Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2013, Article ID 592890, 12 pages
http://dx.doi.org/10.1155/2013/592890

Research Article

Continuous Top-k Contour Regions Querying in Sensor Networks

Shangfeng Mo,[1,2,3] Hong Chen,[1,2] Cuiping Li,[1,2] Deying Li,[1,2] and Yinglong Li[1,2,3]

[1] *Key Laboratory of Data Engineering and Knowledge Engineering of MOE, Renmin University of China, Beijing 100872, China*
[2] *School of Information, Renmin University of China, Beijing 100872, China*
[3] *Hunan University of Science and Technology, Xiangtan 411201, China*

Correspondence should be addressed to Hong Chen; chong@ruc.edu.cn

Received 8 January 2013; Revised 24 February 2013; Accepted 9 March 2013

Academic Editor: Zhangbing Zhou

Copyright © 2013 Shangfeng Mo et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Wireless sensor networks (WSNs) are important parts of Internet of Things or Cyber-Physical Systems (CPS). WSNs can be seen as a new type of distributed database systems. The data query processing is very important for WSNs. In this paper, we proposed a Continuous Top-k Contour Regions Querying algorithm (CTCRQ) which can continuously obtain the top-k contour regions and does not lose the rate of precision (accuracy). We take full advantage of the kth value of top-k result in current round as the threshold to suppress the nodes whose readings do not belong to the top-k result in next round. Extensive experiments are conducted to evaluate the performance of the proposed CTCRQ approach by using a synthetic data set. The results provide a number of insightful observations and show that CTCRQ substantially outperforms Centralized algorithm, Centralized Optimized algorithm, and CCM algorithm in terms of data transmitted.

1. Introduction

With the development of microelectronics, embedded computing, and wireless communication technology, sensor hardware technology is also improved. Low-power, tiny sensor nodes can be integrated with information collection, data processing, wireless communication, and other functions [1]. Wireless sensor networks (WSNs) composed by a large number of sensor nodes are used to collect and process information of perceived objects.

The sensor nodes are usually battery-powered and deployed in harsh physical environments. It is usually impossible to replace the batteries or the nodes. So the goal of querying in wireless sensor networks (WSNs) is to reduce the energy consumption and prolong the network lifetime. Compared with the calculation, the communication between sensor nodes consumed much more energy. For example, executing one instruction needs energy consumption about 0.84 nJ, but the energy consumption of transmitting a sensory data packet is about 0.685 mJ between MICA2 sensor nodes [2]. So the key problem of saving the energy consumption is to reduce the amount of data transmission.

There are many energy-efficient queries in WSNs, for example, top-k querying [3–8], contour regions querying [9–15], aggregate querying [16], event querying [17, 18], and so forth. The traditional continuous top-k querying in WSNs can return the list of k sensor nodes with the highest (or lowest) readings at every sampling period.

To visualize the sensor network regions, we can use the contour mapping. A contour map of an attribute (e.g., temperature) displays the distribution of the attribute value over the topographic regions. Figure 1 shows the contour regions of temperatures in a real volcanic area, Kawah Ijen crater lake [3]. The sink can detect and analyze the environmental events using the contour regions.

There are many continuous contour regions querying schemes for WSNs, including eScan [9], isoline aggregation [10], Iso-Map [11, 12], CCM [13], the literature in [14], and improved Isoline aggregation [15]. These schemes will obtain the overall contour mapping regions. Most of these protocols use approximate algorithms to reduce the data transmitted but may lose the rate of precision (accuracy). In other words, these algorithms obtain approximate contour mapping regions.

It is difficult to achieve the overall contour mapping regions and not lose the rate of precision (accuracy), because sensor nodes have constrained resources and insufficient knowledge. If top-k highest (or lowest) contour regions can satisfy the user's requirements, it will significantly reduce the data transmitted and then save a great number of energy as well as prolong the network lifetime. In practice, as shown in Figure 2, scientists wish to concentrate on studying the most important environmental events, and they can continuously obtain the top-k contour regions with the highest (or lowest) temperatures at every sampling period. In addition, in some specific applications, people are often interested in the most important regions in the network. For example, Traffic Radio Station wants to find the most congested regions to obtain the overall traffic situation and then broadcast the traffic conditions to drivers in a city; scientists need to find the most dense regions of animals gathering in a forest to learn about the animals' living habit status; farmers hope to find the most arid regions in a farm to give priority to irrigate; museum curator wants to know the most dense showcases which visitors gathered to determine the tour route.

In this paper, we focus on the top-k highest (or lowest) contour regions, not the overall contour regions. But the top-k highest (or lowest) contour regions are accurate, not approximate. Our contributions are summarized as follows.

(i) Compared with obtaining the approximate overall contours region, the amount of data transmitted of our obtaining top-k highest (or lowest) contour regions algorithm is less.

(ii) We take full advantage of the kth value of top-k result in current round as the threshold to suppress the nodes whose readings do not belong to the top-k result in next round.

(iii) Extensive experiments are conducted to evaluate the performance of the proposed CTCRQ approach by using a synthetic data set. The results provide a number of insightful observations and show that CTCRQ substantially outperforms Centralized algorithm, Centralized Optimized algorithm, and CCM algorithm in terms of amount of Kbytes (kilo bytes) data transmitted.

The remainder of this paper is organized as follows. Section 2 summarizes related work. Section 3 introduces some assumptions. Section 4 describes the proposed CTCRQ scheme in detail. Section 5 presents experimental results. Section 6 concludes the paper.

2. Related Work

We have studied many querying algorithms in WSNs, but we have not found a Continuous Top-k Contour Regions Querying algorithm as we proposed. There are many top-k querying algorithms [3–8] and continuous contour regions querying schemes for WSNs [9–13]. We have selected part of them to analyze their core mechanisms, characteristics, advantages, and disadvantages.

First, let us introduce some top-k querying algorithms.

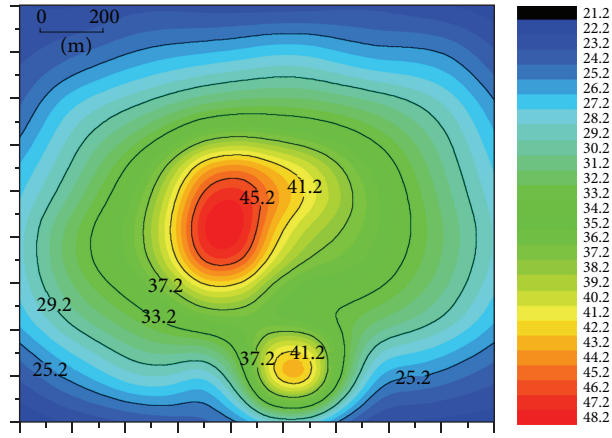


Figure 1: Contour regions of Kawah Ijen crater lake.

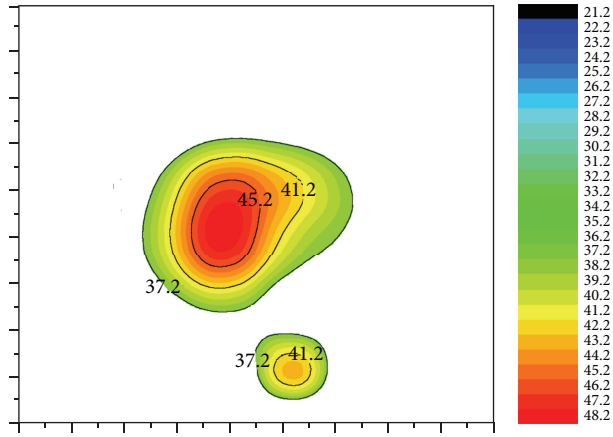


Figure 2: Top-k contour regions of Kawah Ijen crater lake.

In POT [3] protocol, they classify sensor nodes into a number of Partial Ordered Trees (POT), and the sink maintains the global ranking list (GR). POT protocol is useful for the occasion that top-k results are spatially correlated. When the top-k results are not spatially correlated, FILA [4] protocol outperforms POT.

In FILA [4] protocol, at every sampling point, if the new sensed data of a node does not change beyond the filtering window, the data will not be sent to the sink. If the sensed data comes into the filtering window of other nodes, the sink will broadcast to all nodes in the WSNs and acquire the needed data. In FILA, the transmission of data is discrete.

In PRIM [5] and PRIM-c [6] protocols, there are N partial TDMA frames in the TDMA schedule for collecting sensor readings. The sensed data are sent at different frames based on different values, and the higher sensor data can be sent to the sink in the more previous frames. The protocols are typical methods to save energy at the expense of time.

In XP [7] protocol, the authors construct a new routing structure in a bottom-up, spatially clustered fashion (called cluster tree) for the cross-pruning (XP) framework. Because the aggregation node will broadcast a filtering threshold to

its remaining children, it will consume more energy additionally.

Second, we will show some continuous contour regions querying protocols.

eScan [9] focuses on monitoring the remaining energy information of nodes in wireless sensor networks. The sensor nodes will report their remaining energy via a data collection tree. When the data is being sent back to the sink, intermediate nodes will aggregate the information as it flows. If the nodes are geographically adjacent and their readings are in the same value range, the aggregation may be done. Data is aggregated into polygons of similar value.

In isoline aggregation [10] protocol, each node needs to broadcast its reading to its neighbors. When a node receives the readings of all neighbors, the node will compare its reading with the readings of all neighbors. If the readings lie in different sides of an isoline, then a report needs to be generated. Reported isoline consists of the reading of one node and the readings of its neighbors whose readings come across the isoline.

In Iso-Map [11, 12] protocol, they proposed a parameter gradient direction based on the isoline aggregation [10] protocol. When all isoline nodes send their 3-tuple to the sink, the sink can construct the contour map based on the received 3-tuple. 3-tuple includes the isolevel of the node, position of the node, gradient direction of the node. The literature in [12] is an expanded version of the Iso-Map [11].

In CCM [13] protocol, each node maintains a CN-array structure, where 1 or 0 bit information of s_i's one-hop neighbors is saved, sequenced in counterclockwise cyclic order around the node s_i. If the reading of node s_i and the reading of its neighbor fall into the same level, s_i uses 0 to represent the reading of its neighbor in CN-array, otherwise s_i uses 1 to represent it. Each node updates its CN-array after receiving neighboring node broadcasts. Only a few contour nodes need to report their readings and CN-arrays to the sink and suppress their neighbors.

In the literature [14], a group of mobile data collecting nodes are deployed. The sensors are mounted on mobile objects so that they can be located in sample positions within target areas. Then the nodes emit signal vertically towards an upper reference plane. By detecting the returned wave, the receiver will work out the correct distance. Finally, an algorithm is applied on all the collected samples to plot the contour map. In this literature, many mobile nodes are deployed. While in our application scenario, the sensor nodes are stationary.

3. Preliminaries

In this paper, there are N sensor nodes constituting a network by self-organized manner. The sensor nodes sample the data periodically. Each sampling period is called a round. The sink node continuously requests the list of top-k contour regions with the highest (or lowest) contour level in every sampling period. The ith sensor node is denoted by s_i and the corresponding sensor nodes set $S = \{s_1, s_2, \ldots, s_n\}$, $|S| = N$.

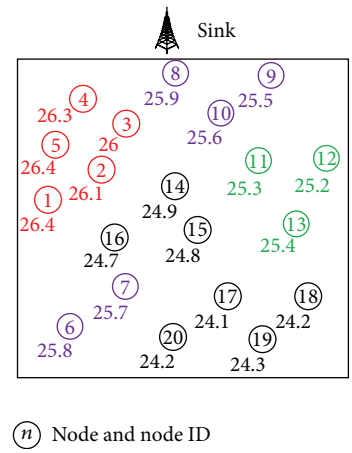


Figure 3: Temperature of each node in the network.

We make the following assumptions.

(1) All ordinary sensor nodes are homogeneous and have the same capabilities. The communication radiuses of all ordinary nodes are the same. When all nodes are deployed, they will be stationary, and each one has a unique identification (ID).

(2) There is only one sink (base station), and the sink node can be recharged.

(3) Links are symmetric. If node s_i can communicate with node s_j, node s_j can also communicate with node s_i.

(4) The energy resource of ordinary sensor nodes is highly limited and unreplenished.

4. The CTCRQ Scheme

The temporal data correlation [19, 20] means that the sensed readings are quite similar during a short period of time. We can use the temporal data correlation to reduce the number of data transmitted. If the data in current round is the same as the data in the last round, the data do not need to be sent to the sink repeatedly, which can save the energy consumption, as well as extend the network lifetime. If the data of a node is unlikely to belong to the top-k regions, the data will also be filtered.

4.1. Definitions. The idea of a normalized mechanism (which is the same as quantization of SENS-Join [21]) is to approximate a continuous range of values by a relatively small set of discrete values.

Definition 1. The whole range of an attribute value can be bounded using [**min_value**, **max_value**). Figure 3 displays the detailed temperature of each node in the network, and the whole range of the temperature is [0, 100).

Definition 2. A reading of a node belongs to a value subrange. A value subrange is denoted by [*LowerB, UpperB*), which means the lower bound and upper bound, respectively. Parameter **step** indicates the difference between *LowerB* and

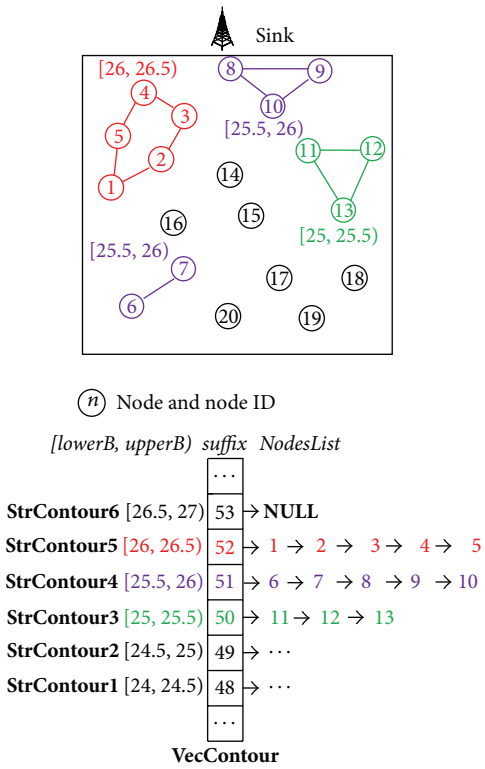


FIGURE 4: top-3 contour (polygon) regions.

UpperB. **step** = *UpperB* − *LowerB*. As shown in Figure 4, there is a temperature value sub-range [26.5, 27.0) and the **step** is 0.5.

Definition 3. We use a parameter *suffix* to express the normalized result value. *suffix* = *floor((value* −**min_value**)/**step**). The function *floor(A)* returns the nearest integer which is less than or equal to *A*. For example, $floor(5/4) = 1$. If the temperature is 26.2, the normalized result value *suffix* is $floor((26.2 - 0)/0.5) = 52$, which can be expressed by 1 byte. We have defined the whole range of an attribute value as **[min_value, max_value)**. The whole normalized range of an attribute value will be **[min_suffix, max_suffix)**. Thus the continuous real readings can be expressed by discrete integers.

Definition 4. The contour regions can be expressed by a structure **StrContour**, which is described as follows:

```
struct StrContour
{double LowerB; // lower bound.
 double UpperB; // upper bound.
 int suffix;
 list <int> NodesList; // node id list.
};
```

The parameter *NodesList* means the list of node id. If the number of nodes in the *NodesList* is 0, the *NodesList* will be denoted by **NULL**. If the nodes have the same *suffix*, they can be aggregated into the same *NodesList*. As shown in Figure 4, the *NodesList* of **StrContour6** has no nodes, which is denoted by **NULL**. The attribute value sub-range [*LowerB*, *UpperB*) of **StrContour5** is [26.0, 26.5), and the corresponding *suffix* is 52. The *suffix*es of nodes 1, 2, 3, 4, and 5 are the same, and these nodes are linked to the *NodesList* of **StrContour5**.

Definition 5. The total contour regions of the network can be expressed by a vector **VecContour**. Vector is implemented using dynamic array. The element of vector **VecContour** is the structure **StrContour**. As shown in Figure 4, the **VecContour** includes 6 valid **StrContour**, which are **StrContour1**, **StrContour2**,. . ., and **StrContour6**, respectively.

If the *NodesList* ofa **StrContour** is not **NULL**, the **StrContour** denotes a contour (polygon) region. The adjacent nodes in the same *NodesList* are interconnected to form one or more than one contour subregions. As shown in the top subfigure of Figure 4, we use three kinds of solid lines of different colors to represent the top-3 contour (polygon) regions. The corresponding **StrContours** are **StrContour5, StrContour4**, and **StrContour3**, whose value subranges are [26.0, 26.5), [25.5, 26.0), and [25.0, 25.5), respectively. The *NodesList* of **StrContour5** forms 1 contour subregion, which is denoted by red lines. The *NodesList* of **StrContour4** forms 2 contour subregions, which are denoted by purple lines. The *NodesList* of **StrContour3** forms 1 contour subregion, which is denoted by green lines.

4.2. The Detailed CTCRQ Scheme. Based on the above discussions and analyses, we proposed a Continuous Top-k Contour Regions Querying (CTCRQ) algorithm, which means continuously obtaining top-k **StrContours** with the highest (or lowest) *suffix* of the attribute value. In CTCRQ, if the reading of a node is not out of the value sub-range, the node will not report its reading to the sink in next round.

As shown in Figure 5, it is the flow chart of the sink. At the beginning of each round, the sink waits and receives the reported values from sensor nodes. Then the sink calculates the top-k results. If the number of the top-k results is greater than or equal to *k*, the sink broadcasts **M_NEXT_ROUND_BEGIN**(*ThresholdSuffix*) message and exits. Otherwise, the sink broadcasts a **M_PROBE**(*ThresholdSuffix*, *OldThresholdSuffix*) message to all nodes in the network. Then, the sink waits and receives the reported values from sensor nodes. After that, the sink calculates the top-k results and broadcasts **M_NEXT_ROUND_BEGIN**(*ThresholdSuffix*) message and exits.

The detailed algorithms are shown in Algorithm 1 (*Sink algorithm*) and Algorithm 2 (*Sensor node algorithm*).

As shown in Algorithm 1, in the initialize phase, the threshold suffix *ThresholdSuffix* is equal to **min_suffix**. When the sink receives a data message which includes one or more than one sensor node's data information, the sink finds the old location of each node in the *NodesList* of **StrContour** based on *suffix* and deletes it then links to the new location in *NodesList* of other **StrContour**. Then the sink sorts **StrContours** based on *suffix* in the **VecContour**.

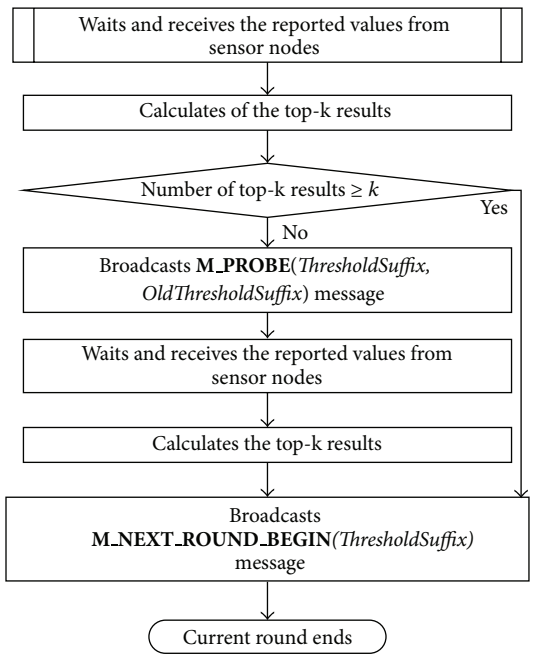


Figure 5: The flow chart of the sink.

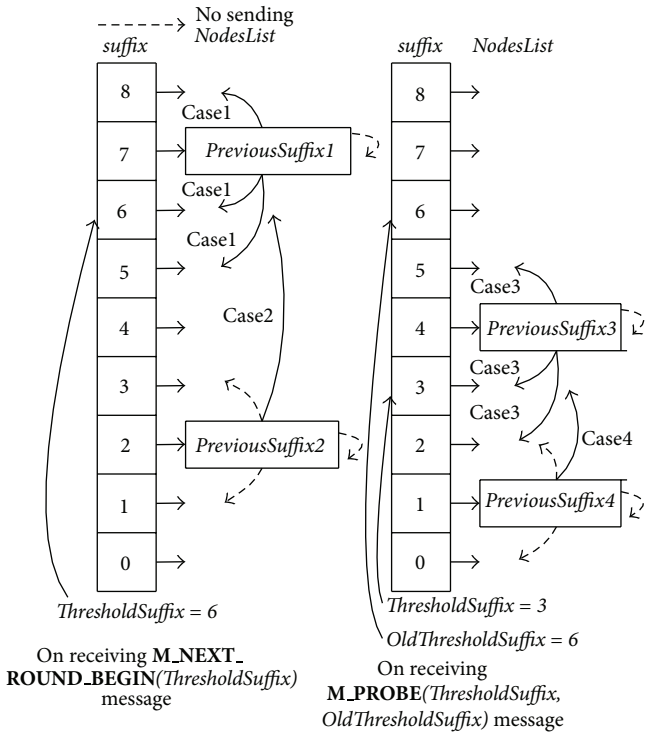


Figure 6: The cases of sending data to the sink.

From **max_suffix** to *ThresholdSuffix*, the sink calculates the top-k **StrContour**s (contour regions) whose *NodesList* is not **NULL** based on *suffix*. If the number of top-k results is greater than or equal to k, the sink sets the *ThresholdSuffix* to be the *suffix* of the kth **StrContour**s. The *suffix* of the kth **StrContour**s is denoted by *TopkSuffix*. After that the sink broadcasts an **M_NEXT_ROUND_BEGIN**(*ThresholdSuffix*) message to all nodes in the network to prepare for next round sampling. The threshold suffix *ThresholdSuffix* is contained in this message. The current round terminates. If the number of **StrContour**s is less than k, it will enter the probing phase. The sink sets *OldThresholdSuffix* to be *ThresholdSuffix*. From *ThresholdSuffix* to **min_suffix**, if the **StrContour** has one or more than one data updated, the sink adds this **StrContour** to the top-k result set (contour regions). If the number of result sets is greater than or equal to k, the sink sets the *ThresholdSuffix* to be *TopkSuffix*, otherwise sets the *ThresholdSuffix* to be **min_suffix**. The sink broadcasts an **M_PROBE**(*ThresholdSuffix*, *OldThresholdSuffix*) message to all nodes in the network and waits for a time period which guarantees the required data can be collected. When the time period expired, the sink calculates the top-k results and broadcasts **M_NEXT_ROUND_BEGIN**(*ThresholdSuffix*) message and exits.

Next, we will describe the sensor node algorithm which is shown in Algorithm 2. The parameter *PreviousSuffix* preserves the previous *suffix* of last round in which the sensor node sends data to the sink. In the initialize phase, *PreviousSuffix* is set to be −1.

When the sensor node receives an **M_NEXT_ROUND_BEGIN**(*ThresholdSuffix*) message, which means the beginning of the next round, the sensor node begins to sample the sensory attribute value and calculates *CurrentSuffix* based on the attribute value. The data (readings) of a sensor node will be sent to the sink in the following 2 cases.

Case 1. If *CurrentSuffix* is not equal to *PreviousSuffix*, and *PreviousSuffix* is greater than or equal to *ThresholdSuffix*, the sensor node sets *PreviousSuffix* to be *CurrentSuffix* and sends the data to the sink. As shown in Figure 6 on the left part of the figure, *ThresholdSuffix* is 6. The dotted line denotes no data sending. When *PreviousSuffix1* is 7 and *CurrentSuffix* is equal to 8, 6, or 5, the sensor node will send the data to the sink.

Case 2. If *CurrentSuffix* is not equal to *PreviousSuffix*, *PreviousSuffix* is less than *ThresholdSuffix* and *CurrentSuffix* is greater than or equal to *ThresholdSuffix*, the sensor node sets *PreviousSuffix* to be *CurrentSuffix* and sends the data to the sink. As shown in Figure 6 on the left part of the figure, *ThresholdSuffix* is 6. When *PreviousSuffix2* is 2 and *CurrentSuffix* is 6, the sensor node will send the data to the sink.

When the sensor node receives an **M_PROBE**(*ThresholdSuffix*, *OldThresholdSuffix*) message, the data (readings) of a sensor node will be sent to the sink in the following 2 cases.

Case 3. If *CurrentSuffix* is not equal to *PreviousSuffix*, *PreviousSuffix* is greater than or equal to *ThresholdSuffix* and less than *OldThresholdSuffix* and *CurrentSuffix* is less than *OldThresholdSuffix*, the sensor node sets *PreviousSuffix* to be *CurrentSuffix* and sends the data to the sink. As shown in Figure 6 on the right part of the figure, *ThresholdSuffix* is 3 and *OldThresholdSuffix* is 6. When *PreviousSuffix3* is 4 and

```
(1) initialize
(2)    ThresholdSuffix = min_suffix;
(3) end-initialize
Main program:
(4) Waits and receives the reported values from sensor nodes;
(5) for (each node which sends data to the sink)
(6)    Finds old location in NodesList of StrContour based on suffix and deletes it;
(7)    Links to the new location in NodesList of other StrContour;
(8) end-for
(9) Sorts StrContour based on suffixin the VecContour;
(10) for (max_suffix to ThresholdSuffix)
(11)    Calculates the top-k results;
(12) end-for
(13) if (number of top-k results) ⩾ k
(14)    ThresholdSuffix = TopkSuffix; // The suffix of the kth StrContour.
(15)    Broadcasts M_NEXT_ROUND_BEGIN(ThresholdSuffix) message and exits;
(16) else // need to probe
(17)    OldThresholdSuffix = ThresholdSuffix;
(18)    for (ThresholdSuffix to min_suffix)
(19)       if data updated then updates top-k results; end-if
(20)    end-for
(21)    if (number of top-k results) ⩾ k
(22)       ThresholdSuffix = TopkSuffix; // TopkSuffix has updated.
(23)    else ThresholdSuffix = min_suffix;
(24)    end-if
(25)    Broadcasts M_PROBE(ThresholdSuffix, OldThresholdSuffix) message;
(26)    Waits and receives the reported values from sensor nodes;
(27)    Repeats the steps of lines 5–15;
(28) end-if
```

Algorithm 1: Sink algorithm.

CurrentSuffix is 5, 3, or 2, the sensor node will send the data to the sink.

Case 4. If *CurrentSuffix* is not equal to *PreviousSuffix*, *PreviousSuffix* is less than *ThresholdSuffix* and *CurrentSuffix* is greater than or equal to *ThresholdSuffix* and less than *OldThresholdSuffix*, the sensor node sets *PreviousSuffix* to be *CurrentSuffix* and sends the data to the sink. As shown in Figure 6 on the right part of the figure, *ThresholdSuffix* is 3 and *OldThresholdSuffix* is 6. When *PreviousSuffix4* is 1 and *CurrentSuffix* is 3, the sensor node will send the data to the sink.

4.3. Theorem

Theorem 6. *The described CTCRQ algorithm correctly reports the top-k* **StrContour**s *(contour regions) result set ordered on their suffix.*

Proof. In each round, the final top-k result set will be obtained with probing phase or without probing phase.

If the final top-k result set is obtained without probing phase, each sensor node only receives the **M_NEXT_ROUND_BEGIN** message, as shown in Figure 6 on the left part of the figure. *FS* denotes the final top-k result set:

$$FS = \{s_i \mid s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\}_{\text{top-k}}. \quad (1)$$

When the sensor node receives an **M_NEXT_ROUND_BEGIN**(*ThresholdSuffix*) message, the data will be sent to the sink in Cases 1 and 2, as shown in Algorithm 2.

SS denotes the sending data set by sensor nodes.

Case 1

$$\begin{aligned} &SS\text{-}case\ 1 \\ &\quad = \{s_i \mid s_i \cdot PreviousSuffix \geqslant ThresholdSuffix\} \\ &\quad \supset \{s_i \mid s_i \cdot PreviousSuffix \geqslant ThresholdSuffix \\ &\qquad \wedge s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\}. \end{aligned} \quad (2)$$

Case 2

$$\begin{aligned} &SS\text{-}case\ 2 \\ &\quad = \{s_i \mid s_i \cdot PreviousSuffix < ThresholdSuffix \\ &\qquad \wedge s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\} \\ &(SS\text{-}case\ 1 \cup SS\text{-}case\ 2) \\ &\quad = (\{s_i \mid s_i \cdot PreviousSuffix \geqslant ThresholdSuffix\} \\ &\qquad \cup \{s_i \mid s_i \cdot PreviousSuffix < ThresholdSuffix \\ &\qquad \wedge s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\}) \end{aligned} \quad (3)$$

```
(1) initialize
(2)     int PreviousSuffix = −1; // preserve the previous suffix.
(3) end-initialize
(4) On receiving M_NEXT_ROUND_BEGIN(ThresholdSuffix) message;
(5)    Begin sampling;
(6)    CurrentSuffix = floor((value − min_value)/step); // floor(A) returns the nearest integer which ⩽A.
(7)    if (CurrentSuffix ! = PreviousSuffix)
(8)       if PreviousSuffix ⩾ ThresholdSuffix // belongs to the top-k contour regions in last round. // Case 1
(9)          PreviousSuffix = CurrentSuffix;
(10)          Sends the data to the sink;
(11)       else // does not belongs to the top-k contour regions in last round.
(12)          if (CurrentSuffix >= ThresholdSuffix) // Case 2
(13)             PreviousSuffix = CurrentSuffix;
(14)             Sends the data to the sink;
(15)          end-if
(16)       end-if
(17)    end-if
(18) end processing M_NEXT_ROUND_BEGIN message;
(19) On receiving M_PROBE(ThresholdSuffix, OldThresholdSuffix) message;
(20)    if (CurrentSuffix ! = PreviousSuffix)
(21)       if ((PreviousSuffix >= ThresholdSuffix) && (PreviousSuffix < OldThresholdSuffix)
                && (CurrentSuffix < OldThresholdSuffix)) // Case 3
(22)          PreviousSuffix = CurrentSuffix;
(23)          Sends the data to the sink;
(24)       else
(25)          if ((PreviousSuffix < ThresholdSuffix) && (CurrentSuffix >= ThresholdSuffix)
                 && (CurrentSuffix < OldThresholdSuffix)) // Case 4
(26)             PreviousSuffix = CurrentSuffix;
(27)             Sends the data to the sink;
(28)          end-if
(29)       end-if
(30)    end-if
(31) end processing M_PROBE message;
```

Algorithm 2: Sensor node algorithm.

$$\begin{aligned}
&\supset \left(\{s_i \mid s_i \cdot PreviousSuffix \geqslant ThresholdSuffix\right.\\
&\quad \wedge s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\}\\
&\quad \cup \{s_i \mid s_i \cdot PreviousSuffix < ThresholdSuffix\\
&\qquad \wedge s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\}\left.\right)\\
&= \{s_i \mid \ s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\}\\
&\supset \{s_i \mid s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\}_{\text{top-k}} = FS.
\end{aligned} \tag{4}$$

Hence, the final top-k result set FS is a part of the sending data set ($SS\text{-}case\ 1 \cup SS\text{-}case\ 2$).

If the final top-k result set is obtained with probing phase, each sensor node receives the **M_NEXT_ROUND_BEGIN** and **M_PROBE** messages, as shown in Figure 6 on the right part of the figure. FS denotes the final top-k result set:

$$FS = \{s_i \mid s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\}_{\text{top-k}}. \tag{5}$$

If the sensor node receives an **M_PROBE**(*ThresholdSuffix*, *OldThresholdSuffix*) message, the data will be sent to the sink in Cases 3 and 4, as shown in Algorithm 2.

Case 3

$SS\text{-}case\ 3$

$$\begin{aligned}
&= \{s_i \mid s_i \cdot PreviousSuffix \geqslant ThresholdSuffix\\
&\qquad \wedge s_i \cdot PreviousSuffix < OldThresholdSuffix\\
&\qquad \wedge s_i \cdot CurrentSuffix < OldThresholdSuffix\}\\
&\supset \{s_i \mid s_i \cdot PreviousSuffix \geqslant ThresholdSuffix\\
&\qquad \wedge s_i \cdot PreviousSuffix < OldThresholdSuffix\\
&\qquad \wedge s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\\
&\qquad \wedge s_i \cdot CurrentSuffix < OldThresholdSuffix\}.
\end{aligned} \tag{6}$$

Case 4

$SS\text{-}case\ 4$

$$\begin{aligned}
&= \{s_i \mid s_i \cdot PreviousSuffix < ThresholdSuffix\\
&\qquad \wedge s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\\
&\qquad \wedge s_i \cdot CurrentSuffix < OldThresholdSuffix\}
\end{aligned} \tag{7}$$

$$
\begin{aligned}
&(SS\text{-}case\ 3 \cup SS\text{-}case\ 4)\\
&\supset (\{s_i \mid s_i \cdot PreviousSuffix \geqslant ThresholdSuffix\\
&\quad \wedge s_i \cdot PreviousSuffix < OldThresholdSuffix\\
&\quad \wedge s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\\
&\quad \wedge s_i \cdot CurrentSuffix < OldThresholdSuffix\}\\
&\quad \cup \{s_i \mid s_i \cdot PreviousSuffix < ThresholdSuffix\\
&\quad\quad \wedge s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\\
&\quad\quad \wedge s_i \cdot CurrentSuffix < OldThresholdSuffix\})\\
&= \{s_i \mid s_i \cdot PreviousSuffix < OldThresholdSuffix\\
&\quad \wedge s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\\
&\quad \wedge s_i \cdot CurrentSuffix < OldThresholdSuffix\}.
\end{aligned} \tag{8}
$$

The following is based on the formula (2).

$$
\begin{aligned}
&SS\text{-}case\ 1\\
&= \{s_i \mid s_i \cdot PreviousSuffix \geqslant OldThresholdSuffix\}\\
&\supset (\{s_i \mid s_i \cdot PreviousSuffix \geqslant OldThresholdSuffix\\
&\quad \wedge s_i \cdot CurrentSuffix \geqslant OldThresholdSuffix\}\\
&\quad \cup \{s_i \mid s_i \cdot PreviousSuffix \geqslant OldThresholdSuffix\\
&\quad\quad \wedge s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\\
&\quad\quad \wedge s_i \cdot CurrentSuffix < OldThresholdSuffix\})
\end{aligned} \tag{9}
$$

$$
\begin{aligned}
&(SS\text{-}case\ 1 \cup SS\text{-}case\ 3 \cup SS\text{-}case\ 4)\\
&\supset (\{s_i \mid s_i \cdot PreviousSuffix \geqslant OldThresholdSuffix\\
&\quad \wedge s_i \cdot CurrentSuffix \geqslant OldThresholdSuffix\}\\
&\quad \cup \{s_i \mid s_i \cdot PreviousSuffix \geqslant OldThresholdSuffix\\
&\quad\quad \wedge s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\\
&\quad\quad \wedge s_i \cdot CurrentSuffix < OldThresholdSuffix\})\\
&\quad \cup \{s_i \mid s_i \cdot PreviousSuffix < OldThresholdSuffix\\
&\quad\quad \wedge s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\\
&\quad\quad \wedge s_i \cdot CurrentSuffix < OldThresholdSuffix\}\\
&= (\{s_i \mid s_i \cdot PreviousSuffix \geqslant OldThresholdSuffix\\
&\quad\quad \wedge s_i \cdot CurrentSuffix \geqslant OldThresholdSuffix\}\\
&\quad \cup \{s_i \mid s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\\
&\quad\quad \wedge s_i \cdot CurrentSuffix < OldThresholdSuffix\}).
\end{aligned} \tag{10}
$$

The following is based on the formula (3).

$$
\begin{aligned}
&SS\text{-}case\ 2\\
&= \{s_i \mid s_i \cdot PreviousSuffix < OldThresholdSuffix\\
&\quad \wedge s_i \cdot CurrentSuffix \geqslant OldThresholdSuffix\}
\end{aligned} \tag{11}
$$

$$
\begin{aligned}
&(SS\text{-}case\ 1 \cup SS\text{-}case\ 2 \cup SS\text{-}case\ 3 \cup SS\text{-}case\ 4)\\
&= (SS\text{-}case\ 1 \cup SS\text{-}case\ 3 \cup SS\text{-}case\ 4)\\
&\cup SS\text{-}case\ 2\\
&\supset (\{s_i \mid s_i \cdot PreviousSuffix \geqslant OldThresholdSuffix\\
&\quad \wedge s_i \cdot CurrentSuffix \geqslant OldThresholdSuffix\}\\
&\quad \cup \{s_i \mid s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\\
&\quad\quad \wedge s_i \cdot CurrentSuffix < OldThresholdSuffix\})\\
&\cup \{s_i \mid s_i \cdot PreviousSuffix < OldThresholdSuffix\\
&\quad \wedge s_i \cdot CurrentSuffix \geqslant OldThresholdSuffix\}\\
&= (\{s_i \mid s_i \cdot CurrentSuffix \geqslant OldThresholdSuffix\}\\
&\quad \cup \{s_i \mid s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\\
&\quad\quad \wedge s_i \cdot CurrentSuffix < OldThresholdSuffix\})\\
&= \{s_i \mid s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\}\\
&\supset \{s_i \mid s_i \cdot CurrentSuffix \geqslant ThresholdSuffix\}_{\text{top-k}} = FS.
\end{aligned} \tag{12}
$$

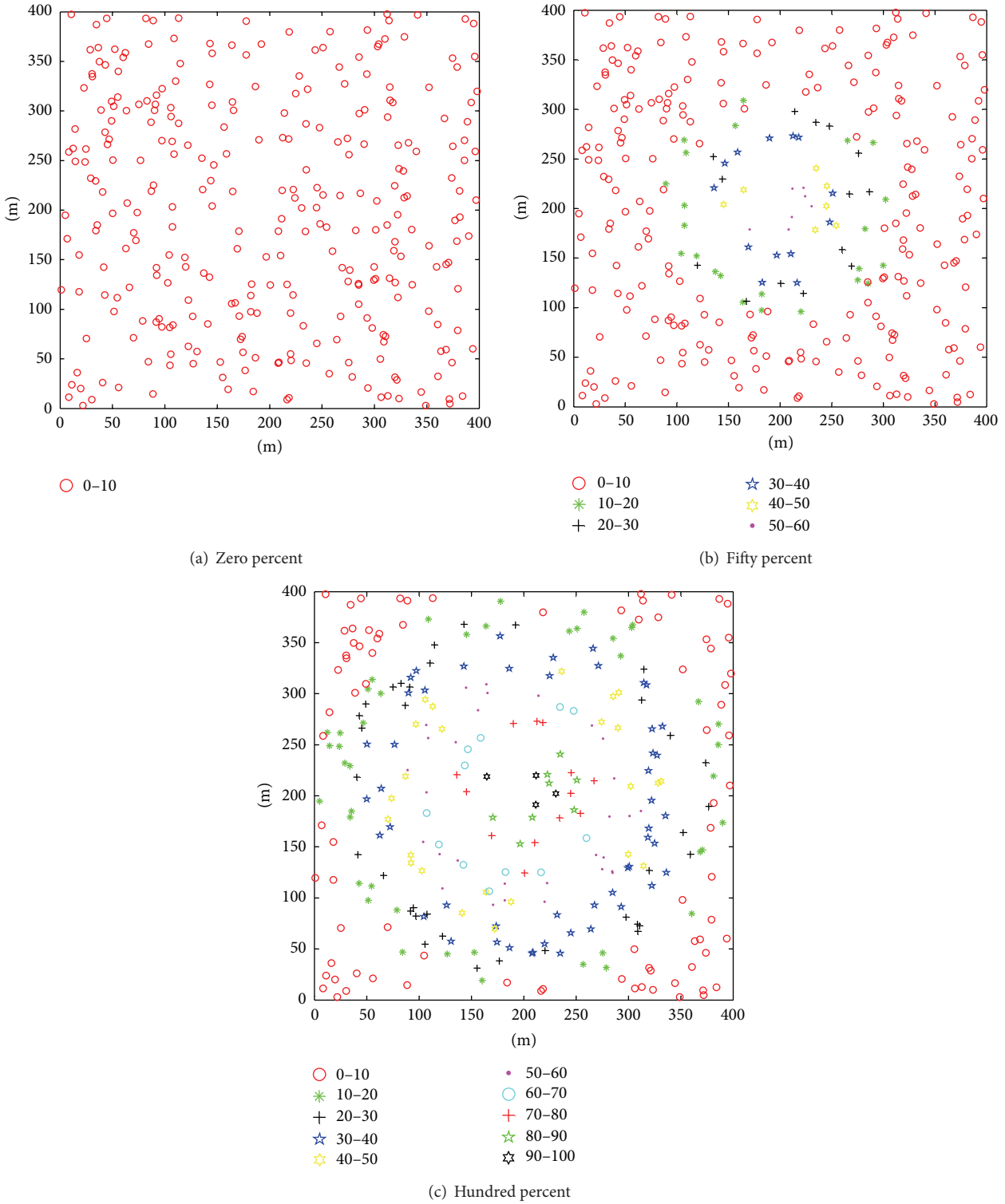
Hence, the final top-k result set FS is a part of the sending data set ($SS\text{-}case\ 1 \cup SS\text{-}case\ 2 \cup SS\text{-}case\ 3 \cup SS\text{-}case\ 4$).

Based on the above analysis, the sink will obtain the correct top-k result set. □

5. Simulation Results

To analyze the performance of our algorithm, we conduct experiments using omnetpp-4.1 [22].

We use a synthetic data set. We randomly deployed 300 homogeneous sensor nodes in the $400 * 400\,m^2$ rectangular region and the sink is located at the center. The data is generated using Gaussian distribution in which mean is proportional to the distance between a sensor node and the sink; the standard deviation is 0.5. As shown in Figure 7(a), the data values of all nodes are less than 10 in round 1. Starting from round 2, the data values of nodes closed to the sink increase gradually. As shown in the Figure 7(b), about round 25, the data values of the nodes which are closer to the sink are in the range [50–60), and the node is farther away from the sink; the data value is smaller. As shown in Figure 7(c), about round 50, the data values of the nodes which are closer to the sink are in the range [90–100). After that the data



(a) Zero percent

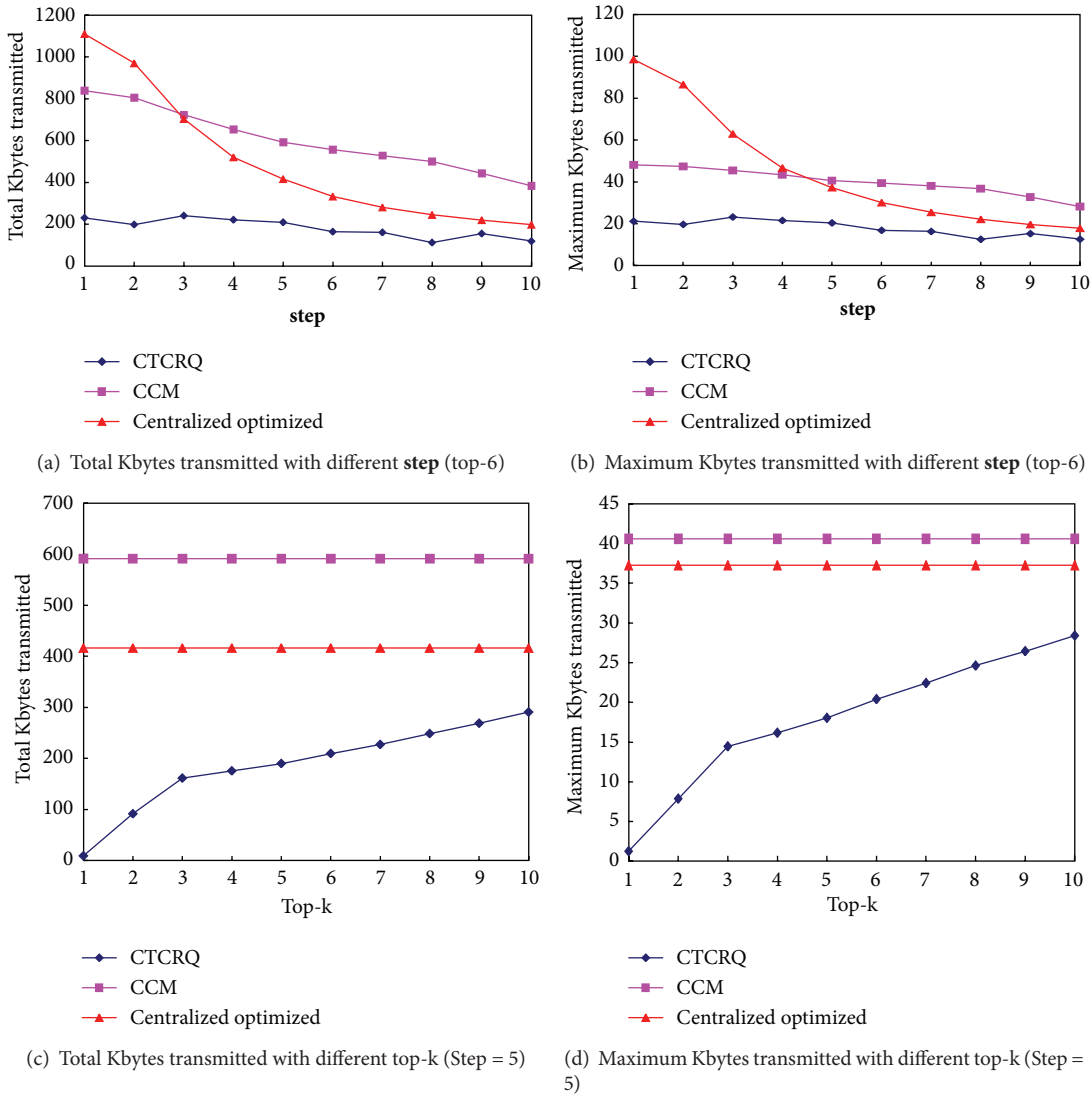
(b) Fifty percent

(c) Hundred percent

FIGURE 7: The data distribution of synthetic data set.

value of all nodes decreases gradually, it forms the subfigure (b) about round 75 and subfigure (a) about round 100. After that the data values of all nodes increase gradually, the cycle continues until it gets 1000 rounds data. We use this data set to simulate the process of expansion and shrinking gradually of the contour lines.

Our scheme is based on the TAG [23] routing algorithm and assumes communication links are error-free as well as



(a) Total Kbytes transmitted with different **step** (top-6)

(b) Maximum Kbytes transmitted with different **step** (top-6)

(c) Total Kbytes transmitted with different top-k (Step = 5)

(d) Maximum Kbytes transmitted with different top-k (Step = 5)

Figure 8: The performance comparison of synthetic data set.

MAC layer is ideal case. To compute the bytes transmitted, we define a sampling period as a round. Each of the following simulation result represents an average summary of 10 runs.

In our simulation, the size of node ID is 2 bytes. The attribute values are normalized and one normalized attribute value *suffix* occupies 1 byte. In CCM [13] algorithm, the parameter CN-array occupies 2 bytes.

The transmission range of the sink node is usually greater than the transmission range of ordinary sensor node, so we assume that the transmission range of the sink node can cover most regions of the monitoring networks.

5.1. Performance Analysis. As mentioned above, energy consumption is a critical issue for wireless sensor networks and radio transmission is the most dominate source of energy consumption. Thus, we measure the total and maximum amount of Kbytes (kilo bytes) data transmitted in wireless sensor networks as the performance metrics. Maximum amount of Kbytes data transmitted means the largest amount of data transmitted to the sink of a node among all nodes. We use this metric to evaluate the network lifetime, because one node which sends the largest amount of data to the sink means the node will consume the maximum energy and will die fastest. Therefore, compared with these two metrics, maximum amount of Kbytes data transmitted is more important than the total amount of Kbytes data transmitted.

As mentioned above, there are many continuous contour querying algorithms, such as eScan [9], isoline aggregation [10], Iso-Map [11, 12], CCM [13], the literature in [14], and improved Isoline aggregation [15]. In these algorithms, the literature in [12] is an expanded version of the Iso-Map [11]. In the literature in [14], many mobile nodes are deployed. While in our application scenario, the sensor nodes are stationary. So the algorithm proposed in the literature in [14] is not suitable as a comparison algorithm. The improved Isoline

aggregation [15] algorithm has no essential breakthrough compared with the isoline aggregation [10] algorithm, while the CCM [13] algorithm is a representative and energy efficient algorithm in these algorithms. Thus, we compared our CTCRQ algorithm with CCM algorithm.

For fair comparison, we modified the CCM algorithm a little. If CN-array and value range are not changed then node s_i does not broadcast a "report sent message" to its one-hop neighbors and does not send its ID back to the sink. If the sink does not receive a "report sent message" from node s_i, it denotes that the value of node s_i is not changed. In this way, the network can further reduce the amount of data transmitted than the original CCM algorithm.

To obtain the top-k contour region, the naive approach, that we called Centralized Algorithm, is that all nodes transmit their data to the sink at every round. The total data transmitted is 5031.5 Kbytes. The maximum data transmitted is 337.125 Kbytes. The total and maximum data transmitted of Centralized algorithm are obviously larger than the ones of other three algorithms (Centralized Optimized, CTCRQ, and CCM algorithms).

Centralized Optimized Algorithm uses temporal data correlation to reduce the amount of data transmitted. In this optimized version, nodes report their data (readings) directly to the sink only when their values have changed from one value sub-range to another.

We first investigate the impact of changing the value sub-range of **StrContour** on the network performance. Parameter **step** indicates the difference of sub-range, as shown in Definition 2. As shown in subfigures (a) and (b) of Figure 8, with the **step** increasing, the total and maximum Kbytes transmitted of all three algorithms decrease. The greater the **step** is, the greater the scope of the value sub-range is, as well as the smaller the possibility of the data need to be sent is. With the **step** increases, the total and maximum Kbytes transmitted of CTCRQ algorithm decrease too, and the trend is gentle. Because they only calculate the bytes transmitted of top-6 region nodes, and the number of top-6 region nodes is less than the nodes of total network.

When the **step** size is relatively smaller, the number of nodes which have changed their values from one sub-range to another sub-range is more, and the number of nodes that need to be reported is more too. In CCM algorithm, using the CN-array technology, a reporting node can suppress all contour nodes around it. Hence, when the **step** size is relatively smaller, the CCM algorithm is superior to the Centralized Optimized algorithm.

With the **step** size increasing, the number of nodes which have changed their values from one sub-range to another sub-range decreases and the number of nodes that need to be reported decreases too. In CCM algorithm, if a node changed its value from one sub-range to another sub-range, the node will broadcast its node ID and value information to its neighbor nodes. Its neighbor nodes may not change their values. Then the node will send "report sent message" which only includes the information of itself to the sink. While in the Centralized Optimized algorithm, nodes send their data (readings) to the sink only when their values have changed from one sub-range to another. Hence, when the **step** size is relatively larger, CCM algorithm is not as good as the Centralized Optimized algorithm.

The total and maximum Kbytes transmitted of CTCRQ algorithm are less than the ones of Centralized Optimized and CCM algorithms.

As shown in subfigures (c) and (d) of Figure 8, with the number of top-k increasing, the total and maximum Kbytes transmitted of CTCRQ algorithm increase too. Both in calculating the total number of bytes transmitted and the maximum amount of bytes transmitted, CTCRQ algorithm is better than Centralized Optimized and CCM algorithms.

Experimental result shows that CTCRQ algorithm outperforms the existing ones in term of data transmission.

6. Conclusions and Future Work

In this paper, we proposed a Continuous Top-k Contour Regions Querying (CTCRQ) algorithm in wireless sensor networks. Our experimental result shows that the proposed CTCRQ scheme can reduce the amount of bytes transmitted as well as extend the network lifetime.

In the future, we plan to extend the proposed scheme to other aggregate functions such as join, average, and sum.

Acknowledgments

This research was supported by the National Basic Research Program of China (973 program) (2012CB316205) and the National Natural Science Foundation of China (61070056, 61033010).

References

[1] Institute of Electrical Electronics Engineers, "Ten emerging technologies that will change your world," *IEEE Engineering Management Review*, vol. 32, pp. 20–30, 2004.

[2] H. Zhang, Z. Wu, D. Li, and H. Chen, "A sampling-based algorithm for approximating maximum average value region in wireless sensor network," in *Proceedings of the 39th International Conference on Parallel Processing Workshops (ICPPW '10)*, pp. 17–23, San Diego, CA, USA, September 2010.

[3] Y. Cho, J. Son, and Y. D. Chung, "POT: an efficient top-k monitoring method for spatially correlated sensor readings," in *Proceedings of the 5th International Workshop on Data Management for Sensor Networks (DMSN '08)*, pp. 8–13, August 2008.

[4] M. Wu, J. Xu, X. Tang, and W. C. Lee, "Top-k monitoring in wireless sensor networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 7, pp. 962–976, 2007.

[5] M. H. Yeo, D. O. Seong, and J. S. Yoo, "PRIM: priority-based top-k monitoring in wireless sensor networks," in *Proceedings of the International Symposium on Computer Science and its Applications (CSA '08)*, pp. 326–331, Hobart, Australia, October 2008.

[6] M. Yeo, D. Seong, and J. Yoo, "Data-aware top-k monitoring in wireless sensor networks," in *Proceedings of the IEEE Radio and Wireless Symposium (RWS '09)*, pp. 103–106, San Diego, CA, USA, January 2009.

[7] X. Liu, J. Xu, and W. C. Lee, "A cross pruning framework for Top-k data collection in wireless sensor networks," in

Proceedings of the 11th IEEE International Conference on Mobile Data Management (MDM '10), pp. 157–166, Kansas City, MO, USA, May 2010.

[8] S. Mo, H. Chen, and Y. Li, "Clustering-based routing for top-k querying in wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, article 73, 2011.

[9] Y. J. Zhao, R. Govindan, and D. Estrin, "Residual energy scans for monitoring wireless sensor networks," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC '02)*, pp. 17–21, 2002.

[10] I. Solls and K. Obraczka, "Efficient continuous mapping in sensor networks using isolines," in *Proceedings of the 2nd Annual International Conference on Mobile and Ubiquitous Systems-Networking and Services (MobiQuitous '05)*, pp. 325–332, July 2005.

[11] Y. Liu and M. Li, "Iso-map: energy-efficient contour mapping in wireless sensor networks," in *Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS '07)*, Toronto, Canada, June 2007.

[12] M. Li and Y. Liu, "Iso-map: energy-efficient contour mapping in wireless sensor networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 5, pp. 699–710, 2010.

[13] C. Zhong and M. Worboys, "Continuous contour mapping in sensor networks," in *Proceedings of the 5th IEEE Consumer Communications and Networking Conference (CCNC '08)*, pp. 152–156, Las Vegas, NV, USA, January 2008.

[14] Q. Chen, "Automatic contour mapping system in sensor network," *Applied Mechanics and Materials*, vol. 182-183, pp. 1164–1168, 2012.

[15] R. Guocan and D. Guowei, "An improved isoline based data aggregation scheme in wireless sensor networks," *Procedia Engineering*, vol. 23, pp. 326–332, 2011.

[16] J.-J. Kim, I.-S. Shin, Y.-S. Zhang, D.-O. Kim, and K.-J. Han, "Aggregate queries in wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2012, Article ID 625798, 15 pages, 2012.

[17] R. Zhu, "Efficient fault-tolerant event query algorithm in distributed wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2010, Article ID 593849, 7 pages, 2010.

[18] R. Zhu, "Intelligent collaborative event query algorithm in wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2012, Article ID 728521, 11 pages, 2012.

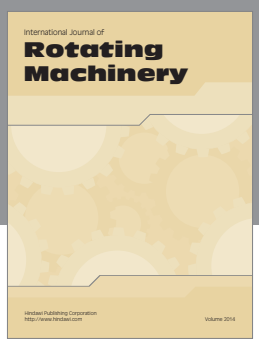
[19] I. F. Akyildiz, M. C. Vuran, and Ä. B. Akan, "On exploiting spatial and temporal correlation in wireless sensor networks," in *Proceedings of the Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt '04)*, pp. 71–80, 2004.

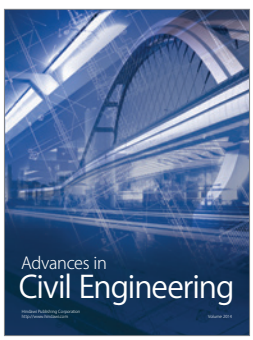
[20] N. D. Pham, T. D. Le, and H. Choo, "Enhance exploring temporal correlation for data collection in WSNs," in *Proceedings of the IEEE International Conference on Research, Innovation and Vision for the Future in Computing and Communication Technologies*, pp. 204–208, Ho Chi Minh City, Vietnam, July 2008.

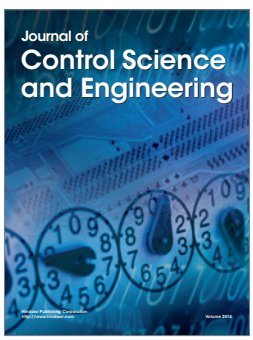
[21] M. Stern, E. Buchmann, and K. Böhm, "Towards efficient processing of general-purpose joins in sensor networks," in *Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE '09)*, pp. 126–137, Shanghai, China, April 2009.

[22] http://www.omnetpp.org/.

[23] S. Madden, M. J. Franklin, J. Hellerstein, and W. Hong, "TAG: a tiny aggregation service for ad-hoc sensor networks," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, pp. 131–146, December 2002.







Hindawi
Submit your manuscripts at
http://www.hindawi.com