Snapshot Density Queries on Location Sensors

Xuegang Huang
Department of Computer Science
Aalborg University, Denmark
xghuang@cs.aau.dk

Hua Lu
Department of Computer Science
Aalborg University, Denmark
luhua@cs.aau.dk

ABSTRACT

Density queries are of practical importance in many mobility related applications. In this paper, we employ the location sensors, which are placed in a geographical area and can only detect the amount of objects moving in vicinity, to estimate the dense regions of objects. Our approach partitions the region of interest into subregions, and deploys in each subregion both location sensors and a processing node that issues and answers density queries. Three algorithms are proposed to process density queries on those processing nodes. Their accuracy and efficiency are empirically evaluated.

Categories and Subject Descriptors

H.2.8 [**DATABASE MANAGEMENT**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms

Keywords

Density queries, location sensors

1. INTRODUCTION

Cheap, tiny sensors are often widely distributed to collect information in vicinities and transmit the collected data to the query source through a wireless ad-hoc peer-to-peer network. From the database point of view, the data being collected, transmitted and processed in sensor networks form a database, namely sensor network database. Many types of queries and query processing techniques have been discussed in this context. This paper assumes an application scenario where location sensors 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. An example of such queries in the real world is "find regions that are smaller than 100 m^2 in area size but contain more than 10 vehicles."

Density query is an essential functionality in systems that monitor moving objects. Such systems often have an assumption that smart clients with positioning functionality (e.g., GPS) collect and send locations to a central server [5, 6]. The emergence of sensor networks brings an alternative for the traditional client/server architecture. We consider density query processing in a different setting where positions are not reported from moving objects but detected by location sensors. Based on a two-level framework of sensor network, the paper proposes three approaches for discovering density areas and empirically studies their accuracy and efficiency.

The study of aggregate query processing and mobile objects tracking in sensor networks [2, 4, 7] is very related to our discussion. The architecture of our solution is identical to the sensor network architecture described in [7]. But our discussion is on the high-level query processing techniques rather than the the low-level infrastructure issues [7]. Instead of focusing on the low level details of tracking and managing mobile targets in sensor networks [4], we assume that a location sensor can only count the amount of objects in a small range and aim at finding effective and efficient algorithms to estimate the dense regions of objects.

This paper assumes that a population of target objects move within a 2D space where a sensor network is deployed. These sensors can detect and count objects in vicinity, but cannot tell the exact locations of each detected object. Each sensor node have limited power as well as computation and storage capabilities. Each node is connected to other nodes in vicinity through a wireless network. A sensor node communicates with other nodes that are spatially distant through a multi-hop routing protocol.

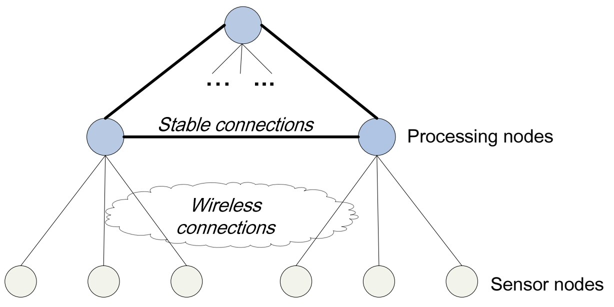


Figure 1: System Architecture

In addition to sensor nodes, a special type of nodes, *processing nodes*, issue and process queries. Sensor nodes collect data, send and route data to processing nodes. Processing nodes have abundant resource and stable, long-range connections to each other. The density query sent from the processing node is disseminated to sensor nodes in the vicinity of the processing node and also sent to other processing node through the stable connection. Results of the query are collected and summarized at these processing nodes and then reported the original one. Figure 1 illustrates the architecture described above.

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. We formalize the definitions of snapshot density and snapshot density query in the following.

Definition 1.1. **(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 $\mathcal{A}$.

Definition 1.2. **(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 $Density_t(\mathcal{A}_i) \geq \theta$ $(i = 1, 2, \ldots)$.

As the sensors cannot get the precise locations of moving objects, we will focus on algorithms that give estimated result of the density query. We consider the accuracy of such algorithms in terms of *false positives*, the error of not finding a dense pattern that does exist in the data, and *false negatives*, the error of finding a "dense pattern" that does not exist in the data. Our density query algorithms are supposed to reduce both of them.

2. ONLINE DENSITY COMPUTATION

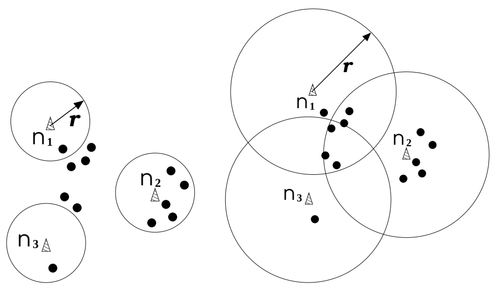
With our sensor network, snapshot density queries can be periodically initiated from a processing node. Processing a snapshot density query involves two parts, i.e., communication among processing nodes and query processing at each processing node. Steps involved in the first part is straightforward: 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 answer one type of query, denoted as $AMOUNT(r)$, which 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 its vicinity are $n_1, \ldots, n_m$, to process the density query, the node p first broadcasts the query $AMOUNT(r)$ to $n_1, \ldots, n_m$. 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 processing node uses a filtering algorithm to analyze these answers and return the dense regions. We proceed to introduce the filtering algorithms that find the dense regions.

2.1 Circular Filtering Approach

A 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



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

Figure 2: Accuracy with Different r Values

the filtering algorithm, the parameter r should be tuned to a value such that the reported regions of all sensor nodes cover the whole space but have very little intersection with each other. Since the value r is actually constrained by the technical configurations on the sensors, 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 $CF(\mathcal{C}, \theta, \alpha_1, \alpha_2)$
(2) sort $\mathcal{C}$ s.t. $c_i.amnt \geq c_{i+1}.amnt, i = 1, \ldots, m$-1
(3) $b \leftarrow$TRUE
(4) while $b$
(5)   $b \leftarrow$FALSE
(6)   for each $c_i \in \mathcal{C}$, if $c_i.valid$ :
(7)     if $\alpha_1 \leq AREA(c_i.shp) \leq \alpha_2 \wedge \frac{c_i.amnt}{AREA(c_i.shp)} \geq \theta$
(8)       $\mathcal{D} \leftarrow \mathcal{D} \cup \{c_i\}$
(9)       $b \leftarrow$ TRUE; $c_i.valid \leftarrow$ FALSE
(10)      for each $c_j \in \mathcal{C}, j \neq i$, if $c_j.valid$ :
(11)        if $c_j \cap c_i \neq \emptyset$
(12)          $c_j.shp \leftarrow c_j.shp - c_j.shp \cap c_i.shp$
(13) return $\mathcal{D}$
```

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 $\mathcal{C}$.

Since the snapshot density query is issued periodically, we propose an approach *VCF* (Varied Circular Filter) that lets each processing node send out the *AMOUNT(r)* queries with different r values at different time points and combine the collected observations. At time t_a, e.g., processing node p sends out query $AMOUNT(r_a)$ to the sensor nodes; at a later time t_b, the node p sends out query $AMOUNT(r_b)$ ($r_b < r_a$). Assuming that objects do not move very much during the period, we can combine the received results for density queries. Details of *VCF* are described in [3].

2.2 Grid Filtering Approach

Following the spirit of the grid-based approach in [5], 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 every circular region 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 $g_1.amnt = 0.6 \times 5 + 0.12 \times 4 = 3.48$.

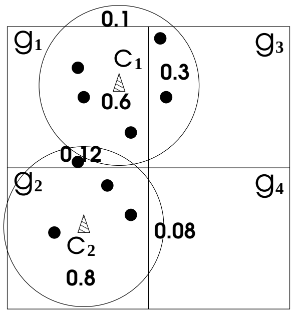


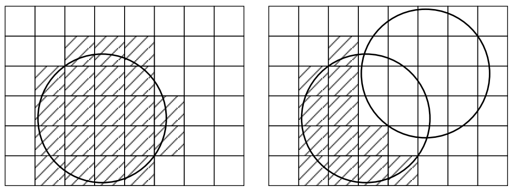
Figure 3: Example of Grid Filtering

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 $\mathcal{D}$
```

2.3 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 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.



(a) Example of Circle (b) Circle Subtraction

Figure 4: Tile-Based Solution

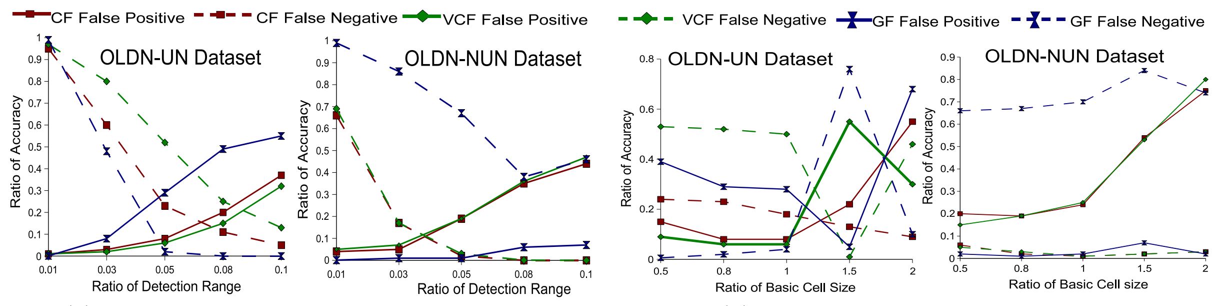
3. EMPIRICAL EVALUATION

We use Brinkhoff's network-based generator [1] to generate 10,000 moving object trajectories on the Oldenburg network (6105 nodes) and treat each trajectory as a unique object. We use integer as unit time instance and set the whole time period from 0 to 50. We generate both uniformly and non-uniformly distributed sensor and processing nodes based on the road networks. To generate the uniformly distributed nodes (denoted as OLDN–UN), 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 (denoted as OLDN–NUN) are generated by randomly choosing the road network nodes as the location of sensor or processing nodes.

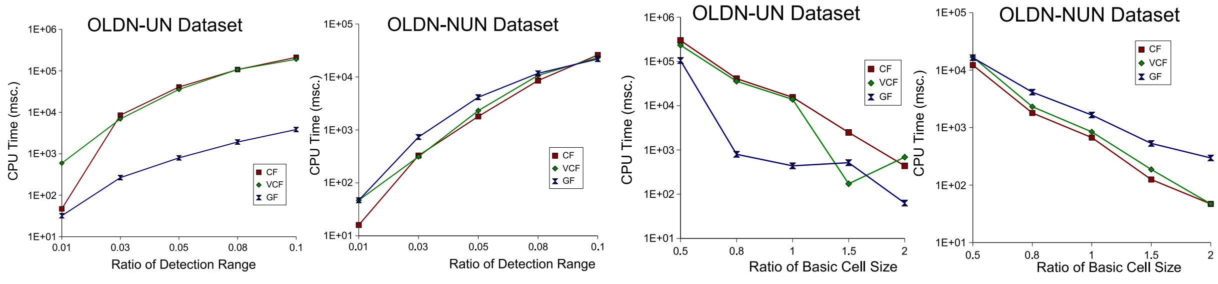
To simulate the density query, at each time instance from 0 to 50, we randomly select one processing node to send out the *AMOUNT* query. The sensor nodes then collect the result of the *AMOUNT* query based on the location of all the moving objects at the current time instance. For the *VCF* algorithm, 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 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 [6] 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 [6] reports the amount of dense basic cells as D, we execute each algorithm 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 the sensor detection range and the size of basic



(a) Sensor Detection Range Versus Accuracy (b) Basic Cells Size Versus Accuracy



(c) Sensor Detection Range Versus CPU Time (d) Basic Cells Size Versus CPU Time

Figure 5: Experimental Results of the Density Query Algorithms

cell. 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.

Figure 5 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 5(a), when the detection range is small, many dense areas cannot 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 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 cannot 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 next experiments, we set the basic cells size as a ratio of the average edge length of the road network. As illustrated in Figure 5(b), 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*.

Figures 5(c) and 5(d) illustrate the effect of sensor detection range and basic cell size on CPU time. As expected, the CPU time increases when the detection range grows and decreases when the size of basic cells becomes smaller. Another observation is that the CPU 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. More consistent experimental results, including those on the variation of node numbers, can be found in [3].

4. CONCLUSION

Assuming a two-level architecture with both location sensor nodes detecting moving objects and processing nodes issuing and answering density queries, we propose three density query processing algorithms. Experimental evaluation results disclose the performance and the most appropriate scenarios of these algorithms.

5. 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] X. Huang and H. Lu. Snapshot Density Queries on Location Sensors. A DB Technical Report, 2007: www.cs.aau.dk/DBTR/DBPublications/DBTR–21.pdf

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

[5] 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.

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

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