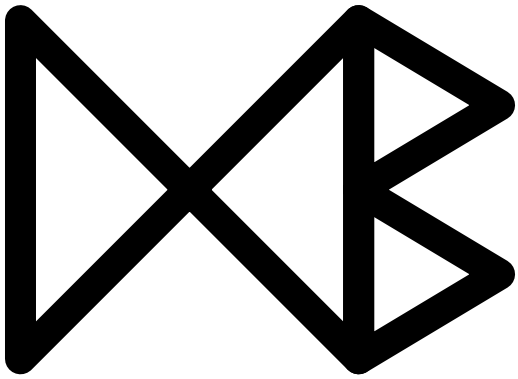


Snapshot Density Queries on Location Sensors

Xuegang Huang, Hua Lu

May 3, 2007

TR-21

A DB Technical Report

Title	Snapshot Density Queries on Location Sensors
Author(s)	Xuegang Huang, Hua Lu
Publication History	May 2007. A DB Technical Report. Extended version of: Xuegang Huang, Hua Lu, "Snapshot Density Queries on Location Sensors," in *Proceedings of the 6th International ACM Workshop on Data Engineering for Wireless and Mobile Access*, Beijing, China, June 10th, 2007.

The DB TECH REPORTS icon is made from two letters in an early version of the Rune alphabet, which was used by the Vikings, among others. Runes have angular shapes and lack horizontal lines because the primary storage medium was wood, although they may also be found on jewelry, tools, and weapons. Runes were perceived as having magic, hidden powers. The first letter in the logo is "Dagaz," the rune for day or daylight and the phonetic equivalent of "d." Its meanings include happiness, activity, and satisfaction. The second letter is "Berkano," which is associated with the birch tree. Its divinatory meanings include health, new beginnings, growth, plenty, and clearance. It is associated with Idun, goddess of Spring, and with fertility. It is the phonetic equivalent of "b."

Abstract

Density queries are of practical importance in many mobility related applications including traffic monitoring. Previous work has so far assumed a client/server architecture to solve such queries. Whereas in this paper, carefully constructed sensor networks, which consist of both lightweight location sensors and more powerful processing nodes, are proposed to answer density queries. We assume the location sensors are sensors that are placed in a geographical area and can only detect the amount of objects moving in vicinity. This paper is focused on estimating the dense regions of objects. Our approach partitions the region of interest into subregions, and deploys in each subregion both sensor nodes detecting moving objects and a processing node issuing and answering density queries. Three algorithms, *CF*, *VCF* and *GF*, are proposed to efficiently process density queries on those processing nodes. Indications of extensive empirical evaluation are threefold. First, our solution is effective as accuracy gained is acceptably high. Second, our solution is efficient as it incurs short CPU time. Third, different query processing algorithms are fit for different scenarios with specific environment settings.

1 Introduction

Recent development in wireless communication and hardware miniaturization technologies has made wireless sensor networks possible. Sensor networks consist of low power, autonomous sensor nodes with limited computation and communication capabilities. These cheap, tiny sensor nodes are often widely distributed to collect information in each vicinity and transmit the collected data to the query source through a wireless ad-hoc peer-to-peer network. Data communication in sensor networks often follows an ad hoc way which simplifies the network information but also brings difficulties to the management and configuration of the networks. Also, the resource-constraints on sensor nodes, such as limited power consumption, low communication bandwidth, and very small storage space and energy, gives more challenges to sensor network applications.

From the database point of view, the data being processed, transmitted and stored in sensor networks form a database, namely sensor network database. Many types of queries and query processing techniques have been discussed in this context. For instance, aggregate query processing in sensor networks has received broad interest in the literature [2, 5, 12, 13].

In this paper, we assume an application scenario where sensor networks are used to detect moving objects' activities. Focus of our discussion is the monitoring and processing of so-called density queries which find areas where objects tend to be very close to each other. Examples of such queries in the real world are "finding the road intersections or streets that have more than 200 cars stayed or passed in the last 10 minutes" or "finding regions that are smaller than 10 m^2 in area size but contain more than 100 objects." Density query is an essential functionality in systems that monitor moving objects data. Such systems often have an assumption that a smart client with a positioning device (e.g., GPS) collects and sends location data of moving objects to a central server. However, the widespread of positioning device has not yet come to be true and the assumed architecture has to be modified to comply with issues such as violation of privacy. The emergence of sensor networks brings an alternative for the current architecture. We list three motivating examples for our discussion.

First, finding roads or junctions that are "busy" at the moment is an essential task of traffic-monitoring systems. Sensor networks can be deployed to detect the quantity of moving vehicles at certain areas and such data can be collected and transmitted to the traffic monitoring center so that those roads or junctions with high volume of detected vehicles are the areas of traffic jam or congestion. This solution has no cost on the vehicles side and, compared to the GPS-based the solutions, does not expose the drivers' location privacy (i.e., the privacy of being at anywhere at any time) as no identity information is involved in the sensor network. Second, in the study of animal ecology, it is very useful to observe the activities and behavior of a population of particular species. Sensor networks is the most convenient solution for such observations. Since the activities that involve a large population of a species are usually of interest to scientists, monitoring areas with large amount of observed objects is an essential tasks of these scientific observation systems. Another example is from the battle field. Considering that a lot of "motion detection" sensors are deployed at a battle field to detect activities of enemy targets, small areas with a great amount of enemy targets will be cost-effective candidates for missile attacks.

Motivation of this paper also comes from the spatiotemporal database literature. The discussion on dense area discovery [7] and density queries [8] addresses a novel spatiotemporal query, namely the density-based query, which finds regions with vast amount of spatiotemporal objects against the relatively small size of region.

With all the motivation, this paper proposes a sensor-network-based framework for moving object detection. Based on a top-level framework, the paper proposes three approaches for discovering density areas and studies the

accuracy and efficiency of these approaches. The paper also reports an empirical study that thoroughly evaluates the proposed approaches.

Contributions of this paper are in two aspects. First, this paper is the first to consider density query processing in the sensor network architecture. The major distinction between density query and aggregate query is that the latter only counts the result without considering the spatial area. With the aforementioned examples, discovering dense areas with sensor networks has many useful applications which can not be solved with aggregate queries. Second, this paper proposes three query algorithms for density query in sensor networks. Instead of designing a new sensor network architecture, this paper reuses an existing architecture of sensor network so that the proposed algorithms is easily extensible.

The rest of this paper is organized as follows. Section 2 introduces related work. Section 3 presents background. Section 4 addresses a generic framework for monitoring dense regions in sensor networks and introduces three query algorithms. Section 5 discusses the implementation detail and computational complexity of the algorithms. Section 6 empirically compares the accuracy and efficiency of the proposed algorithms. Finally, Section 7 concludes.

2 Related Work

Aggregate query processing in sensor networks is an orthogonal topic to our discussion. Quite a few papers from the database community have been addressing techniques and solutions for this topic [2, 3, 5, 11, 12, 13, 14]. In our paper, we choose to form our discussion based on the sensor network architecture described in [12]. This architecture assumes the existence of both light-weight sensor nodes which only collect and transmit the raw data, and powered sensor proxies that further process and route the answers to the query processor. Different from [12] which addresses the low-level infrastructure issues and considers how to maintain high query throughput with limited sensor resource demands, our paper considers density queries on moving objects in a sensor network and we are focused on getting more precise result of dense areas. The discussion of our paper can be seen as one concrete application based on the architecture of [12]. Our discussion in this paper, snapshot density queries on location sensors, is different from snapshot queries discussed in [10], which is focused on energy efficient collection of quick but approximate answers from part of all sensors.

Our work is similar to papers that work on tracking and managing mobile targets in sensor networks [6, 15]. Unlike these papers that focus on the low level details of location tracking and object clustering, our paper assumes that a location sensor can only count the amount of objects in a small range and we are focused on finding effective and efficient algorithms to estimate the dense regions of objects. Such sensors can either be anonymous RFID readers or optical motion detectors.

The topic of density query has been brought forward by two recent papers from the database community that study the querying of spatiotemporal regions with a high concentration of moving objects [7, 8]. The first paper [7] proposes to divide the data space into a uniform grid so that the density query is simplified as reporting cells that satisfy the density conditions. This solution provides fast answers but can lead to *answer loss* (as termed in the second paper [8]), such as regions that cover boundaries of several cells with a high density of objects (but each individual cell does not contain enough number of objects to be dense). The second paper [8] provides a new definition of density query that eliminates answer loss and proposes a filter-and-refinement algorithm for computing the queries. We formulate the definition of density query based on these two papers. The difference between our paper and these two is that we consider density query processing in a different architectural setting, where each moving object' positions can not be reported from itself but are detected by the sensor network. Since paper [8] provides the most accurate density query result, in our empirical study, we choose to compare the density computation result with [8] to evaluate on the estimation accuracy of the proposed algorithms.

The discussion of this paper is also relevant to the topic of density clustering [4] and clustering of spatiotemporal objects [9]. In [4], the DBSCAN algorithm is proposed to find dense clusters of objects. Paper [9] is focused on the discovery of moving clusters in a database of moving object trajectories. The techniques in these papers can not be directly applied to our scenario since the physical limitation of location sensors makes it impossible to get the exact location of objects. However, our assumption on the limitation of location sensors is very simple which, on the other hand, makes our discussion more extensible in real world applications.

3 Background

This paper assumes that a population of target objects move within a 2D area where a sensor network is deployed. These sensors can detect objects in vicinity. Specifically, we do not assume the sensors can tell the exact locations of each detected object. Instead, a sensor can only count the amount of objects in its vicinity. We are interested in getting those areas where objects are very close to each other, by processing and analyzing the collected sensor data. We proceed to describe the settings of the sensor network and the definitions of density queries.

3.1 Sensor Network Architecture

In our setting, sensor nodes have limited power as well as computation and storage capabilities. They possess equal role in the network functionality. Each node is connected to other nodes in vicinity through a wireless network, which form its neighborhood. A node can either send a message to one of its neighbors or simultaneously broadcast the message to all or most neighbors. Thus, a sensor node communicates with other nodes that are spatially distant through a multi-hop routing protocol. Two types of messages are transmitted among the sensor nodes for query processing: *query messages* that transport the query information and *answer messages* that transport the query result.

In addition to sensor nodes, we distinguish a special type of node called *processing node*. Sensor nodes collect data, send and route data to processing nodes. Processing nodes can issue and process queries. Processing nodes have abundant resource and stable, long-range connections (such as cable and satellite connections) to each other. Each processing node is able to talk with sensor nodes through wireless protocols. When a processing node sends out a density query, the query is disseminated to sensor nodes in the vicinity of the processing node and also sent to other processing node through the stable connection. After the query is distributed by all the processing nodes, the results from sensor nodes are sent back to these processing nodes which are then returned to the one that originates the query. Figure 1 illustrates the architecture described above.

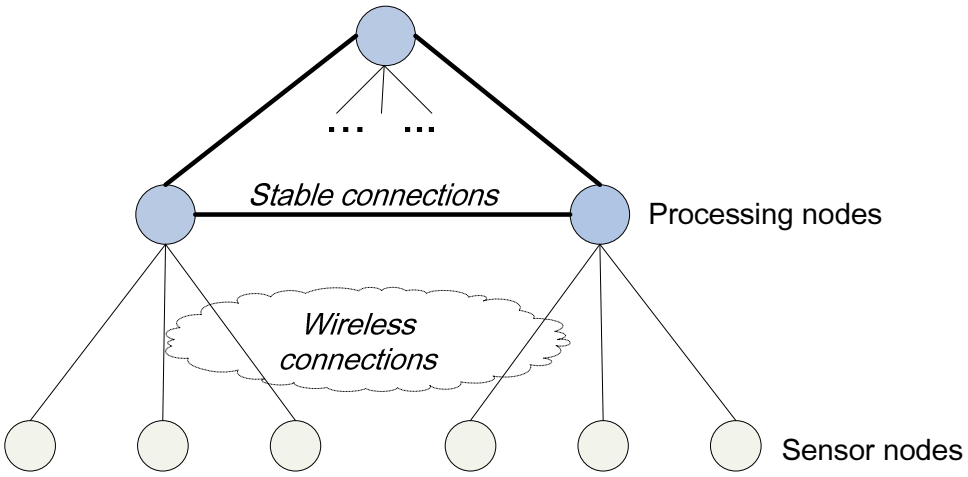


Figure 1: System Architecture

Next, we assume that each sensor node knows its location as well as the location of its neighbors. Each sensor node also knows its closest processing node (e.g., it can be detected by an activation operation of the processing node). A "motion-detection" sensor is installed at each sensor node which is able to detect the amount of objects within a finite range. All sensors are timely synchronized to a global clock and sample their readings periodically. The size of moving objects can be ignored compared to the distance among sensor nodes. The positions of processing nodes can be decided, based on the distribution of sensor nodes, during the system installation.

3.2 Density Query

Each object is represented as tuples $\ldots, (x_i, y_i, t_i), \ldots$ where (x_i, y_i) is the location of this object at time instance t_i. Since the objective is to find spatial areas where, with high probability, objects are very close to each other,

we adapt similar definitions in related work [7, 8] and formalize the definitions of snapshot density and snapshot density query in the following.

Definition 3.1. (Snapshot Density) The snapshot density of a region $\mathcal{A}$ is defined as $Density_t(\mathcal{A}) = \frac{MO_t}{AREA(\mathcal{A})}$, where MO_t is the amount of moving objects inside the area at time point δ_t and $AREA(\mathcal{A})$ is the total area of $\mathcal{A}$.

Hence, we say that a region is *dense* at time t if its density is over a density threshold θ. To skip the cases when the regions become too small which leads to high density values, we assume that the total area of any region should be within an area range $[\alpha_1, \alpha_2]$. The time t is a single time point. Since the sensor nodes have limited storage capabilities and the sensor readings are collected like data streams, only the most recent data can be shortly cached at each node before sending to processing nodes. In the discussion of our paper, we are focused on queries that find dense regions at individual time points.

Definition 3.2. (Snapshot Density Query) Given a set of moving objects, the thresholds $\theta, \alpha_1, \alpha_2$, and a time point t, a density query finds non-overlapping regions $\mathcal{A}_1, \mathcal{A}_2, \ldots$ such that $\alpha_1 \leq AREA(\mathcal{A}_i) \leq \alpha_2$ and $Density_t(\mathcal{A}_i) \geq \theta$ ($i = 1, 2, \ldots$).

A snapshot density query is issued from a processing node, when it is interested in the current distribution of dense regions. We call these density queries *online density queries*.

Since the sensors cannot get the precise locations of moving objects, we will only focus on algorithms that give estimated result of the density query. We consider the accuracy of such algorithms in terms of *false positives* and *false negatives*. False negative is the error of not finding a dense pattern that does exist in the data. False positive, is the error of finding a "dense pattern" that does not exist in the data. Thus, density query algorithms are supposed to reduce both of them. We proceed to describe the detail of our solutions.

4 Online Density Computation

With our sensor network, the density queries are periodically initiated from a processing node. Processing one snapshot of the density query involves two parts, i.e., communication among processing nodes and query processing at each processing node. Steps involved in the first part is straightforward: The current processing node sends out the query to other processing nodes, receives their results, and combines all the results as the query answer. We assume that query processing at each processing node does not involve any communication with other processing nodes. Our discussion is focused on techniques for query processing at each processing node. This involves a processing node and the sensor nodes that are in vicinity of this node.

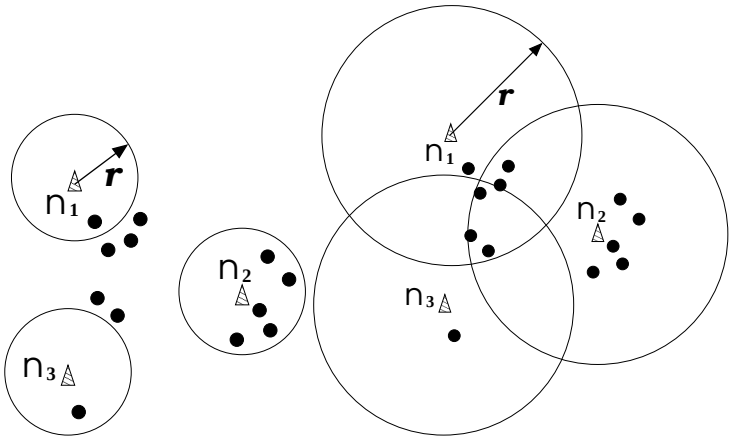
We assume that all sensor nodes can only process one type of query, denoted as *AMOUNT*(r). This query asks for the amount of moving objects detected by a sensor within a specified distance range r at the current time. The value of parameter r should be within a technical bound of the sensor. Suppose the query is issued at processing node p and the sensor nodes in the vicinity of p are $n_1, n_2, \ldots, n_m$, to process the density query, the node p first broadcasts the query *AMOUNT*(r) to $n_1, n_2, \ldots, n_m$. We model the result of query *AMOUNT*(r) on sensor node n_i as $c_i = (shp, amnt)$ $(i = 1, 2, \ldots, m)$ where shp is the geometrical description of the region detected by the sensor at node n_i (intuitively, it is a circular shape with radius r), $amnt$ is the amount of moving objects detected by the sensor. After receiving answers $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$ from the sensor nodes, a filtering algorithm is used to analyze these answers and return the dense regions. We proceed to introduce three filtering algorithms.

4.1 Fixed Circular Filtering Approach

The straightforward way of finding dense regions is to scan the received results $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$. Recall that we have threshold values $\alpha_1, \alpha_2, \theta$ for determining dense regions. For a region c_i, if $\alpha_1 \leq AREA(c_i.shp) \leq \alpha_2$ and $\frac{c_i.amnt}{AREA(c_i.shp)} \geq \theta$, then c_i is a dense region. We denote this straightforward approach as *CF* (Circular Filtering).

The parameter r in the query *AMOUNT*(r) broadcasted to sensor nodes actually decides the accuracy of the filtering algorithm. When r is too small, there will be false negatives since there can be objects not reported from any sensor nodes. As illustrated in Figure 2(a), the circular area around sensor node n_2 is easily found to be dense as there are 5 objects (black dots) inside the area. But there is also a dense area of 5 objects not reported by any sensor nodes. When the value r is increased, the region that each sensor node should report overlaps with

regions of other adjacent sensor nodes which can bring false positives. For instance, in Figure 2(b), if the circular areas of n_1, n_2, n_3 are found to be dense, the objects inside both the region of n_3 and n_1 are actually reported twice. To improve the accuracy in the filtering algorithm, the parameter r should be tuned to a value so that each



(a) Small r Value (b) Big r Value

Figure 2: Accuracy with Different r Values

reported region of sensor node has very little intersection with other regions but does not have too much overlap with the others. However, the value r is actually constrained by the technical configurations on the sensors. Under the current settings, to reduce false negatives, one can use the biggest r value possible and apply the following heuristic: First, we sort $c_1, c_2, \ldots, c_m$ based on their $amnt$ values and scan those with bigger $amnt$ values at first; Next, for all the other non-reported regions, we cut their overlaps with c_i so that their area values are decreased; Then, we reevaluate the density of these regions based on their new area values.

The *CF* algorithm returns a set $\mathcal{D}$ of dense regions. In the pseudo code, we associate each region c_i with an additional boolean attribute $c_i.valid$, which is set to TRUE initially. The code is listed in the following.

```
(1)  procedure CF(C, θ, α1, α2)
(2)  sort C s.t. c_i.amnt ≥ c_{i+1}.amnt, i = 1, ..., m-1
(3)  b ←TRUE
(4)  while b
(5)     b ←FALSE
(6)     for each c_i ∈ C, if c_i.valid :
(7)        if α1 ≤ AREA(c_i.shp) ≤ α2 ∧ c_i.amnt / AREA(c_i.shp) ≥ θ
(8)           D ← D ∪ {c_i}
(9)           b ← TRUE; c_i.valid ← FALSE
(10)          for each c_j ∈ C, j ≠ i, if c_j.valid :
(11)             if c_j ∩ c_i ≠ ∅
(12)                c_j.shp ← c_j.shp − c_j.shp ∩ c_i.shp
(13) return D
```

This algorithm scans each element in the list $\mathcal{C}$ and finds out the possible dense regions. Specifically, the algorithm reads one element c_i and checks if it is dense (line 7). If so, c_i is pushed to queue $\mathcal{D}$ and the rest of the non-dense elements in $\mathcal{C}$ are checked if they have overlaps with c_i (lines 10–12). If a circular region, such as c_j, has overlaps with c_i, the algorithm updates the geometrical representation of region c_j to exclude its overlap with c_i (line 12). The while–loop (line 4–12) continues until there is no dense region left in the list $\mathcal{C}$.

4.2 Varied-Circular Filtering Approach

Since the snapshot density query is issued periodically, it is possible to ask the each processing node to send out the *AMOUNT* queries to sensor nodes with different r values at different periods and combine the observations

of all these periods. Specifically, we call this approach *VCF* (Varied Circular Filter). Our discussion focuses on using two different r values. Suppose the time period for sending out snapshot density queries is μ. Based on this assumption, at time t_a, processing node p sends out query *AMOUNT*(r_a) to the sensor nodes. Then, at time $t_a + \mu$, the node p sends out query *AMOUNT*(r_b) $(r_b < r_a)$. Assuming that objects do not move very much during the period μ, we can combine the received results for density queries. We denote the results received from the first query as queue $\mathcal{C}^a = \{c_1^a, c_2^a, \ldots, c_m^a\}$ and the second as queue $\mathcal{C}^b$.

Like the *CF* algorithm, the *VCF* algorithm uses the boolean attribute $c_i.valid$ which is associated with each region c_i and is set to TRUE initially. We assume elements in both $\mathcal{C}^a, \mathcal{C}^b$ are sorted based on the *amnt* value. For region c_i in one of the queues $\mathcal{C}^a$ and $\mathcal{C}^b$, we use an auxiliary function *CoCenter*(c_i) to return the region in the other queue that is submitted from the same sensor node as c_i. The algorithm also uses a variable c for an instance of a circular region. Output of the algorithm is the set of possible density regions $\mathcal{D}$.

(1) **procedure** *VCF*$(\mathcal{C}^a, \mathcal{C}^b, \theta, \alpha_1, \alpha_2)$
(2) sort $\mathcal{C}^a$, $\mathcal{C}^b$
(3) $b \leftarrow$ TRUE
(4) **while** b
(5) $b \leftarrow$FALSE
(6) **for each** $c_i^a \in \mathcal{C}^a$, **if** $c_i.valid$:
(7) $c \leftarrow$NULL
(8) $c_j^b \leftarrow CoCenter(c_i^a)$
(9) **if** $\alpha_1 \leq$ *AREA*$(c_i^a.shp) \leq \alpha_2 \wedge \frac{c_i^a.amnt}{\textit{AREA}(c_i^a.shp)} \geq \theta$
(10) $c \leftarrow c_i^a$; $b \leftarrow$ TRUE; $c.valid \leftarrow$ FALSE
(11) **else**
(12) **if** $c_j^b \neq$NULL$\wedge\alpha_1 \leq$ *AREA*$(c_j^b.shp) \leq \alpha_2 \wedge \frac{c_j^b.amnt}{(c_j^b.shp)} \geq \theta$
(13) $c \leftarrow c_j^b$; $b \leftarrow$ TRUE; $c.valid \leftarrow$ FALSE
(14) **if** $c \neq$NULL
(15) $\mathcal{D} \leftarrow \mathcal{D} \cup \{c\}$
(16) **for each** $c_k \in \mathcal{C}^a \cup \mathcal{C}^b$
(17) **if** $c_k.valid \wedge c_k \cap c \neq \emptyset$
(18) $c_k.shp \leftarrow c_k.shp - (c_k.shp \cap c.shp)$
(19) **for each** $c_j \in \mathcal{C}^b$, **if** $c_j.valid$:
(20) **if** $\alpha_1 \leq$ *AREA*$(c_j.shp) \leq \alpha_2 \wedge \frac{c_j.amnt}{\textit{AREA}(c_j.shp)} \geq \theta$
(21) $b \leftarrow$ TRUE; $c \leftarrow c_j$; do lines 16–19
(22) **return** $\mathcal{D}$

The algorithm scans the queue with bigger circular regions in queue $\mathcal{C}^a$. For a circular region c_i^a and its "co-center" region $c_j^b \in \mathcal{C}^b$, the algorithm first checks if the bigger circle c_i^a is a dense area (line 9). If it is not, the smaller c_j^b is checked (line 12). Then, the dense region $c = c_i^a$ or $c = c_i^b$ is pushed to queue $\mathcal{D}$ and other non-dense elements in $\mathcal{C}^a$ and $\mathcal{C}^b$ are scanned to remove the overlapping part with c (lines 14–18). After scanning through the queue $\mathcal{C}^a$, the algorithm checks the elements left in $\mathcal{C}^b$ and do the same steps of checking density as well as updating corresponding region shapes. The while–loop continues until there is no dense areas left in both queues $\mathcal{C}^a$ and $\mathcal{C}^b$.

Extending the *VCF* algorithm to three or more queues of different circular regions is straightforward. Specifically, if we have several queues $\mathcal{C}^1, \mathcal{C}^2, \ldots, \mathcal{C}^k$ with the radius sizes $r_1 > r_2 > \ldots > r_k$, the algorithm can start scanning each element in queue $\mathcal{C}^1$. If an element from $\mathcal{C}^1$ is not dense, the algorithm checks its smaller "co-center" peers in $\mathcal{C}^2, \ldots, \mathcal{C}^k$ until a dense element is found (otherwise, the current loop stops and the algorithm continues with the next element in $\mathcal{C}^1$). The dense region is then saved to the result $\mathcal{D}$ and is also used to update the regions that overlap with the current region. After scanning through $\mathcal{C}^1$, the algorithm continues to check elements left in $\mathcal{C}^2, \ldots, \mathcal{C}^k$ until the $valid$ values of all these elements are FALSE.

4.3 Grid Filtering Approach

A simple 2D grid has been used in the proposals of related work [7] to find dense areas. Following the spirit of that approach, we propose to use a grid-based filtering algorithm and estimate the amount of objects of each cell based

on the overlap between grid cells and circular regions of sensor nodes. Specifically, suppose a region c overlaps with a grid cell g, we associate g with tuple $(shp, amnt)$ where shp is the geometrical description of g and $amnt$ is the estimated objects amount and $g.amnt = \frac{AREA(c.shp \cap g.shp) \times c.amnt}{AREA(c.shp)}$.

For instance, suppose the area size of the circular regions in Figure 3 is 1. The numbers on each circle denote the area size of the overlap between circles and grid cells. For example, value 0.3 is the area size of the overlap between circular region c_1 and grid cell g_3. Since 5 objects have been detected in the circle c_1, we can calculate $g_3.amnt = 5 \times 0.3 = 1.5$. For grid cell g_1, since the overlap area size between g_1 and c_1 is 0.6, and the overlap between g_1 and c_2 (the overlap between the two circles is still considered) is 0.12, the estimated objects amount $g_1.amnt = 0.6 \times 5 + 0.12 \times 4 = 3.48$.

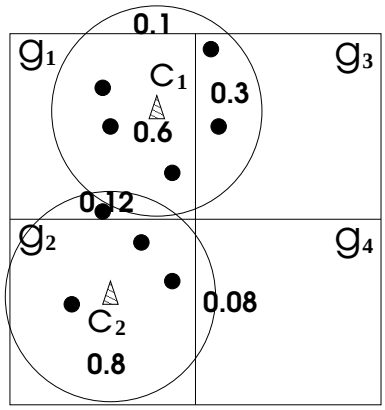


Figure 3: Example of Grid Filtering

After the calculation for all the grid cells, we can filter out the sparse cells based on the density threshold. The pseudo code of *GF* is listed in the following.

(1) **procedure** $GF(\mathcal{G}, \mathcal{C}, \theta)$
(2) **for each** $g_i \in \mathcal{G}$
(3) **for each** $c_j \in \mathcal{C}$, **if** $g_i \cap c_j \neq \emptyset$:
(4) $g_i.amnt \leftarrow g_i.amnt + \frac{AREA(c_j.shp \cap g_i.shp) \times c_j.amnt}{AREA(c_j.shp)}$
(5) **if** $g_i.amnt \geq \theta$
(6) $\mathcal{D} \leftarrow \mathcal{D} \cup \{g_i\}$
(7) **return** $\mathcal{D}$

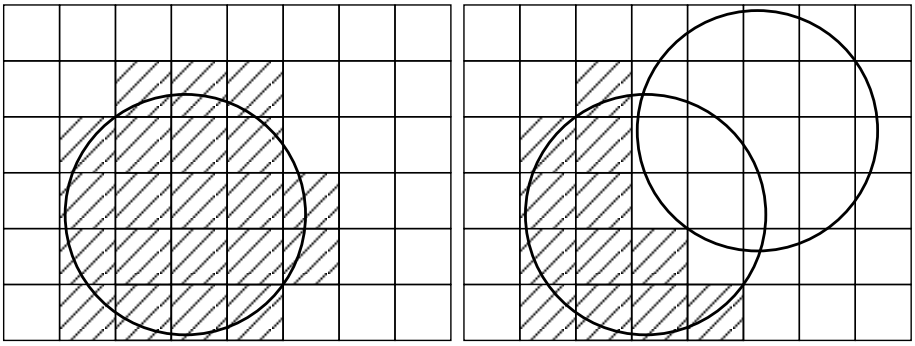
5 Discussion

Since all the algorithms proposed in Section 4 are based on the query *AMOUNT*(r), these algorithms can only find the dense regions whose center is a sensor node. When the sensor nodes are sparsely distributed and r is small, these algorithms can not find the dense regions that are distant from any nodes. Since the location sensors can not tell the exact locations of the moving objects in a neighborhood but just count the amount of such objects, it is impossible to guarantee the accuracy of a density query. The proposed algorithms give feasible ways of estimating the dense regions. Although this paper assumes that the sensor range has a circular shape, it is very easy to modify the proposed algorithms to estimate the dense regions in terms of different shapes of sensor range.

In a density monitoring scenario of our problem setting, a functionality based on an algorithm proposed in Section 4 is installed on each processing node. It is worth noting that different processing nodes can install different algorithms based on their own specific situations including computation capability and geographical proximity.

5.1 A Tile-Based Implementation

Since both *CF* and *VCF* algorithms require the operations on the circle and other shapes, we implement a tile-based solution to simplify such operations. Specifically, we apply a very dense 2D grid on the whole space and represent each shape by the set of grid cells that cover this shape. As illustrated in Figure 4(a), a circle is represented as a set of dark grid cells. Then, the operations on any shapes can be transformed into the operations on the corresponding sets of grid cells. For instance, the dark cells in Figure 4(b) represent the rest of the left circle after a subtraction



(a) Example of Circle (b) Circle Subtraction

Figure 4: Tile-Based Solution

between the two circles. We name the cells of the dense 2D grid as *basic cells*. As will be explained later, these basic cells provide a concrete way of defining *false positives* and *false negatives* in the empirical study of the proposed algorithms.

5.2 Algorithm Complexity

The efficiency of the density estimation depends on several aspects, i.e., the communication among sensor nodes and processing nodes, the computation cost at the sensor nodes, and the computational complexity of the estimation algorithms at the processing nodes. Since the cost of communication and computation at sensor nodes does not differ among the three proposed algorithms, we are focused on discussing the time complexity of these density estimation algorithms.

Suppose a processing node N_i receives sensed results from $\mathcal{N}_i$ sensor nodes, we give the brief time complexity of each algorithm.

For algorithm *CF*, it uses a three-layered nested loop. For the most outer while–loop, it is executed at most $\mathcal{N}_i$ times. This is because in each iteration of the while–loop, the if–statement (line 7) within the intermediate for–loop is at least executed once, otherwise b will be FALSE and the while–loop will be over. That means the if–statement is executed at most $\mathcal{N}_i$ times, which makes the while–loop have only one iteration. Thus, the outer nested loop is at most executed $\mathcal{N}_i$ times. The most inner for–loop is at most executed $\mathcal{N}_i - 1$ times in each iteration of the intermediate for–loop. Therefore, the worst time cost of *CF* is $\mathcal{N}_i \cdot (\mathcal{N}_i - 1)$.

Within the while–loop of algorithm *VCF*, it carries out two passes of nested loop, where the outer one is on all sensed results of a single time stamp and the inner one on all of two consecutive time stamps. The functionality and execution of the outer while–loop is similar to that in algorithm *CF*. Therefore, the worst time cost of *VCF* is $4 \cdot \mathcal{N}_i^2$.

Algorithm *GF* uses a nested loop on all grid cells and all sensed results. Therefore, its worst time cost is $\mathcal{N}_i \cdot |\mathcal{G}|$.

Comparing *CF* and *VCF*, the former is faster but the accuracy is worse as it computes the query result based on sensed data of a single time stamp only. For *GF*, its result accuracy is dependent on the grid configuration. A fine grid partition costs more computation time and is expected to produce better results. Nevertheless, the grid cell granularity should not be too fine as very small grid cells will probably cause many cells to be reported as dense ones.

6 Empirical Evaluation

To evaluate density query algorithms, we use Brinkhoff's network-based generator of moving objects [1] to generate moving object trajectories. We use integer as unit time instance and set the whole time period from 0 to 50. We generate 10,000 moving object trajectories on the Oldenburg network and 30,000 trajectories on the San Joaquin network. Each generated trajectory is treated as a unique moving object.

We generate both uniformly and non-uniformly distributed sensor and processing nodes based on the road networks. To generate the uniformly distributed sensor and processing nodes, we uniformly partition the MBR of the road networks into a 2D grid and use the center of each grid cell as the location of a node. The non-uniformly distributed nodes are generated by randomly choosing the road network nodes as the location of sensor or processing nodes. We assume that each sensor node can communicate to at least one adjacent sensor node and it sends the result of the *AMOUNT* query only to its nearest processing node. Figure 5 describes the Oldenburg datasets used in the empirical study.

Trajectories	Trajectory	Sensor and Processing Nodes
OLDN–UN	Oldenburg	Uniform
OLDN–NUN	Oldenburg	Non-uniform
OLDN–NUN'	Oldenburg	Sensor nodes: Road network nodes whose degrees ≥ 3; Processing nodes: non-uniform

Figure 5: Dataset Description

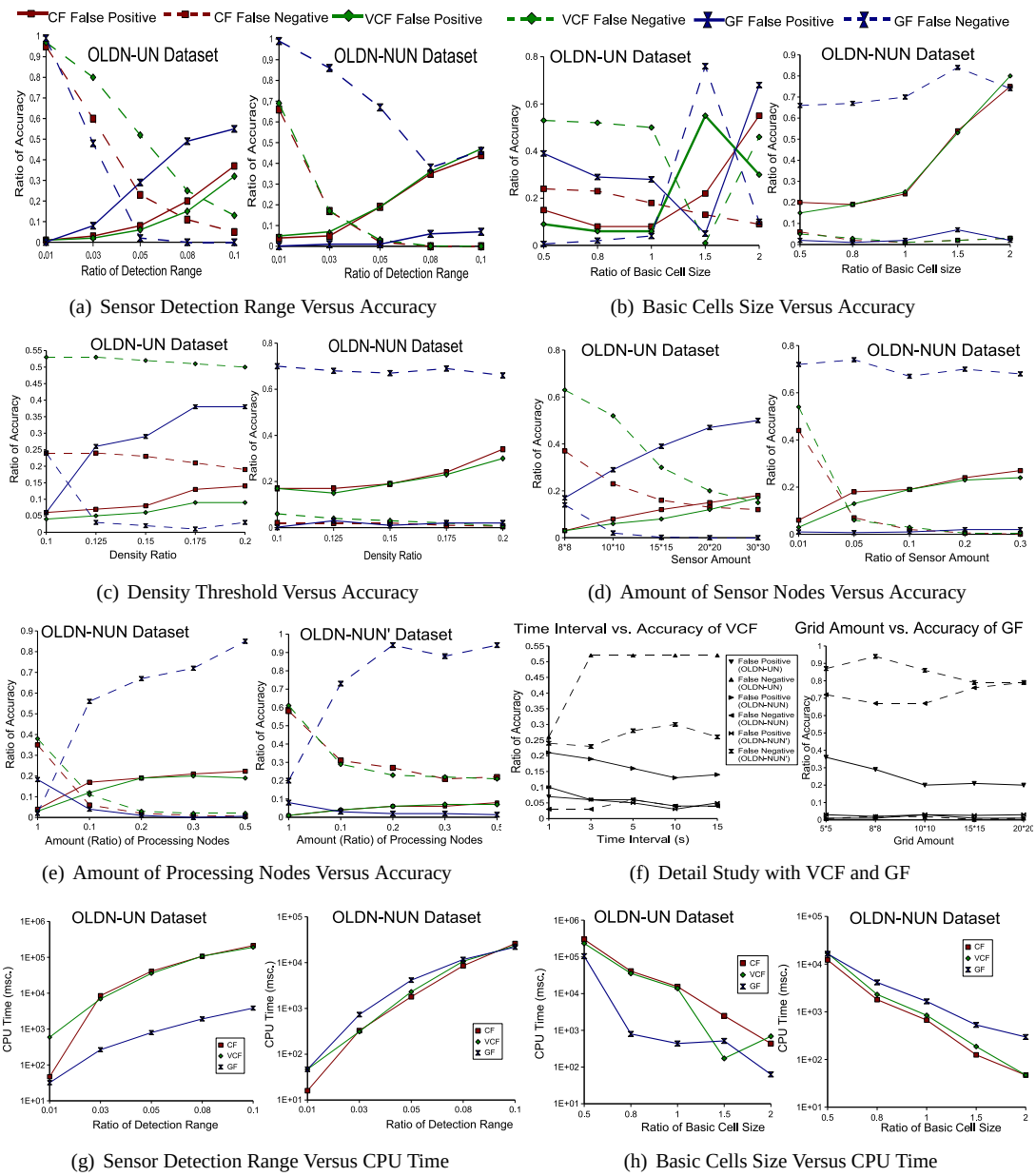
To simulate the density query, we assume that objects are moving during time instances 0 to 50, and at each time instance, we randomly select one processing node to send out the *AMOUNT* query. The sensor nodes then collect the result of the *AMOUNT* query based on the location of all the moving objects at the current time instance. For the *VCF* algorithm, the second *AMOUNT* query is sent out after μ time instances from the current time. Size of the basic cells are determined as the average length of the road network edges.

Experiments have been conducted to test the three density algorithms in terms of accuracy and CPU time. The CPU time is the sum of CPU running time of all the processing nodes. It reflects the total energy consumption for the density computation at the processing nodes. To compare the algorithms in terms of accuracy, we also implement the density query algorithm described in [8] and use the results of this implementation as the evaluation target. The implementation first constructs a 2D grid over the MBR of the whole road network so that the size of each grid cell is equal to the minimal area threshold α_1. We use the algorithm in [8] to find dense grid cells. The basic cells that are inside each dense grid cell are marked "dense" and are used to check the result of our proposed density query algorithms. If a basic cell is marked "dense" but is not found to be dense by our proposed algorithms, it is a *false negative*. If a basic cell is not marked "dense" but is found to be dense by our algorithms, it is a *false positive*. Suppose the implementation of [8] reports the amount of dense basic cells as D, we execute each of our proposed algorithms and collect the number of false positives P, false negatives N and report the ratio between these values and D.

To observe the algorithms under different settings, we tune the following parameters to report the accuracy ratios and CPU time of the algorithms: the sensor detection range, the size of basic cell, the density threshold value, the amount of sensor and processing nodes, the time interval μ for collecting the second query results for the *VCF* algorithm, and the grid cell amount for the *GF* algorithm. Note that the sensor detection range in real world settings may be a fixed value for the same type of sensors. Such values can be different for different sensor types. When the energy or surrounding conditions of a sensor change, this value may also change. Thus, we choose to tune this value to empirically study how it influences the results of the density algorithms. To report the result of each algorithm, we randomly pick 10 time instances during time period 0 to 50, send out the *AMOUNT* query from all the processing nodes, execute the algorithm at each processing node, and report the average result.

All experiments are performed on a Pentium IV 2.8 GHZ processor with 1 GB of main memory and running Windows XP. The C++ programming language is used.

Figure 6 shows the result of experiments. In the experiment with sensor detection range, we tune the range value as a ratio of the diagonal distance of the MBR of the road network. As illustrated in Figure 6(a), when the detection range is small, many dense areas can not be scanned by the sensors so that the amount of false negative is very big. When the detection range is increased, there are more dense areas found by the sensors so that the ratio



(a) Sensor Detection Range Versus Accuracy

(b) Basic Cells Size Versus Accuracy

(c) Density Threshold Versus Accuracy

(d) Amount of Sensor Nodes Versus Accuracy

(e) Amount of Processing Nodes Versus Accuracy

(f) Detail Study with VCF and GF

(g) Sensor Detection Range Versus CPU Time

(h) Basic Cells Size Versus CPU Time

Figure 6: Experimental Results of the Density Query Algorithms

of false negative is decreasing. However, the amount of false positive is increasing at the same time. In the uniform dataset, the *GF* algorithm returns less false negatives than the other two algorithms and is the first to achieve 0 false negatives when the detection range grows. The other two algorithms can not achieve 0 false negatives unless the detection ranges grows very big. In the non-uniform dataset, the accuracy ratio of *CF* and *VCF* algorithm is very close and they both achieve 0 false negatives very early when the detection range grows with a reasonable amount of false positives. Thus, when the distribution of sensor nodes is similar to the distribution of dataset, both *CF* and *VCF* perform better than *GF*.

In the subsequent experiments, we set the basic cells size as a ratio of the average edge length of the road network, the density threshold as the ratio of the amount of moving objects in a single basic grid cell, the ratio of sensor and processing node amount (in the right figure of Figure 6(d) as the percentage of sensor nodes against the total amount of road network nodes, and vary these parameters to observe the ratio of accuracy. Similar

observations can be found on these experiments. As illustrated in Figures 6(b) to 6(d), the accuracy ratio of the *GF* algorithm is very different from the other two algorithms. In the uniform dataset, the *GF* algorithm has less false negatives but more false positives than *CF* and *VCF* when the parameter value is very small. When the parameter value grows bigger, both the *CF* and the *VCF* algorithms have less false positives than *GF* in the uniform dataset. In the non-uniform dataset, the *CF* and *VCF* algorithms have much less false negatives and more false positives than *GF*.

Figure 6(f) shows a detailed study on the *VCF* and *GF* algorithms. For the *VCF* algorithm, when length of the time interval μ between the two queues $\mathcal{C}^a$ and $\mathcal{C}^b$ grows, locations of objects in $\mathcal{C}^b$ may be distant from their previous locations in $\mathcal{C}^a$. As a result, the *VCF* algorithm filters out more cells that are possibly dense which, in turn, increases false negatives and decreases false positives. For the *GF* algorithm, it is seen that only the false positives of the uniform dataset and the false negatives of the non-uniform datasets are influenced by the grid amount. When the grid becomes denser, the accuracy of the algorithm with uniform dataset becomes better. An interesting observation is that, when the grid amount grows, the false negative amount on the non-uniform dataset (OLDN–NUN) first decreases then increases while the amount on the OLDN–NUN' dataset first increases and then decreases. We believe this is due to the distribution and amount of sensor nodes at these two datasets (OLDN–NUN has 611 sensor nodes, OLDN–NUN' has 227 sensor nodes).

Figures 6(g) and 6(h) illustrate the effect of sensor detection range and basic cell size on CPU time. As expected, the CPU running time increases when the detection range grows and decreases when the size of basic cells becomes smaller. Another observation is that the CPU running time of *GF* algorithm is better than the others on uniform datasets but worse on non-uniform datasets.

To summarize, *GF* algorithm has the best performance when the sensor and processing nodes are uniformly distributed while *CF* and *VCF* algorithms are better if all nodes are non-uniformly distributed. In addition, experiments were also done on the trajectory data generated based on San Joaquin road network. The results of these experiments, not covered in detail here, are quite consistent to those reported and thus provide a further validation of our findings.

7 Conclusion

This paper addresses mobility related density query processing in sensor networks. We have assumed a two-level architecture with both location sensor nodes detecting moving objects and processing nodes issuing and answering density queries. For processing nodes, we have proposed three query processing algorithms: *CF*, *VCF* and *GF*. Extensive experimental evaluation results show that three algorithms in our solution are able to answer density queries with acceptably high accuracy within short CPU times. Those results also disclose the most appropriate scenario settings for different algorithms.

References

[1] T. Brinkhoff. A Framework for Generating Network-Based Moving Objects. In *GeoInformatica,* (6)2, pp. 153–180, 2002.

[2] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate Aggregation Techniques for Sensor Databases. In *Proc. ICDE*, pp. 449–460, 2004.

[3] A. Coman, M. A. Nascimento, and J. Sander. Exploiting Redundancy in Sensor Networks for Energy Efficient Processing of Spatiotemporal Region Queries. In *Proc. CIKM*, pp. 187–194, 2005.

[4] M. Ester, H. P. Kriegel, J. Sander, X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. KDD,* pp. 226–231, 1996.

[5] Q. Fang, F. Zhao, and L. Guibas. Counting Targets: Building and Managing Aggregates in Wireless Sensor Networks. *Technical Report* P2002–10298, Palo Alto Research Center, 2002.

[6] T. He, P. Vicaire, T. Yan, et. al. Achieving Real-Time Target Tracking UsingWireless Sensor Networks. In *IEEE Real Time Tech. and App. Symp.* pp. 37–48, 2006.

[7] M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. J. Tsotras. On-Line Discovery of Dense Areas in Spatio-Temporal Databases. In *Proc. SSTD,* pp. 306–324, 2003.

[8] C. S. Jensen, D. Lin, B. C. Ooi, and R. Zhang. Effective Density Queries on Continuously Moving Objects. In *Proc. ICDE,* pp. 71, 2006.

[9] P. Kalnis, N. Mamoulis, and S. Bakiras. On Discovering Moving Clusters in Spatio-temporal Data. In *Proc. SSTD,* pp. 364–381, 2005.

[10] Y. Kotidis. Snapshot Queries: Towards Data-Centric Sensor Networks. In *Proc. ICDE,* pp. 131–142, 2005.

[11] C. K. Lee, W.-C. Lee, B. Zheng, and J. Winter. Processing Multiple Aggregation Queries in Geo-Sensor Networks. In *Proc. DASFAA,* pp. 20–34, 2006.

[12] S. Madden and M. J. Franklin. Fjording the Stream: An Architecture for Queries over Streaming Sensor Data. In *Proc. ICDE,* pp. 555–566, 2002.

[13] S. Madden, R. Szewczyk, M. J. Franklin and D. Culler. Supporting Aggregate Queries over Ad-Hoc Wireless Sensor Networks. In *Proc. 4th IEEE Workshop on Mobile Computing Systems and Applications,* 2002.

[14] S. Xiang, H. B. Lim, and K. L. Tan. Impact of Multi-query Optimization in Sensor Networks. In *Proc. DMSN,* 2006.

[15] F. Zhao, J. Shin, and I. Reich. Information Driven Dynamic Sensor Collaboration for Target Tracking. In *IEEE Signal Processing Magazine,* (19)2, 61–72, 2002.