Geoinformatica
DOI 10.1007/s10707-007-0024-1

ERRATUM

Utilizing Voronoi Cells of Location Data Streams for Accurate Computation of Aggregate Functions in Sensor Networks

Mehdi Sharifzadeh · Cyrus Shahabi

Received: 22 March 2005 / Accepted: 25 June 2005

Abstract Sensor networks are unattended deeply distributed systems whose database schema can be conceptualized using the relational model. Aggregation queries on the data sampled at each sensor node are the main means to extract the abstract characteristics of the surrounding environment. However, the non-uniform distribution of the sensor nodes in the environment leads to inaccurate results generated by the aggregation queries. In this paper, we introduce "spatial aggregations" that take into consideration the spatial location of each measurement generated by the sensor nodes. We propose the use of spatial interpolation methods derived from the fields of spatial statistics and computational geometry to answer spatial aggregations. In particular, we study Spatial Moving Average (SMA), Voronoi Diagram and Triangulated Irregular Network (TIN). Investigating these methods for answering spatial average queries, we show that the average value on the data samples weighted by the area of the Voronoi cell of the corresponding sensor node, provides the best precision. Consequently, we introduce an algorithm to compute and maintain the accurate Voronoi cell at each sensor node while the location of the others arrive on data stream. We also propose AVC-SW, a novel algorithm to approximate this Voronoi cell over a sliding window that supports dynamism in the sensor network. To demonstrate the performance of in-network implementation of our aggregation operators, we have developed prototypes of two different approaches to distributed spatial aggregate processing.

Keywords sensor networks · aggregation · spatial interpolation · Voronoi cell · spatial data stream

The online version of the original article can be found at: doi: 10.1007/s10707-005-4884-y. This paper, originally published in GeoInformatica, volume 10, number 1, March 2006, contained several uncorrected typographical errors. The errors occurred in the online version of this paper as well. The correct version of this paper is published here in its entirety.

M. Sharifzadeh (✉) · C. Shahabi
Computer Science Department, University of Southern California, Los Angeles, CA 90089-0781, USA
e-mail: sharifza@usc.edu

C. Shahabi
e-mail: shahabi@usc.edu

1 Introduction

Sensor networks are wireless networks of low power, memory constrained, autonomous, cheap and compact sensor nodes. These unattended nodes communicate within limited radio ranges. They are designed to be usually abandoned in unexplored inaccessible environments in which failure is a common event. The main idea behind deploying a sensor network in a physical environment is to monitor a real-world phenomenon. In other words, the measurement values monitored by each sensor node are data samples representing the phenomenon. The user studies these measurements to understand the underlying physical process in the environment. This process can be a wild fire or development of a specific population of marine microorganisms in nature.

Considering the data generated by a sensor network as a database, a broad range of queries is feasible. The class of traditional aggregation queries, in particular, are of main interest within the sensor network community [2], [5], [7], [16]. The common basis of all aggregation queries is to compute a summary value based on a set of database items. When applied on the sensor network data, the aggregation queries are the main means to extract the abstract characteristics of the phenomenon from the samples taken in the environment. For example, the following query is a range aggregate query on a network of sensor nodes with humidity and temperature sensors:

```
SELECT AVG(humidity)
FROM sensors
WHERE location INSIDE Room-101
AND temperature < threshold;
```

In general, no assumption should be made about the distribution of the sensor nodes in the environment. Hence, applying traditional average on a non-uniformly distributed observation set leads to erroneous results and false reasoning about the phenomenon. That is, as the phenomenon is usually a *continuous* process, only a uniform sample set is a good representative of the whole process. Furthermore, traditional aggregation operators are not amenable to outlier and noisy values. To illustrate, the average of the humidity monitored by the sensor network in Fig. 1 at times a and b is 10. With a continuously increasing humidity from left to right, the real average humidity at time a should be 15. While at time b, the real average is expected to be close to 5 since sensor node s_4 happens to be close to a local maxima in the space of humidity values.

While the research on data engineering for sensor networks has been focusing on providing efficient in-network query processing of traditional aggregation operators, our example showed that these operators are not reliable means to observe and study a real-world phenomenon. Instead, we propose to use the *spatial interpolation* methods at each node to infer about the missing values at neighboring unmonitored locations. We introduce the use of spatial aggregation on the data generated by sensor networks. With our spatial

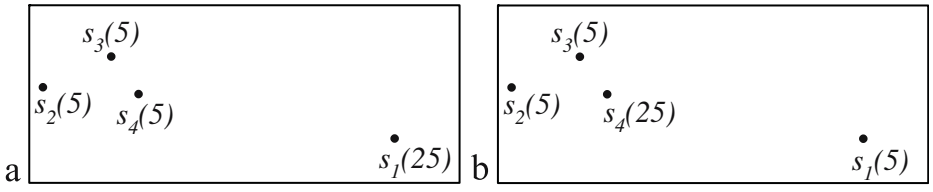

Fig. 1 Two snapshots of a non-uniformly distributed sensor network at times a and b

aggregation operators, the sensor measurements of sparse areas (e.g., s_1 in Fig. 1) contribute more to the final result as compared to those of dense areas. That is, the spatial average operator relies on the value of the node s_1 more than that of s_4 with two other nearby measurements. As of traditional aggregate processing for sensor networks, the spatial aggregation operators should be processed within the network considering power and memory constraints of the nodes.

Two classes of spatial interpolation methods have been proposed in the fields of spatial statistics and computational geometry. The global methods use the knowledge about the entire set of observations to interpolate any of the missing values. Applying these methods at each sensor node with limited local information requires high communication overhead. Instead, with the local methods, only information about a limited neighborhood is needed. Implementing the local methods at a sensor node is still challenging as the communication range of a node may not be sufficient to generate the required local neighborhood. In addition, with more accurate methods, the extent of this neighborhood is dependent on the locations of the neighboring observation points (i.e., nodes) which are not fixed.

Throughout this paper, we focus on processing spatial average on the data generated by the sensor nodes. We show the application of the classical spatial interpolation methods to formalize the spatial average operator as a weighted average over sensor measurements. In particular, we study three different methods. The first method, Spatial Moving Average (SMA), divides the deployment field of a sensor network into equal sized grid cells. SMA assigns a value to each cell based on the values of its contained nodes. With SMA, a spatial average is transformed into a traditional average on the values assigned to the grid cells. The second method, the Voronoi method, uses the Voronoi diagram of the node locations to partition the deployment field into convex polygons (i.e., Voronoi cells). The weight assigned by the method to each node is the area of its corresponding Voronoi cell. Finally, the Triangulated Irregular Network (TIN) which is originally an elevation model for visualizing geospatial data uses the Delaunay triangulation of the field. It assigns to each triangle the average value of the nodes located on its vertices.

We propose two approaches to in-network processing of the spatial aggregation queries. With the semi-distributed approach, a single node computes the weights and assigns them to the other nodes using the location data sent by each node. The distributed approach leaves the task of weight computation to individual nodes. Both approaches can be implemented on top of any traditional aggregation scheme in the sensor networks. Our experimental results show that the overall communication overhead of the semi-distributed approach is always 25% more than that of the distributed approach. This significant result saves a lot of energy in the network nodes.

To exploit the low communication cost of the distributed aggregation processing, we propose a distributed *local* Voronoi cell computation algorithm for sensor networks. We develop this algorithm to be used by each node for computing Voronoi and TIN weights. Our Voronoi cell algorithm can be utilized to generate the Voronoi cell of a query point using a stream of point data. At each sensor node, the algorithm iteratively uses the stream of locations it receives from the other nodes to construct its own local Voronoi cell.

In a real-world scenario, the sensor nodes frequently fail and stop generating measurement values. That is, we require that the contribution of any neighboring node to the Voronoi cell end after some period of time (i.e., when the node dies). This is called the *sliding window* model in Muthukrishan [8]. In this case, to compute its exact Voronoi cell, the node requires to store the location of *all* nodes arrived so far (i.e., $O(w)$ space complexity for w currently live nodes). To reduce the $O(w)$ space complexity of Voronoi algorithms in the sliding window model, we propose AVC-SW, an approximation algorithm

which maintains only a sample of the node locations. The core idea behind AVC-SW is to maintain a minimum subset of locations including the closest ones to node p in each direction and compute the Voronoi cell of p with respect to this subset instead of the set of all locations. While the theoretical proofs are out of scope of this paper, we show that AVC-SW computes a $(1+\varepsilon)$-approximation to the actual exact Voronoi cell. For a uniform point distribution, we can theoretically compute the average sample size of AVC-SW (i.e., its required memory) in terms of the window size w and its single parameter. We show that the sample size is $O(\log w/k)$ [11]. To the best of our knowledge, this is the first distributed algorithm for computing the approximate Voronoi cell over a sliding window on a data stream. It is completely applicable to any network of mobile nodes as well.

Finally, we conducted several experiments to evaluate the accuracy of the spatial average operators implemented using each of our studied interpolation methods. This is the first work on comparing these spatial aggregation operators in the context of sensor networks. Our experimental results show that the Voronoi-based operator outperforms the other two operators in accuracy. Moreover, considering communication cost, our distributed Voronoi cell computation algorithm outperforms the semidistributed approach using classical algorithms [9].

A preliminary version of this work is presented in Sharifzadeh and Shahabi [12], where Voronoi aggregation operator uses the exact Voronoi cell at each node. This paper subsumes [12] and proposes AVC-SW that supports dynamism in the sensor network. AVC-SW is a novel algorithm for computing the Voronoi cell of a fixed 2-d point when the locations of the neighboring points arrive as a geometric data stream. The remainder of the paper is organized as follows. Section 2 reviews the current research on aggregation in sensor networks, spatial interpolation and Voronoi cell computation. Section 3 defines spatial aggregation on sensor networks using classical spatial interpolation methods. In Section 4, we describe semi-distributed and distributed approaches to spatial aggregation processing. We propose our exact and approximate algorithms for Voronoi cell computation in Section 5. Section 6 includes our experimental results, and Section 7 discusses the conclusion.

2 Related work

While the database community has proposed many approaches for aggregate processing in database systems, these works focus on efficient processing of the traditional aggregate operators. Literature on spatial databases also mainly focus on modelling issues, the class of nearest neighbor queries and index structures to provide fast answers to the queries [4]. Meanwhile, the research work on aggregate processing in wireless sensor networks is orthogonal to our work on spatial aggregate processing. We can utilize any energy-efficient average aggregate processing algorithm to process our spatial average operators.

Intanagonwiwat et al. in [5] discuss direct difusion, a set of novel techniques to data-centric routing through the network. Their proposed operators are used to provide energy-efficient in-network data aggregation. Our spatial query processing scheme is capable to use the established aggregation path in their opportunistic and greedy approaches [5].

Madden et al. in [7] proposed TAG, an aggregation service as a part of TinyDB[1] which is a query processing system for a network of Berkeley motes. The service employs a SQL interface to the sensor data streams. It presents in-network processing of the aggregation

[1] http://telegraph.cs.berkeley.edu/tinydb/.

queries on the data generated in the sensor network. We use the adhoc query routing algorithm of TAG to disseminate our query into the network. Our spatial aggregate operators are compatible with the aggregate processing of TAG and easily portable to TinyDB.

Zhao et al. in [16] introduce an architecture for sensor network monitoring. Their architecture benefits from an energy-efficient aggregate processing for network properties (digest functions). An average query is computed on the digest tree which their digest diffusion scheme constructs. This digest tree can also be used as a query tree to route a spatial average query through the network. Similar to TAG, they process an average query through computing and forwarding partial results at the nodes on the digest tree.

The Voronoi diagrams have been extensively studied in the field of Computational Geometry [9]. However, the focus has been on the efficient computation and representation of these diagrams for an entire set of points. The only research works on computing Voronoi *cell* of a point is the two approaches presented in Stanoi et al. [13] and Zhang et al. [15]. Stanoi et al. [13] study the problem of finding the influence set (Voronoi cell) of a query point considering data points stored in the database. They show how this problem can be reduced into nearest neighbor and range queries. They propose an algorithm to extract a superset of all the data points which can potentially contribute to the Voronoi cell of the query point. However, they define a rectangular range that includes this superset as they intend to use an R*-tree index to retrieve the set. Our approach is similar in determining how a local cell can be affected by the recently received locations at each sensor node. However, we use a radial range to be compatible with the communication style of the nodes. Zhang et al. [15] also propose an approach to compute the cell (validity region in their terminology) when the data points are indexed by an R-tree. Their ray shooting scheme benefits from time parameterized nearest neighbor queries using an R-tree. However, our assumption is that a sensor node with a limited memory space needs to compute the cell while the location of all the data points are not available at the time of computation.

Arya et al. [1] 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 exact 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 is the same as what we call a $(1+\varepsilon)$-approximate cell in this paper. 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 nondeterministic change results into storing all the points in the window.

3 Aggregation on sensor networks

A sensor network SN consists of n sensor nodes deployed over an area R. Each sensor node s is equipped with different types of sensory devices. In general, the sensor nodes are not of the same type. They are monitoring different types of values depending on their assigned task within the network. However, to simplify our discussion and without loss of generality, we assume that all sensor nodes in our sensor network SN are operationally equal. Each sensor in node s continuously samples a value v measuring the phenomenon being mon-

itored (e.g., a temperature sensor samples the temperature at the location of the sensor node). Without loss of generality, we assume that all sensors are sampling the corresponding values at the same time intervals. Each sensor node has m sensors generating corresponding values $v_1,...,v_m$. Let the unique fixed location of the sensor node s in the 2-d space, (x, y), be known to the sensor node. Therefore, the sensor node s generates a data tuple d of the form $<x, y, t, v_1,...,v_m>$ at each time-stamp t. We conceptualize the set of data tuples generated by all n sensor nodes in SN as a relational table T. Each row of the table is uniquely identified using the location and time columns as the primary keys.

A set of different aggregation queries are now formally definable on the the realized conceptual model of the sensor network. The average query in Section 1 can be generalized as follows:

```
SELECT aggr(exp(attrs))
FROM T
[WHERE (x, y) INSIDE R
[AND pred]]
```

As in traditional databases, the above query applies the aggregation operator *aggr* on non-spatial attributes *attrs* of a *discrete* set of data items which are qualified according to the predicate clause *pred* and their location attribute being inside region R.

A spatial aggregation query on the same set of data is defined using the same SQL statement. This means that the interface of the spatial aggregation query is the same as that of traditional aggregation query but the functionality is totally different. Considering the fact that the data items are discrete observations representing a phenomenon in the space, the spatial operator computes the summary value for all the data items which can be extracted from the continuous space of the data items according to the query predicate/ constraints. These data items need not to be monitored at a specific location (i.e., stored in our conceptual database). For example, in the context of sensor networks, we only have access to those data items which are measurements of an actual sensor node located at a position in the space. A spatial aggregation operator needs to *infer* about the data items at the positions in the space R where there are no sensor nodes. It means that the query processing for the aggregation operator needs to apply an interpolation method to find the missing values.

Although the functionality of a spatial aggregation is different from that of its corresponding traditional aggregation, it may result in the same answer value. To be specific, the results of applying operators such as `MIN` (or `MAX`) to a discrete set of observations and the same set including interpolated data items is not different. The rationale behind this is that these characteristics of the set are invariants during the interpolation procedure. However, `AVERAGE` and `SUM` operators are different comparing to their corresponding traditional operators. The reason is that they take to the consideration the distribution of the data samples. Finally, `COUNT` operator is an example of the aggregation operators which does not have a meaningful corresponding spatial operator. Here, throughout this paper, we focus on spatial average query as a basic instance of the class of spatial aggregation queries.

Assume that the aggregation operator will be applied only on one attribute (i.e., one column of table T) and the predicate clause *pred* is omitted without loss of generality. To be specific, consider the sensor network in Fig. 2a. Each sensor node s_i located at (x_i, y_i) generates a value v_{it} at time t resulting a tuple of the form $<x_i, y_i, t, v_{it}>$. The traditional average on values v_{it} at time t_x is formulated as $\sum_{i=1}^{n} v_{it_x}/n$. Let the function $v(x, y)$ give the value of each location (x, y). Therefore, values v_{it} are samples of this function at specific

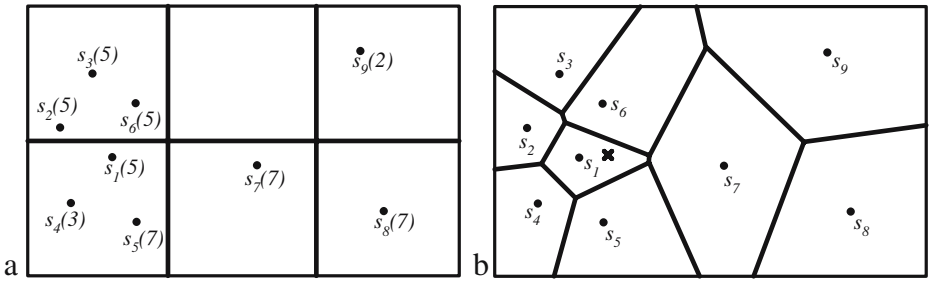

Fig. 2 **a**) A snapshot of a sensor network consisted of 9 sensor nodes with their sampled values, and **b**) its Voronoi diagram interpolating the value of the location × to the value of sensor node s_1

locations. The spatial average of the function $v(x, y)$ over the continuous region R is formally defined as follows:

$$SAVG = \frac{1}{area(R)} \cdot \int_R v(x,y)dxdy \tag{1}$$

Spatial interpolation is the main mean to provide values for unmonitored locations. "*Spatial interpolation is the procedure of estimating the values of properties at unsampled sites within an area covered by existing observations*" [6]. Different spatial interpolation methods have been proposed for environmental datasets, discrete observations at some locations in the environment. The field of spatial statistics categorizes these methods into global and local groups based on the set of observations they use to interpolate missing values. The group of global methods apply a single function to the whole set of observations in the space. *Trend surface analysis* is an example of the methods in this group. The local methods instead apply a common function repeatedly to subsets of the observed points. These methods such as Spatial Moving Average (SMA) usually generate the interpolated data as a set of local results.

Considering the need for in-network implementation of the aggregations on sensor networks [7], it is clear that global interpolation methods which require the whole set of sensor node measurements are not practical. In the remainder of this section, we briefly describe several local interpolation methods, namely, Spatial Moving Average (SMA), Voronoi Diagram, Triangulated Irregular Network (TIN), and Kriging. We will formally define how each one of our investigated methods calculate the spatial average as a weighted average on the individual sensor values.

3.1 Spatial moving average (SMA)

Spatial Moving Average method is widely used in different fields such as GIS and image processing. SMA divides the space using equal size grid cells. The value assigned to each location in the grid cell is then defined as a weighted average of the value of all observation points inside the cell. The corresponding weight of each value is $1/d$ where d is its distance from the center of the grid cell. For example, SMA assigns $(5/2 + 3/1 + 7/2)/(1/2 + 1 + 1/2) = 4.5$ to the grid cell including s_1, s_4 and s_5 in Fig. 2a.

The spatial average is defined in terms of the average on the values assigned to grid cells. Let $S_\square$ be the set of all grid cells partitioning the region R. The value assigned to a

grid cell δ including all sensor nodes s_i with values v_{it} is $v_\delta = \frac{\sum_{s_i \in \delta} v_{it} \times \frac{1}{d_i}}{\sum_{s_i \in \delta} \frac{1}{d_i}}$. Based on the SMA method, the spatial average is computed as follows:

$$SMA\_Avg = \frac{\sum\limits_{\delta \in S_\square} v_\delta}{\sum\limits_{\delta \in S_\square} 1} \tag{2}$$

We reformulate the above average into a weighted average on the sensor values. Let $G(s_i)$ be the the grid cell including the sensor node s_i. We rewrite Equation 2 as follows:

$$SMA\_Avg = \frac{\sum\limits_{i=1}^{n} \left(\frac{1}{d_i \times \sum_{s_j \in G(s_i)} \frac{1}{d_j}} \right) \times v_{it}}{\sum\limits_{i=1}^{n} \left(\frac{1}{d_i \times \sum_{s_j \in G(s_i)} \frac{1}{d_j}} \right)} \tag{3}$$

The above equation shows that for each point s_i an SMA weight w_i can be defined in the SMA method.

3.2 Voronoi diagram

The use of Voronoi diagram which is sometimes termed as Thiesson polygons is based on Tobler's first law of geography [14]. The law says: "*everything is related to everything else, but nearby things are more related than distant things*." This fact implies *spatial autocorrelation* for the observations in a geographic space. It means that there is a relation between values monitored at the neighboring locations.

This method assigns to a location with an unknown value the value of its closest location in the observation set. Therefore, there is always a unique value interpolated for each location. In other words, the space is partitioned with n sensor nodes (i.e., observation points) into n convex polygons (Voronoi cells) such that each polygon contains exactly one sensor node and every other location in a given polygon is closer to its corresponding sensor node than to any other node. The value of each location is interpolated to the value at its closest sensor node. To illustrate, in Fig. 2b the value at location shown as × is the same as that of s_1 (i.e., 5).

Considering this interpolation scheme, the spatial average over a continuous region R is translated into a weighted average on the values of all sensor nodes in R. The Voronoi weight assigned to each sensor node value is the area of its corresponding Voronoi cell. Formally speaking, let V_{si} and $area\,(V_{si})$ be the corresponding Voronoi cell of the sensor node s_i and its area, respectively. Using the Voronoi diagram method, the spatial average on values v_{it} generated by n sensor nodes s_i at time t is measured as follows:

$$Voronoi\_Avg = \frac{\sum\limits_{i=1}^{n} area(V_{s_i}) \times v_{it}}{\sum\limits_{i=1}^{n} area(V_{s_i})} \tag{4}$$

The averaging function in Equation 4 is amenable to nonuniform distribution of known values (i.e., sensor nodes) over the space. Each known value in the space is a representative for the value of a subset of locations in the space with an unknown value. In a sensor network implementation, each sensor node participates in the average value with a weight proportional to the area of the corresponding Voronoi cell.

3.3 Triangulated Irregular Network (TIN)

TIN is a vector-based method used as a digital elevation model. It is a method to generate a 3-dimensional model for the elevation data collected at a set of observation points in 2-d space. The method generates the model in two steps. First, all the observation points which are of the form (x, y, z) are projected to the xy plane. The Delaunay triangulation of the xy plane is created using the set of projected points. This is a unique partitioning of the space using triangles formed by neighboring points in the Voronoi diagram as their vertices. Then, for each triangle in the xy plane ($\Delta s_1 s_2 s_3$ in Fig. 3) the three observation points corresponding to its vertices are considered. Assuming that the points are not colinear, they define a unique 3-d plane. The projection of the triangle to this plane results a 3-d triangle ($\Delta s_1' s_2' s_3'$). Finally, the set of all 3-d triangles defined by the triangles in the Delaunay triangulation is a 3-d visualization of the observation data.

Although TIN is a visualization technique, it has also been used as a spatial interpolation method. Let the z value of each point be the value of the function $f(x, y)$ to be interpolated. To interpolate the value of a location (x, y), first it is located in the set of Delaunay triangles. Then, it is projected to the corresponding 3-d triangle of its surrounding Delaunay triangle. The z value of the projected point is the interpolated value of the location (x, y).

Figure 3 illustrates our example sensor network partitioned by Delaunay triangles. The figure shows how the method assigns a value to the point (x, y) located inside the triangle $\Delta s_1 s_2 s_3$. The triangle $\Delta s_1' s_2' s_3'$ is the corresponding 3-d triangle which is used to interpolate the value.

Let S_Δ be the set of all Delaunay triangles in the space. The average of the values over each triangle $\Delta s_1 s_2 s_3$ is $v_\delta = (v_{1t} + v_{2t} + v_{3t})/3$. Using the TIN interpolation, the spatial average of the values v_{it} generated by n sensor nodes s_i at time t is computed as follows:

$$TIN\_Avg = \frac{\sum_{\delta \in S_\Delta} area(\delta) \times v_\delta}{\sum_{\delta \in S_\Delta} area(\delta)} \tag{5}$$

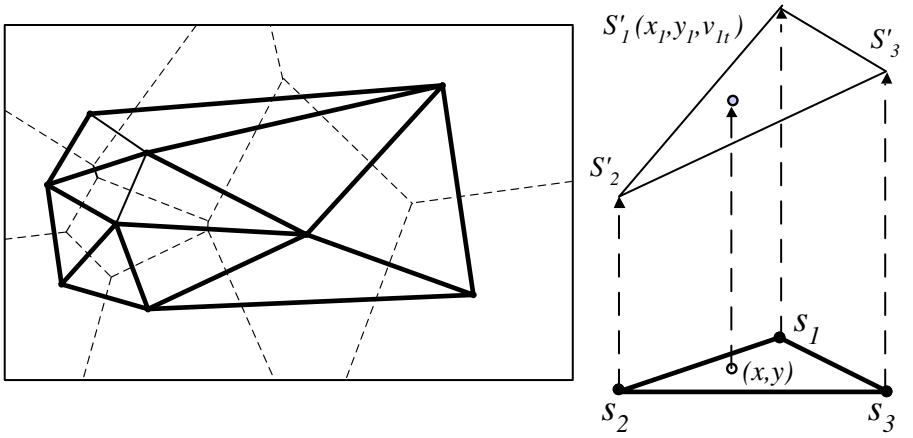

Fig. 3 The Delaunay triangulation of the sensor network in Fig. 2a, interpolating the value of the location (x, y)

As for the SMA method, we rewrite the above formula as a weighted average. Let $T(s_i)$ be the set of all Delaunay triangles in S_Δ with s_i as one vertex. It is clear that $S_\Delta = \cup_{i=1}^{n} T(s_i)$. Rewriting v_δ in terms of v_{it} in Equation 5 results in the following:

$$TIN\_Avg = \frac{\sum_{i=1}^{n} \left(\frac{1}{3} \times \sum_{\delta \in T(s_i)} area(\delta) \right) \times v_{it}}{\sum_{i=1}^{n} \frac{1}{3} \times \sum_{\delta \in T(s_i)} area(\delta)} \tag{6}$$

In Equation 6, each point s_i is assigned a corresponding weight w_i. Interpolation with TIN assigns values only to the locations inside the *convex hull* of the observation points. That is, it assigns no value to the locations which are not inside any of triangles. This shortcoming of the method is overcome by inserting virtual points on the boundaries of the space (e.g., the rectangle bounding the sensor network in Fig. 2a).

3.4 Kriging

Kriging [10] is a complicated interpolation technique developed in the field of geostatistics. The technique observes the underlying process in the space using representative variables (e.g., temperature) and computes unknown values of the variable using the values sampled in a limited set of locations. The interpolation method in Kriging is an optimization procedure which uses a model of the process to determine unknown values. This model is given as a variogram of the process. The method assigns optimal weights to the known values in order to predict the unknown values. As Kriging is a computationally intensive method, we did not include it in our prototype implementation.

4 Spatial average processing

We have formalized three different methods for applying spatial aggregation, specifically average, on a set of observation points. In this section, we briefly describe how this query is processed within the sensor network according to each method. Recall that each sensor node s_i located at (x_i, y_i) generates a value v_{it} at time t resulting a tuple of the form $<x_i, y_i, t, v_{it}>$. Let i be a unique id for the sensor s_i which can be easily generated using its location. For simplicity, we will refer to the most recent v_{it} as v_i.

A query is initiated when a user issues a spatial aggregation query from a *query node* which is a sensor node within the network or a base-station connected to the network. In general, she specifies the query using three parameters: the aggregation operator *aggr*, the attribute *attr* and the range *R*. In our model, the operator is *AVG* and the attribute is the single value that each sensor measures.

We use the ad-hoc query routing scheme used in Madden et al. [7] to disseminate the query. As the query is sent through the network, a routing tree is organized. Here is how the routing algorithm works. First, the query node broadcasts a message including the parameters of the query, its own id and its level in the query tree (i.e., 0 as it is the root node). Then, each sensor node s_j which receives the message including sensor id i and level l, selects the sensor node s_i as its parent, increments the level l and broadcasts its own id, j, and the updated level. At each level in the tree, the leaf nodes repeat this task to extend the tree up to the next level. In cases where a node with no assigned level receives more than one message, it chooses the one including the minimum id and ignores the others. The

algorithm terminates when all nodes have broadcast the "routing" message. At this time, each node in the network has been assigned a level in the tree.

The above algorithm builds a tree rooted at the query node which is used to send the results back from the nodes to the query node. In Section 3, we showed that all three methods can be formulated as the weighted averages of the values at each sensor node. That is, the average is of the form $\sum_{s_i \in R} w_i \times v_i / \sum_{s_i \in R} w_i$. Here we show how this result is generated within the network and is sent back to the query node. Assume for now that each node s_i knows its weight w_i. Each leaf node s_i sends $w_i \times v_i$ and w_i to its parent node. The parent node s_j maintains two variables representing the partial results: 1) WS_j, the weighted sum received from its children, and 2) W_j, the summation of the weights assigned to all of its children. The variables are initialized as $WS_j = w_j * v_j$ and $W_j = w_j$, respectively. On receiving the values from its children, the node updates these variables by adding them to their corresponding variables. It sends the variables to its parent after receiving all the data from its children. Each intermediate node in the tree repeats the above task. Finally, at the query node, s_k, the answer is delivered to the user as WS_k / W_k.

Now we discuss how the weights are computed and assigned. In all the three methods, the weight assigned to the value at each sensor node is computed based on the location of the neighboring nodes. We consider two approaches to compute the weights. With the *semi-distributed approach*, the query node computes the weights and delivers them to the corresponding nodes. While with the *distributed approach*, each node computes its own weight using the information about its neighborhood. In the following sections, we describe how each of the two approaches processes a query.

4.1 Semi-distributed query processing

The semi-distributed processing of the spatial average, consists of two phases. In the first phase, each sensor node sends its location to the query node as soon as the query tree is created. According to the aggregation method, the query node computes the weight of each sensor node. In the second phase, the query node sends the weights to the corresponding nodes over the query tree. Upon receiving the weights by the leaf nodes, they generate the partial results and send them back to their parent node as we discussed earlier in this section. We describe how the weights are assigned at the query node by each method.

4.1.1 SMA

The query node computes the SMA weights as follows. First, the query node divides the region R into grid cells using a cell size defined as an input parameter. Next, it finds the grid cell that contains each sensor node using its location. Finally, it computes the weight of the sensor node s_i using Equation 3.

4.1.2 Voronoi and TIN

As the Delaunay triangulation can be computed using the Voronoi diagram, the query node uses a single Voronoi construction algorithm on the node locations for both TIN and Voronoi methods. First, the query node applies Fortune's algorithm [3], [9] to compute the Voronoi diagram of the set of points (i.e., locations) received from the sensor nodes. Then, the area of the Voronoi cell of each point is computed as the Voronoi weight of the corresponding sensor node (see Equation 4).

For the TIN method, the query node finds the location of all the neighboring nodes in the Voronoi diagram for each node s_i. Subsequently, it considers all the triangles formed by s_i and two of its adjacent neighbors (i.e., the set of Delaunay triangles $T(s_i)$ in Section 3.3). Finally, it uses the area of such triangles to compute the weight of the node based on Equation 6.

The computed weight values will be sent to the nodes during the second phase. Notice that with this approach, we do not send the actual Voronoi cells or Delaunay triangles to the sensor nodes. Each node receives only its weight as a single numeric value.

4.2 Distributed query processing

With our distributed approach to the spatial average processing, each node is responsible for computing its own weight. The weight computation is interleaved with the query routing scheme which we described earlier in this section. We modify the ad-hoc query routing algorithm such that each node also incorporates its own location in the broadcasted message. All nodes which receive the message, use the included location in their weight computation.

4.2.1 SMA

We assume that the communication range of each node is large enough to cover its surrounding SMA grid cell. Therefore, the node receives the location of all the nodes in its grid cell during the query routing algorithm. It finds the distance of each of these nodes from the center of its grid cell and computes the SMA weight using Equation 3. As soon as it computes the weight and receives the partial results from its children, it sends back its own partial result to its parent.

4.2.2 Voronoi and TIN

In Section 4.1.2, we showed that the Voronoi weight and TIN weights of each sensor node are computed in terms of its Voronoi cell and the neighboring nodes which generate the cell, respectively. Unlike the semi-distributed approach, the node does not use a Voronoi diagram construction algorithm as it needs to compute only its own Voronoi cell. In Section 5, we propose an incremental algorithm to compute the Voronoi cell of a point using a set of neighboring points. Employing the algorithm, upon receiving all the broadcasted locations by each sensor node, it starts computing a local Voronoi cell. Then, it computes the area of its cell as its Voronoi weight. Moreover, as our algorithm returns the location of the neighboring nodes that generate the cell, the TIN weights can also be computed by each node.

The *local* Voronoi cell computed at each sensor node might not be the same as the *global* cell computed by the semi-distributed approach specially when the communication range of the node is small. We propose two approaches to resolve this situation. With the *local* approach, it suffices to use the area of the local cell as its weight. With the *global* approach the neighboring nodes exchange their sets of locations. When a node receives new locations, it uses our update procedure to *polish* the local cell (see Section 5). The sensor node terminates the polishing iteration as soon as it receives all partial results from its children or a local timer expires. At this time, the polished local cell is taken as a good approximation for the global cell. We report on the accuracy of both approaches in Section 6. The next section is dedicated to our algorithm for Voronoi cell computation taking place at each node.

5 Voronoi cell computation

The Voronoi cell of a center point p derived from a given set of points N is a *unique convex* polygon which includes all the points in the space that are closer to the center point p than other points of the set N. Each edge of the polygon is a part of the bisector line of the line segment connecting the center point to one of the points in the set. We call each of these edges a *contributor edge*, each of its end points (vertices of the polygon) a *contributor vertex* and its corresponding point in N a *contributor point* to the Voronoi cell. As an example, Fig. 4a shows the Voronoi cell of a point p generated given the set $N = \{n_1,..., n_4\}$. The points n_1, v_1 and the edge v_1–v_2 are the corresponding contributor point, vertex and edge, respectively. The formal definition of the Voronoi cell of a point follows:

Definition 1 If p is a d-dimensional point, N is a set of n points in the d-dimensional space $\mathbb{R}^d$ and $d(.,.)$ is a distance metric defined in the space, Vp, the Voronoi cell of point p given set N is defined as the *unique convex polytope* which contains all the points in the set V_{np}:

$$V_{np} = \left\{ q \in \mathbb{R}^d | \forall n_i \in N, d(q,p) < d(q,n_i) \right\}$$

Throughout the paper, we assume that the points are in 2-dimensional space and the distance metric is Euclidian. The problem of finding Voronoi cell of a point is the same as extracting the contributor points to the cell from the given point set. Finding Voronoi cells of *all the points* in the set (*Voronoi diagram*) is a classic computational geometry problem. Although this problem has an optimal solution with $O(n \log n)$ complexity [9], the solution in its original form is not applicable to our problem. The reason is that as a sweeping algorithm it is a global method which considers the whole set of all points.

In this section, we briefly describe our algorithm for finding the Voronoi cell of a certain point given the set of all neighboring points. We identify two different cases of the problem for each of which we propose a robust solution. The first case, termed the *global* case, is when the set N is a static set of points and is completely known at the time of computing the Voronoi cell. In this case, the set of contributor points to the Voronoi cell is a subset of the initial given set N. We define the second case, *local* case, as the situation in which the set N is dynamically updated. That is, the points in the set N will be added to or deleted from the

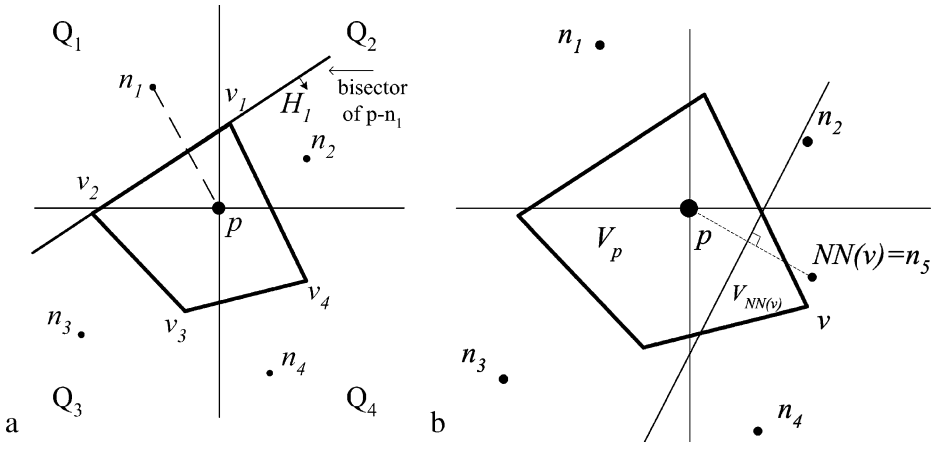

Fig. 4 **a**) The initial approximation to the Voronoi cell of point p. The polygon is the Voronoi cell of p generated using the points n_1,..., n_4. **b**) An iteration of the polishing step

set over the time. However, throughout this paper, we refer to the local case as a special case when points are only added to N. In the local case, the set N is not completely known and some of the member points are missing at the time of computation. The set is incrementally completed and each update operation to the set may triggers an update to the Voronoi cell of the point.

5.1 Global Voronoi cell

Assume that we are given a point p and a set of n points N. The trivial solution to the problem is to examine all n half planes formed by n bisector lines and generate the cell as their intersection. Considering the complexity of finding all pairwise intersections, this solution takes $O(n^2)$ in time. We propose an algorithm which starts with an approximation of the Voronoi cell and incrementally refines this approximation to generate the accurate cell. The algorithm consists of two main steps: approximation and polishing. We describe each of these steps in turn.

The approximation step strives to find a superset of the accurate Voronoi cell. This set which is a convex polygon has the useful property that it can be reduced into the Voronoi cell using a sequence of clipping operations defined by a set of bisector lines. We generate this approximation using a small subset of points in N. First, among all the points in N we find the four closest points to p, one in each quadrant formed by two perpendicular lines intersecting at p (See Fig. 4a). We name these four points n_1, ..., n_4. Second, for each n_i we draw the bisector line of the line segment p-n_i. This bisector line divides the space into two half planes. Consider the space defined by the half plane including p is termed H_i. Finally, the intersection of all half planes H_1, ...,H_4 will define a bounded polygon which is a triangle or a quadrilateral. The set of the points n_i which their half planes form this polygon and their corresponding edges of the polygon are initial *candidate* contributors to the Voronoi cell of p. We store these candidate contributor point-edge pairs as a representation of our initial approximation. Therefore, we can access each point using its corresponding edge (or vertex) and vice versa.

The reason we use points n_i as the initial candidate contributors is that more likely their corresponding half planes will generate a bounded polygon. But there are still cases when the intersection of half planes is an unbounded region in the space. In these cases we can randomly choose a point from N located in the unbounded region and close the region using its corresponding half plane.

The polishing step is an iterative step which tries to refine the approximate cell generated during the first step. In each iteration, we examine current candidate contributor vertices in turn and find the first vertex v which is closer to a point $NN(v)$ in N other than p. Then, we draw the bisector line of the line segment p–$NN(v)$. Considering the fact that the current approximate polygon is convex, it is clear that the line intersects the boundaries of the polygon in exactly two points v_1 and v_2. As Fig. 4b shows, the bisector line divides the polygon into two polygons (V_p and $V_{NN(v)}$). We identify V_p and $V_{NN(v)}$ as the polygons including points p and $NN(v)$, respectively. Finally, we update our current polygon to V_p by clipping the current approximate polygon with the bisector line. This update operation is easily applied by first excluding all old candidate contributor points and their corresponding edges/vertices which are in $V_{NN(v)}$. Then $NN(v)$ and two intersection points v_1 and v_2 are added as the new candidate contributor point and its corresponding vertices, respectively.

As the set N is fixed, the Voronoi cell is finalized when there is no candidate contributor vertex with a closest point other than p. As a result, all candidate points, edges and vertices are actual contributors to the Voronoi cell V_p. Consider the worst case which is when we

include each point in N as a candidate contributor point once during one of the iterations. Each non-contributor point will be excluded from the approximate cell during a different iteration and will never be included in the future. This means that the iteration in the polishing step eventually terminates. Figures 5 and 6 show the pseudo-code for our algorithm.

5.2 Local Voronoi cell

In a sensor network, the messages including the position of the neighboring nodes incur a delay proportional to their distance to the receiver node. Therefore, it might be the case that when a node needs to compute its Voronoi cell, it would not have enough information about its neighboring nodes (i.e., the set N). If the node applies our global Voronoi cell algorithm (Fig. 6) using the set N known at the time, the generated *local* Voronoi cell may not be the same as the actual *global* cell which would have been generated using the positions of all the neighboring nodes. Depending on the application, the sensor must decide whether to use the local Voronoi cell or to compute the global cell using the local one as an initial approximation. In this section we assume that the goal is to achieve the global cell.

A straightforward solution to the problem is to keep applying the iteration in Fig. 6 whenever new points are added to N. The iteration refines the local Voronoi cell computed using points in the old set N. The new update procedure which is similar to our global algorithm is illustrated in Fig. 7. To summarize, computing the global Voronoi cell of a point p consists of the following steps:

1. When the first neighboring set N_0 is known, compute V_p using *FindVoronoiCell* (p, N_0). Notice that in this step $N=N_0$.
2. On receiving N_i, an update to the neighboring set N at time i, apply *UpdateVoronoiCell* (p, V_p, N_i) to polish the current V_p. Notice that in this step $N = \cup N_j, 0 \leq j \leq i$.

Although the above procedure results in a global Voronoi cell for a given point, but the termination of the polishing step can not be guaranteed. This is because as a new set of points N_i is added to N, each single point in N_i is likely to be a contributor to the Voronoi cell. We now prove that the order in which a sensor node receives N_i, guarantees the convergence of the polishing step. On one hand, it is clear that if a point q is far enough from the center point p, it will not be a contributor to the Voronoi cell. On the other hand, the order in which a sensor node receives the location of the other sensor ndoes (i.e., points in N) is related to their distances from the sensor node location (point p); with a high probability,

```
1  ApproximateVoronoiCell( Point p, Neighbors N ) {
2    Divide the space into equal quadrants Q_1, ..., Q_4;
3    n_i = NN(p) in Q_i;
4    H_i = half plane including p defined by bisector line of p-n_i;
5    V_p = polygon bounded by H_1, ..., H_4;
6    Set the candidate contributor points;
7    Return V_p; }
```

Fig. 5 Initializing an approximation to the Voronoi cell

Fig. 6 Computing the global Voronoi cell

```
1   FindVoronoiCell( Point p, Neighbors N ) {
2     V_p = ApproximateVoronoiCell(p, N);
3     do {
4       V_v = V_p.vertices;
5       v = first vertex in V_v so that NN(v) ≠ p;
6       Clip V_p with bisector(p, NN(v));
7       V_p = portion of V_p including p;
8       Update candidate contributor points;
9     } while( v ≠ null );
10    Return V_p; }
```

closer points are accessible earlier than farther points. Let $max(N_i)$ be the farthest point from p in N_i. For each point p, if N_i and N_j are received at times i and j, respectively ($i<j$):

$$d(q,p) < d(max(N_i),p) \Rightarrow Pr(q \in N_i) > Pr(q \in N_j) \tag{7}$$

Assume that there is a minimal neighborhood region R around point p which includes all possible candidate contributor points. Now for each point in N_i, we are able to determine whether it can contribute to the Voronoi cell or not by testing if the region R contains the point. Meanwhile, Equation 7 verifies that if at least one of the points in the set N_i is not contained in the specified region R, the probability that the Voronoi cell generated by applying the procedure *UpdateVoronoiCell* on the current cell using N_i being changed by the points in the future N_j's ($j>i$) is decreasing.

Fig. 7 Updating a local Voronoi cell

```
1   UpdateVoronoiCell( Point p, VoronoiCell V_p,
                       AddedNeighbors N_i ) {
2     do {
3       V_v = V_p.vertices;
4       v = first vertex in V_v so that NN(v) ≠ p;
        // NN(v) is the closest point to v in N_i
5       Clip V_p with bisector(p, NN(v));
6       V_p = portion of V_p including p;
7       Update candidate contributor points;
8     } while( v ≠ null );
9     Return V_p; }
```

Fortunately such a neighborhood region exists and can be easily specified in terms of the current Voronoi cell. The work in Stanoi et al. [13] shows that for each vertex v on the current Voronoi cell of point p, the locus of the points which can exclude v from the cell is inside a circle centered at v with a radius of $d(v, p)$. To illustrate, consider the voronoi cell of point p showed in Fig. 8. The figure shows two new points q_1 and q_2 inside and outside the circle C_1, respectively. The bisector line of the line segment p–q_1 intersects with the Voronoi cell causing the vertex v_1 being excluded from the cell. This is while, q_2 which is outside the circle C_1 has no effect on the the cell.

We rephrase the results from Stanoi [13] to adapt them to our problem. Let C_i be a circle with radius $d(v_i, p)$ centered at a point v_i on the Voronoi cell V_p of point p generated using points in set N. Assume that q is a new point.

Lemma 1 *Updating V_p using point q will exclude v_i from V_p if and only if C_i contains q.*

Lemma 1 reveals a direct conclusion which follows:

Lemma 2 *The Voronoi cell of point p generated using the set $N \cup \{q\}$ is equal to V_p generated using N if and only if for each vertex v_i on V_p, C_i does not contain q.*

The set of circles C_i defines the minimal neighborhood region R including all the points that their presence in N causes changes to the current Voronoi cell of p. Considering the fact that each sensor communicates with the other sensors in a radial range, we relax the region R to a circle C centered at p with a radius of $2\times max(d(p, v_i))$. It is obvious that C is the smallest circle including all circles C_i. This range is used by the sensor node at location p to filter the points received in each set N_i.

In the context of sensor networks, there are cases where the node p should initialize its local Voronoi cell but it does not know the locations of all the nodes in C. That is, the node

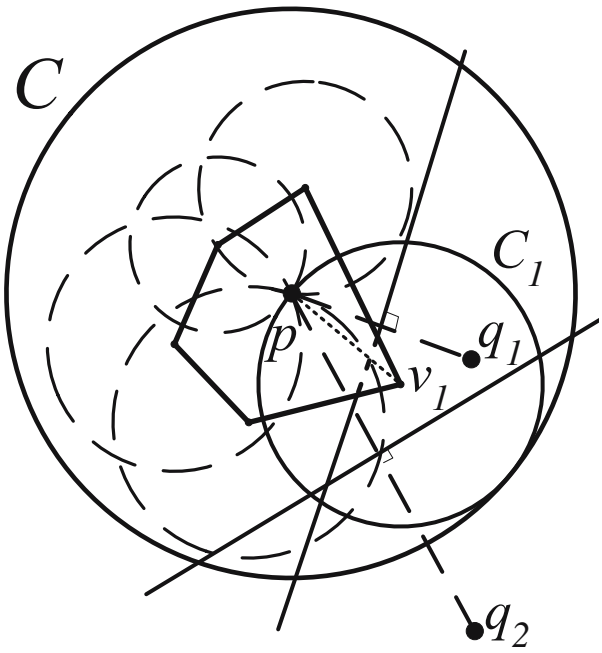

Fig. 8 The effect of the two new points q_1 and q_2 on the Voronoi cell of p. The neighborhood region C is a superset of all circles C_i

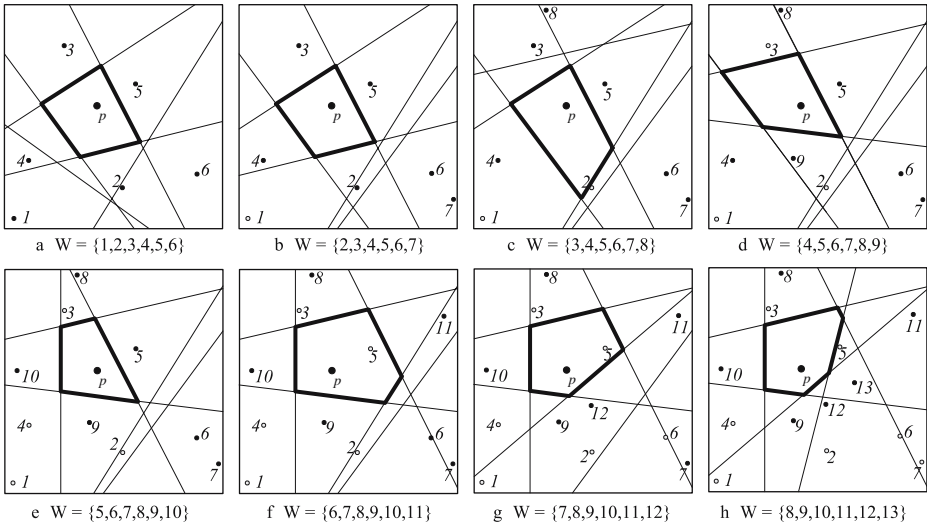

Fig. 9 The Voronoi cell of point p over a sliding window W of size 6 for seven subsequent time instances. The label of the each point shows its arrival order

has received only the location of the nodes inside a circle smaller than C. We call the later circle centered at p the *border circle* of p. When finalizing the current Voronoi cell, the node clips the cell using this circle to remove the unexplored areas from the cell.

5.3 Voronoi cell over a sliding window

In this section, we consider the *sliding window* case when each sensor node must independently build its Voronoi cell with respect to the set of recently received locations (e.g., only those received in the last hour). Here, the goal is to maintain the Voronoi cell of p with respect to the set of points arrived so far in a window W of fixed size. With an example, we show that to compute the exact cell and keep it up-todate at any point in time, we must store all unexpired points (i.e., those in the window). That is, the algorithms described in Sections 5.1 and 5.2 cannot simply drop the points that are not contributing to the cell.

To illustrate, Fig. 9 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 Fig. 9a, 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 Fig. 9c and 9f, the point 6 does become a Voronoi neighbor of p. On the other hand, the point 7 never becomes a Voronoi neighbor of p during any of the time instances when the point 7 is in the current window (Fig. 9b–9g).

This example shows that our global/local algorithms 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.[2]

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

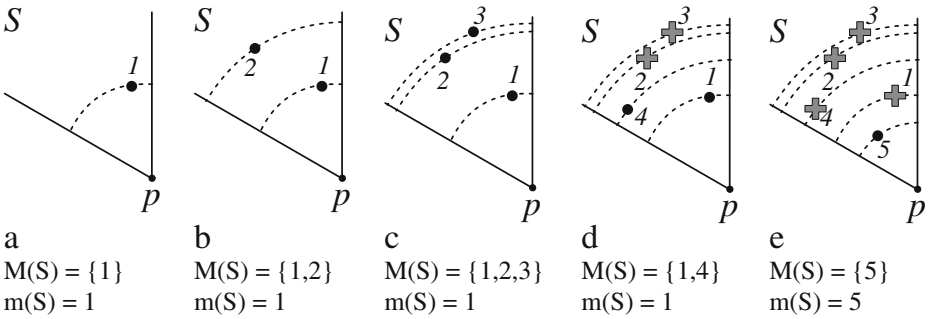

Fig. 10 AVC-SW updates the set $M(S)$ and the minimum point $m(S)$ for each of the 5 arriving points in the sector S. Assume that none of these points expire during this illustration

Therefore, the global and local algorithms are too expensive in terms of memory requirements for realistic scenarios. Motivating by this observation, we propose an algorithm to maintain an approximate Voronoi cell of a point p over a sliding window.

5.3.1 The AVC-SW algorithm

Let w be the window size, and at each time instance t, only one point arrives. 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 $\theta=2\pi/k$. As Fig. 11a 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. We also store a set of points $M(S_i)$ for the sector 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 of points (e.g., sensor locations in our application).

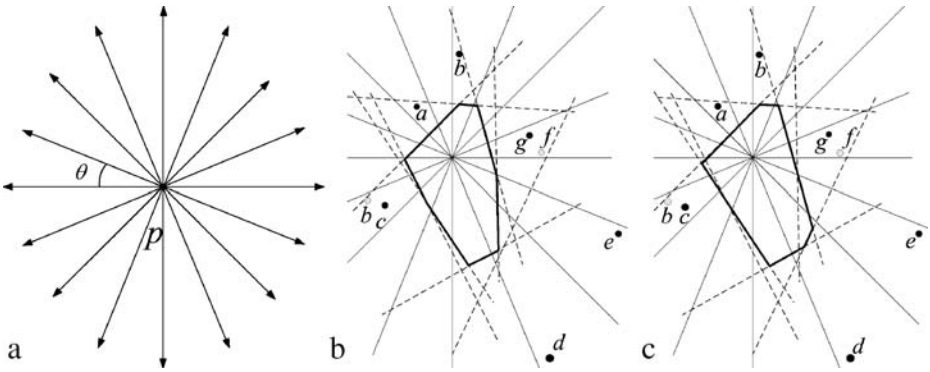

Fig. 11 **a)** $k=16$ vectors originating from p divide the space into k identical sectors, **b)** The Voronoi cell of the point p, and **c)** The approximate Voronoi cell of p

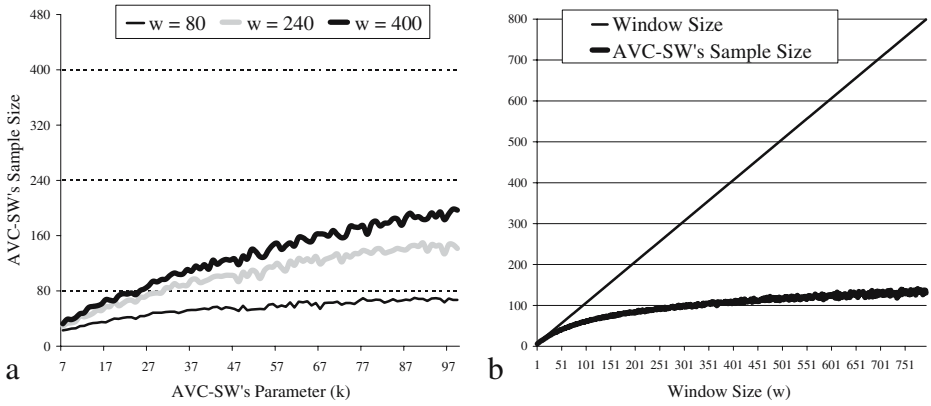

Fig. 12 **a)** Average number of points stored by AVC-SW (i.e., AVC-SW's sample size) for **a)** w=80, 240, and 400 and different values of k, and **b)** different values of w when k=36

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 $d(p,y)>d(p,x)$. 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 $M(S)$. Now, the Voronoi cell of p derived from the set of k minimum points $m(S_i)$'s corresponding to k sectors is the approximation of the actual Voronoi cell of p.

Figure 10 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 Fig. 10a, the point 1 is the current minimum point of S. When the point 2 and 3 arrive in Fig. 10b and 10c, 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 current minimum point of S in the window. In Fig. 10d, 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 Fig. 10e and causes the update to the minimum point and deletion of all the points in $M(S)$.

It is clear that the approximate Voronoi cell of p, contains its actual Voronoi cell. We can compute this approximation in $O(k \log k)$ time and space at any time using the global algorithm from the scratch. We use AVC-SW(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-SW's approximation to the Voronoi cell of the point p. Figure 11b shows the exact Voronoi cell of p with respect to the set $N = \{a, b, c, d, e, f, g, h\}$. Figure 11c shows $V'(p)$ created by AVC-SW($\theta = \pi/8$). The filled dots in the figure are minimum points of the sectors while the empty dots are dropped by AVC-SW.

We need to study the approximation error of $V'(p)$. We prove in [11] that the Voronoi cell computed by the AVC-SW algorithm is a $(1+\varepsilon)$- 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 (e.g., r) is less than its distance to p by at most a factor of $1+\varepsilon$ (i.e., $d(q,p)/d(q,r) \leq 1+\varepsilon$). 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-SW(θ) results in an approximation of tolerable error ε. While

the theoretical proof of our findings is out of scope of this paper, we only mention the following theorem:

Lemma 3 *For a given error bound ε, the largest θ, using which AVC-SW(θ) can compute an approximate Voronoi cell with a maximum error of ε is computed from the following equation:*

$$\theta = \pi/4 - \arccos\left(\frac{\sqrt{2}}{4}\left(3 - \left(\frac{1}{1+\varepsilon}\right)^2\right)\right)$$

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 $M(S_i)$. 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 Fig. 10a–c). We conducted an experiment to evaluate the average space used by AVCSW. 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 12a illustrates the average sample size (κ) of AVC-SW for three different window sizes when we vary the parameter k. It shows when the window size is greater than k (e.g., $k = 67$ and $w = 400$), the sample size is far less than w. Figure 12b shows the same measurement for $k = 36$ and different window sizes. It shows up to 80% reduction in memory requirement for $k = 36$ and large windows.

We theoretically computed the expected sample size of AVC-SW in terms of its parameter k, the window size w, and a harmonic number dependent on both k and w. With the sliding window model and a uniform distribution of the site points, AVC-SW significantly reduces the space complexity of the classic algorithm from $O(w)$ to $O(\log w/k)$ [11].

6 Experimental results

We conducted several experiments using two real-world sensor datasets:

1. **PRECIPITATION** is a real-world dataset that measures the monthly precipitation of gridded points all over the globe for 50 years.[3] We extracted different 53×43 grids based on this dataset. We ran 100 spatial average queries on the precipitation values monitored by simulated sensor nodes located in a randomly defined region R.
2. **TEMPERATURE** is a real-world dataset that measures the temperatures at points all over the globe at different altitudes for 18 months, sampled twice per day. We sliced the dataset at certain altitudes and times to construct different grids of size 64×128 with temperature as the value of each cell. We measure the average accuracy of 100 spatial average queries on the temperature values in different regions.

We compare the accuracy of the spatial average computed using the SMA, Voronoi and TIN methods. In addition, we evaluate the precision of our local Voronoi cell computation algorithm when it is used to approximate the global Voronoi cell.

[3] http://www.jisao.washington.edu/data_sets/willmott/.

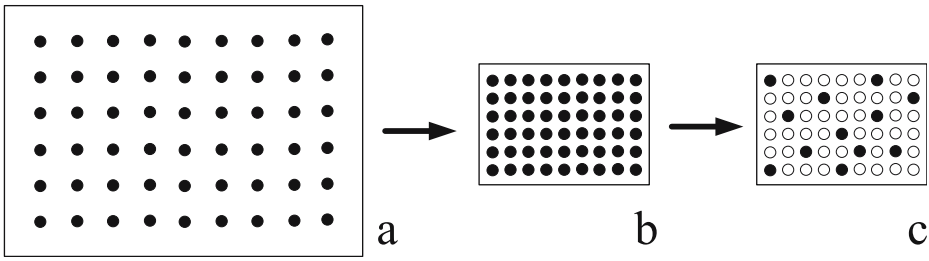

Fig. 13 Mapping the original grid into a grid of 5×5 cells and generating a sensor network with random placement

For our experiments, we developed UNIX processes simulating individual sensor nodes. The sensor network was simulated as follows. First, we used real-world data from our datasets, **PRECIPITATION** and **TEMPERATURE**, collected from observation points on different grids as values monitored by the sensor nodes. To map both datasets into a single coordinate space, we reduced the resolution of each dataset into a course grid dividing the *xy* plane into 5×5 cells. Subsequently, we randomly selected a set of grid points as the locations of the sensor nodes to generate random networks of different densities. Finally, the value that each sensor node monitors was selected as the value of the grid point in the original dataset. Figure 13 illustrates generating a random network using the original dataset. The black points of the grid *c* shows the locations of the nodes in the network generated from the original grid *a*. The *node density* of the network is defined as the percentage of the grid points selected as nodes from the original dataset.

In all of our experiments, we used the original gridded dataset as a reference dataset. We calculated the precision of a spatial average on nodes inside an area R (termed $SAVG_R$) by comparing it with AVG_R, the traditional average of all observations from the original dataset inside R. We have then measured the absolute relative error values as $\frac{|SAVG_R - AVG_R|}{AVG_R}$. In Fig. 13, considering R as the entire network, $SAVG_R$ is the spatial average on the values of the black points in the grid *c* while AVG_R is the traditional average on the values of all the points in the grid *b*.

6.1 Precision

The first set of experiments was aimed to study the precision of each of the three proposed methods to compute the spatial average. The measurements are made based on the distributed

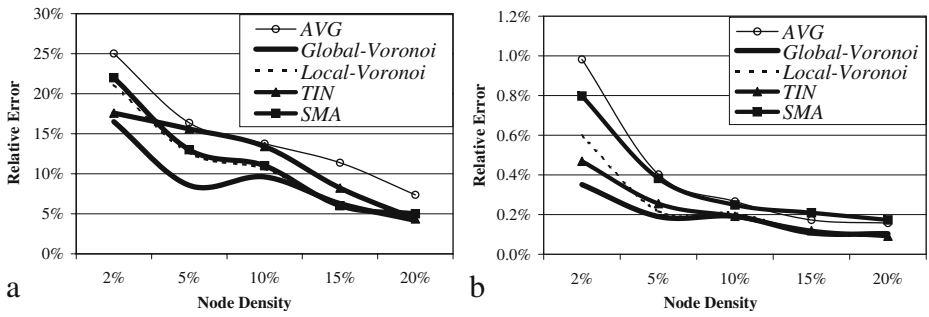

Fig. 14 Comparing the relative error of all methods computing the spatial average on **a) PRECIPITATION** data, and **b) TEMPERATURE** data with the traditional average

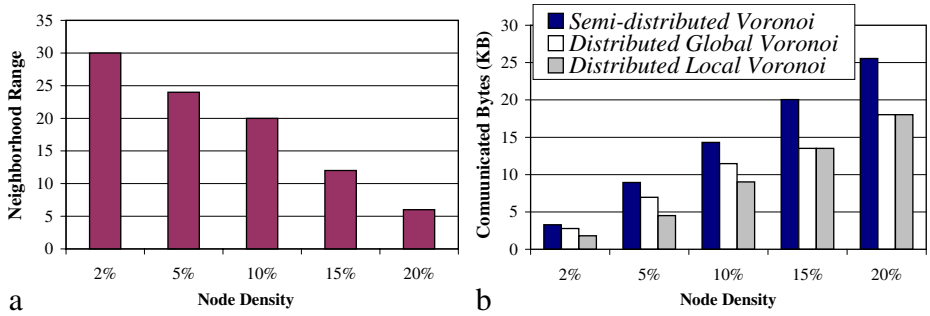

Fig. 15 **a)** The maximum extent of neighborhood required to be known in each sensor node to compute its global Voronoi cell. **b)** Comparing the number of bytes sent through the network by different approaches

approach discussed in Section 4.2. Figure 14a illustrates the relative error of the traditional and spatial average queries on sensor networks of different densities measuring precipitation. In the figure, *Local-Voronoi* is the result of processing the query by the Voronoi method using the local Voronoi cells generated based on a radial neighborhood of radius 15 for each sensor node. Besides, for each node density, the size of the grid cell used for the SMA method is the one which results in the best average. As we are expecting, the error of all methods is decreasing as the density of the nodes (i.e., observation points) is increasing. The global Voronoi generates the most accurate result. Even using the area of the local Voronoi cell generated using a neighborhood of size 15 results in less error than the traditional average. As the figure shows, local Voronoi and SMA perform similarly for this dataset. However, this observation cannot be generalized as SMA's accuracy is completely dependent on the size of the SMA grid cell and the distribution of the values in these cells. These parameters cannot be fine tuned apriori based on the network node density. Consequently, this makes SMA an unfeasible aggregation method for the sensor networks.

Figure 14b illustrates the results of the same experiment on the networks generated based on the **TEMPERATURE** dataset. The error value of all methods are less than 1% even in the sparse networks. The reason is that the values in the original dataset has already been interpolated using a set of observations in order to be transformed to a grid. Despite the ignorable error, the traditional average has the worst error comparing to the other methods. Again, the spatial averaging based on the Voronoi method outperforms all the other methods in accuracy. Here, SMA performs as worse as the traditional average which verifies our conclusion from the previous experiment.

In the next set of experiments, we investigate the radius of radial neighborhood required to compute a local Voronoi cell as good as that of the global cell (i.e., 100% accurate cells). Figure 15a depicts that for the sparse networks the local Voronoi cell computed using the location of all the nodes closer than 30 units to the center node is almost as accurate as the global cell. The average of this distance is 6 for the dense networks. Notice that the local cells generated during this experiment were clipped using the border circles. This observation exploits the fact that our distributed local Voronoi algorithm is expected to cause less communication overhead than the semidistributed approach. We verify this fact through our last experiments.

6.2 Performance

The last set of experiments was aimed to compare the number of bytes sent by all the sensor nodes during semi-distributed and distributed approaches to the Voronoi-based spatial average

processing. We issues 100 spatial average queries on the temperature values over the entire network and measured the average number of bytes sent to route the queries and return the partial results. We considered 7, 2 and 2 bytes for query, partial result and location messages, respectively.[4] The location of the query node was randomly selected for each query run.

Figure 15b shows the total number of payload bytes (ignoring the packet headers) for different densities. In the figure, the distributed local Voronoi approach is the distributed approach in which each node computes its local Voronoi cell based on the location of the nodes one hop away. As the figure illustrates, the communication overhead in the semi-distributed approach is always more than that of distributed approaches by a factor of 25%. This increase is because of the weight messages sent by the query node to each single node in the network. Figure 15b shows that as the node density increases, the amount of information sent to compute global and local Voronoi cells becomes comparable. The intuition here is that the global and local Voronoi cells are the same in the dense sensor networks. The observations verify that employing our algorithm by each node to compute the Voronoi cell results in less communication overhead in a distributed environment. In particular, in the context of sensor network, this saves a lot of energy resulting in longer lifetime for the nodes.

6.3 Discussion

In our prototype of the distributed approach to spatial aggregation, the nodes compute their weights during forwarding the query to the other nodes. This is expensive in terms of communication costs especially when the queries are frequent. Towards this end, the design can be changed to separate the weight computation phase from the query routing phase. Since the weights are query-independent and are only dependent on the location of the neighboring nodes, the nodes can compute the weights at any time. Therefore, at certain time periods or on specific events (e.g., detecting a failure in one of neighbors) a weight computation module can be triggered.

Although we used the ad-hoc query routing to disseminate the query within the network, other suggested efficient query routing schemes can completely support the spatial aggregation queries. This is due to the independence of the weight computation from the aggregate query. However, there might be changes needed to piggy back the messages related to the weight computation with the query routing messages.

7 Conclusion

Applying traditional database aggregation techniques on the data generated by a sensor network is not an appropriate mean to extract characteristics of the underlying real-world continuous phenomenon. In this paper, we introduced the spatial aggregation for sensor network databases. Our general formalization of the spatial aggregation operators can be implemented on top of any in-network aggregation processing scheme (e.g., TAG [7]). We showed how a spatial average operator can be transformed into a weighted average on the sensor node measurements. We used spatial interpolation methods to formally define three different spatial average operators, namely, SMA, Voronoi and TIN-based operators. Throughout several empirical experiments to extract the average of a

[4] We ignored the size of packet headers throughout this experiment.

continuous process using discrete non-uniform samples, we demonstrated that all three operators outperform the traditional average operator in accuracy. We showed that the Voronoi-based operator improves the accuracy of the traditional operator by a factor of 35% for sparse networks.

In addition, we identified semi-distributed and distributed approaches to implement spatial operators on the sensor nodes. According to our experimental results, the communication overhead in the semi-distributed approach is always more than that of the distributed approach by a factor of 25%. Hence, we proposed a local Voronoi cell computation algorithm to be used by the distributed approach. For the real-world case when the sensors fail frequently, we proposed the AVC-SW algorithm to approximate the Voronoi cell at each node reducing the memory requirements of our Voronoi algorithms. When it is used by each sensor node, the algorithm provides the best possible approximation to both global and local Voronoi cells of the node given its limited communication range. This algorithm is applicable to other fields such as moving object databases, robotics and mobile computing.

Acknowledgments This research is based upon work supported in part by the National Science Foundation under award numbers IIS-0324955(ITR), EEC-9529152 (IMSC ERC) and IIS-0238560 (CAREER), in part by a grant from NASA/JPL, and in part by unrestricted cash gifts from Microsoft. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. S. Arya and A. Vigneron. Approximating a Voronoi Cell. Technical Report, 2003, HKUST-TCSC- 2003-10.
2. P. Bonnet, J.E. Gehrke, and P. Seshadri. "Towards sensor database systems," in *Proc. of the Second International Conference on Mobile Data Management*, pp. 3–14, 2001.
3. S. Fortune. "A sweepline algorithm for Voronoi diagrams," in *Proc. of the Second Annual Symposium on Computational Geometry*, pp. 313–322, ACM Press, 1986.
4. R.H. Güting. "An introduction to spatial database systems," *The VLDB Journal*, Vol. 3(4):357–399, 1994.
5. C. Intanagonwiwat, R. Govindan, and D. Estrin. "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proc. of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCOM '00)*, 2000.
6. N. Lam. "Spatial interpolation methods: A review," *The American Cartographer*, Vol. 10(2):29–149, 1983.
7. S. Madden, M. Franklin, J. Hellerstein, and W. Hong. "TAG: A tiny aggregation service for ad-hoc sensor networks," in *Proc. of the 5th Annual Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.
8. S. Muthukrishnan. Data Streams: Algorithms and Applications. Technical Report. Computer Science Department, Rutgers University, 2003.
9. A. Okabe, B. Boots, K. Sugihara, and S.N. Chiu. Spatial Tessellations, Concepts and Applications of Voronoi Diagrams. 2nd edition, John Wiley and Sons Ltd.: London, UK, 2000.
10. M. Oliver and R. Webster. "Kriging: A method of interpolation for geographical information systems," *International Journal Geographic Information Systems*, Vol. 4(3):313–332, 1990.
11. M. Sharifzadeh and C. Shahabi. Approximate Voronoi Cell Computation on Geometric Data Streams. Technical Report. Computer Science Department, University of Southern California, 2004. No. 04–835.
12. M. Sharifzadeh and C. Shahabi. "Supporting Spatial Aggregation in Sensor Network Databases," in *Proc. of the 12th ACM International Symposium on Advances in Geographic Information Systems*, pp. 166–175, 2004.
13. I. Stanoi, M. Riedewald, D. Agrawal, and A.E. Abbadi. "Discovery of influence sets in frequently updated databases," in *Proc. of the 27th International Conference on Very Large Data Bases*, pp. 99–108, Morgan Kaufmann Publishers Inc., 2001.
14. W. Tobler. "Cellular geography," in Ozlsson and Gale (Eds.), *Philosophy in Geography*, 379–386, D. Reidel Publishing Company: Dordrecht, Holland, 1979.
15. J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D.L. Lee. "Location-based spatial queries," in *Proc. of the 2003 ACM SIGMOD International Conference on Management of data*, pp. 443–454, ACM Press, 2003.
16. J. Zhao, R. Govindan, and D. Estrin. "Computing aggregates for monitoring wireless sensor networks," in *Proc. of the First IEEE International Workshop on Sensor Net Protocols and Applications (SNPA'03)*, 2003.

Mehdi Sharifzadeh received the BS and MS degrees in computer engineering from Sharif University of Technology, Tehran, Iran, in 1995 and 1998, respectively. He is currently working towards the Ph.D. degree in Computer Science at the University of Southern California. Mehdi is a research assistant working on Multidimensional and Spatial Databases at the Integrated Media Systems Center (IMSC)—Information Laboratory of the University of Southern California. His research interests include spatial and multidimensional databases, data stream processing, computational geometry, and data mining.

Cyrus Shahabi received the BS degree in computer engineering from Sharif University of Technology, Iran, in 1989 and the M.S. and Ph.D. degrees in computer science from the University of Southern California in 1993 and 1996, respectively. He is currently an associate professor and the director of the Information Laboratory (InfoLAB) in the Computer Science Department and also a research area director at the US National Science Foundation's Integrated Media Systems Center (IMSC) at the University of Southern California. He has two books and more than 100 articles, book chapters, and conference papers in the areas of databases and multimedia. Dr. Shahabi's current research interests include peer-to-peer systems, streaming architectures, geospatial data integration, and multidimensional data analysis. He is currently on the editorial board of IEEE Transaction on Parallel and Distributed Systems and ACM Computers in Entertainment magazine, and he is program committee chair of ICDE NetDB 2005 and ACM GIS 2005. He is also serving on many conference program committees such as ICDE 2006, ACM CIKM 2005, SSTD 2005, and ACM SIGMOD 2004. Dr. Shahabi is the recipient of the 2002 US National Science Foundation CAREER Award and 2003 Presidential Early Career Awards for Scientists and Engineers (PECASE). In 2001, he also received an award from the Okawa Foundations.