diverges when i increases, H_n has proven to be increasing slowly. For $i \leq 2.5 \times 10^8$ terms, it is still less than 20. Furthermore, to get H_n greater than 100, 1.509×10^{43} terms of the series is needed [7].

So far, we have computed the average number of points of each sector stored by AVC-SW. Therefore, if the number of points in each sector $n = w/k$ is an integer, AVC-SW stores only $\kappa = k.H_n$ points out of w points. If w/k is not an integer, we set $n = \lfloor w/k \rfloor$ and $m = w \bmod k$. For a uniform distribution of points in a window of size w, $k - m$ sectors include n points and m sectors

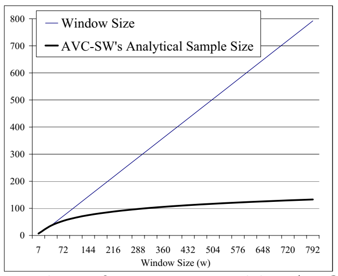


Fig. 10. Analytical average number of points stored by AVC-SW (i.e., AVC-SW's sample size κ) for different values of w when $k = 36$.

include $n + 1$ points. Therefore, the sample size of AVC-SW for window size w is $\kappa = (k - m).H_n + m.H_{n+1}$. As we have $H_{n+1} = H_n + 1/(n + 1)$, we simplify κ to get the following for the general case:

$$\kappa = k.H_n + \frac{m}{n+1} \tag{24}$$

Considering the above equation, the sample size κ is less than $20k$ when $w \leq 2.5 \times 10^8 \times k$. Figure 10 shows the value of κ computed using Equation 24 for $k = 36$ sectors and different window sizes. Notice that the number of sectors is the same as that of results shown in Figure 8b. Comparing two figures proves that the analytical sample size computed as Equation 24 completely supports the sample size computed during our experimental results.

8 Related Work

The field of computational geometry (CG) is a rich area of investigation for the data stream algorithms [12]. Recently, a few studies have been focused on CG problems on data streams. In [6], Feigenbaum et al. find the diameter and convex hull of a geometric data stream over a sliding window. Their sample uses $O(r)$ space with $O(r)$ and $O(\log r)$ processing time per point to maintain a $(1+O(1/r^2))$-approximation and $(1+O(1/r))$-approximation, respectively. In [4], Cormode et al. use the same sampling method as AVCs to build radial histograms and approximate a number of geometric aggregates such as diameter and furthest neighbor search on 2-d point data streams. In [10], Hershberger et al. introduce adaptive sampling to maintain an approximate convex hull of geometric data streams with a distance of $O(D/r^2)$ from their accurate convex hull, where D is the diameter of their sample set. To the best of our knowledge, no study have considered building Voronoi-related data structures on geometric data streams.

Different variations of Voronoi diagrams have been used as index structures for the nearest neighbor search. In [8], Hagedoorn introduces a directed acyclic graph based on Voronoi diagrams. He uses the data structure to answer exact nearest-neighbor queries with respect to general distance functions in $O(\log^2 n)$ time using only $O(n)$ space. Stanoi et al. in [17] combine the properties of Voronoi cells (influence sets in their terminology) with the efficiency of R-trees to retrieve reverse nearest neighbors of a query point from the database. As a more practical example, Kolahdouzan et al. [11] propose a Voronoi-based data structure to improve the performance of exact k-nearest neighbor search in spatial network databases.

Arya et al. [1] focus on approximating the Voronoi diagrams globally to answer ε-nearest neighbor queries. They build cells with the shape of hypercubes or the difference of two hypercubes. Har-Peled [9] partitions the space with an approximation of Voronoi diagrams. His space decomposition generates a compressed quadtree of size $O(n\frac{\log n}{\varepsilon^d}\log\frac{n}{\varepsilon})$ that answers ε-nearest neighbor queries in $O(\log(n/\varepsilon))$ time. Arya et al. [2] have performed the only work on approximating Voronoi cells in d-dimensional space. Their approach combines the shape approximation and adaptive sampling techniques to build an approximate cell of size $O(1/\sqrt{\varepsilon})$ for d=2. They assume that the accurate cell to be approximated is given. Then, they examine the Voronoi neighbors of the given point and the corresponding Voronoi vertices to keep the minimum number of Voronoi neighbors using which an ε-approximate cell for addressing nearest neighbor problem can be computed. This approach is not applicable to sliding windows over data streams as insertion/deletion of each single point might cause the sampling criteria to include or exclude a neighbor from the cell. This non-deterministic change results into storing all the points in the window.

9 Conclusion

We proposed AVC and AVC-SW algorithms for approximating a Voronoi cell on geometric point streams in time series and sliding window models, respectively. We theoretically computed the approximation error of our algorithms in terms of their single parameter. Our main findings are as follows:

- AVCs result ε-approximations to the Voronoi cell.
- Using Theorem 2, the parameter k (or θ) of AVCs can be computed based on the user's tolerable error.
- With the sliding window model, AVC-SW significantly improves the space complexity of the classic algorithm when the window size is greater than its parameter k.

References

1. S. Arya, T. Malamatos, and D. M. Mount. Space-efficient approximate voronoi diagrams. In *Proceedings of the 34th ACM Symp. on Theory of Computing (STOC)*, pages 721–730, 2002.
2. S. Arya and A. Vigneron. Approximating a voronoi cell. Technical report, 2003. HKUST-TCSC-2003-10.
3. P. Bonnet, J. E. Gehrke, and P. Seshadri. Towards Sensor Database Systems. In *Proceedings of the Second International Conference on Mobile Data Management*, pages 3–14, 2001.
4. G. Cormode and S. Muthukrishnan. Radial histograms for spatial streams. DIMACS TR 2003-11, 2003.
5. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer Verlag, 2nd edition, 2000.
6. J. Feigenbaum, S. Kannan, and J. Zhang. Computing diameter in the streaming and sliding-window models, 2002. Manuscript.
7. M. Gardner. *The Sixth Book of Mathematical Games from Scientific American*. University of Chicago Press, 1984.

8. M. Hagedoorn. Nearest neighbors can be found efficiently if the dimension is small relative to the input size. In *Proceedings of the 9th International Conference on Database Theory - ICDT 2003*, volume 2572 of *Lecture Notes in Computer Science*, pages 440–454. Springer, January 2003.
9. S. Har-Peled. A replacement for voronoi diagrams of near linear size. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 94–103, 2001.
10. J. Hershberger and S. Suri. Adaptive sampling for geometric problems over data streams. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems.* ACM, 2004.
11. M. R. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB'04)*, 2004.
12. S. Muthukrishnan. Data streams: Algorithms and applications. Technical report, Computer Science Department, Rutgers University, 2003.
13. A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellations, Concepts and Applications of Voronoi Diagrams.* John Wiley and Sons Ltd., 2nd edition, 2000.
14. H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS.* Addison-Wesley, 1990.
15. C. Shahabi. AIMS: An immersidata management system. In *Proceedings of the first Conference on Innovative Data Systems Research (CIDR'03)*, January 2003.
16. M. Sharifzadeh and C. Shahabi. Supporting spatial aggregation in sensor network databases. In *Proceedings of the 12th ACM International Symposium on Advances in Geographic Information Systems - ACM GIS'04*, 2004. To Appear.
17. I. Stanoi, M. Riedewald, D. Agrawal, and A. E. Abbadi. Discovery of Influence Sets in Frequently Updated Databases. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 99–108. Morgan Kaufmann Publishers Inc., 2001.

Appendix

A The Proof of Lemma 1

Proof. First, assume that $\theta < \pi/3$ and the point $q \in N$ is inside $V'(p)$. The proof is by contradiction. Let S be the sector containing q. It is clear that q cannot be the minimum point of S as $V'(p)$ is the Voronoi cell of p with respect to the minimum points in N. Hence, suppose m is the minimum point corresponding to S (i.e., $m = m(S)$) as shown in Figure 11a. Therefore, we have $|pq| > |pm|$. As q and m are both inside the same sector S, the angle between $\overline{pm}$ and $\overline{pq}$, $\alpha = \angle qpm$, is less than θ. Therefore, $\alpha < \pi/3$. In the triangle $\triangle pqm$, $|pq| > |pm|$ concludes that $\gamma > \beta$. Moreover, as α, β, and γ are the angles of the same triangle, thus $\alpha + \beta + \gamma = \pi$. At least one of the angles β or γ must be greater than $\pi/3$ as $\alpha < \pi/3$. The fact that $\gamma > \beta$ yields that $\gamma > \pi/3 > \alpha$. That is, in the triangle $\triangle pqm$, we have $|pq| > |mq|$ and hence, q is closer to m than p. Therefore, the bisector of $\overline{pm}$, $B(p, m)$, would exclude q from $V'(p)$. This means that $V'(p)$ does not contain the point q which contradicts our assumption.

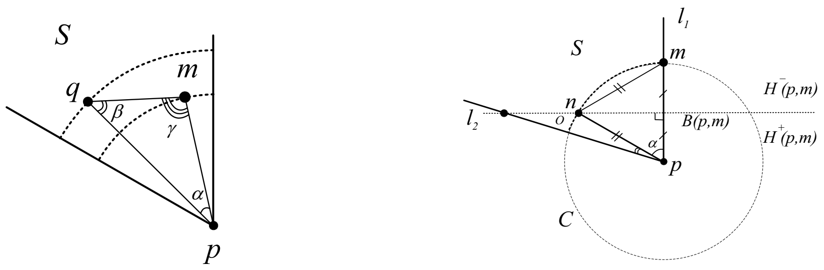


Fig. 11. The sector S and its minimum point m where a) $\theta < \pi/3$, and b) $\theta > \pi/3$

Now assume that the angle θ is greater than $\pi/3$. We show that there exist a set of potential points in N which are not excluded from $V'(p)$ by any of bisector lines corresponding to the minimum points. Figure 11b shows a sector S and its boundaries l_1 and l_2. Assume that the corresponding minimum point of S, m, is on l_1. The locus of all the points in the sector S which are removed from N because of the minimum point m is outside the circle C centered at p with a radius of $|pm|$. The bisector line $B(p, m)$ intersects with the circle C at point n. We show that this point is inside S. As n is on $B(p, m)$ we have $|mn| = |pn|$. Besides, as n is on the circle C, thus $|pn| = |pm|$. Therefore, the triangle $\triangle pmn$ is an equilateral triangle. Hence, the angle $\alpha = \angle mpn = \pi/3$. This yields that $\alpha < \theta$. Therefore, the point n is inside the sector S. Now consider the intersection of the sector S, outside of the circle C, and $H^+(p, m)$ (marked as O in Figure 11b). As n is inside S, this intersection is not empty. The AVC algorithm removes any point in this intersection because of the minimum point m. However, the bisector line $B(m, p)$ does not exclude these points from the approximate Voronoi cell of p. For any point q in this area, if none of bisector lines corresponding to the minimum points of other sectors excludes q from $V'(p)$, q will be inside $V'(p)$.[5] □

[5] When $\theta = \pi/3$, n is the only point in O which reside on the edge of $V'(p)$ if no other bisector line excludes it from $V'(p)$.