

Isovector: a New Data Aggregation Method in Sensor Networks

Cheng Zhong

National Center for Geographic
Information and Analysis
University of Maine, Orono, ME, 04469, USA
Email: cheng.zhong@umit.maine.edu

Michael Worboys

National Center for Geographic
Information and Analysis
University of Maine, Orono, ME, 04469, USA
Email: worboys@spatial.maine.edu

Abstract—This paper presents some efficient techniques for generating a contour map for a field monitored by a wireless sensor network. Current approaches cannot efficiently compute the answer to queries such as “Which areas of the field have values between 50 and 60?” A contour map is a useful data representation schema that provides an efficient way to visualize an approximation to the monitored field. In this paper, we discuss an energy-efficient technique, Isovector aggregation, for generating such contours using an in-network approach. Our technique achieves energy efficiency in two principal ways. Firstly, only a selection of nodes near contours are chosen to report, and each report message contains information about a part or all of the contours, rather than any single nodes ID and value pair. Secondly, contours are progressively merged and simplified along the data routing tree, which eliminates many unnecessary contour points from contour vectors before they are transmitted back to the base station. By Isovector aggregation, the base station receives complete representations of the contours, which require no further processing. Analysis shows that, for region related monitoring, Isovector aggregation is the only technique that has $O(\sqrt{n})$ traffic generation and that considers in-network traffic reduction at the same time. These two factors make Isovector aggregation highly scalable. Experimental results using simulations also show that Isovector aggregation sends considerably less data compared to other approaches, such as the no aggregation approach and the Isolines aggregation technique, without compromising the accuracy of representations of the baseline maps.

I. INTRODUCTION

The rapid development of wireless sensor technology provides the potential to track and monitor dynamic phenomena in the world. A simple approach to this is to monitor regions of high activity using the value threshold approach [1], [2], [3]. When a node’s value is higher than the threshold, we deem this node to be in a high activity region, and this is reported to the base station. Though the threshold-based approach is simple to implement, sometimes it provides insufficient information about the field. We may be interested in region related questions like “What does the whole field look like?” and “Which region has values between 50 and 60?”. Contours provide an efficient way to visualize the field monitored by wireless sensor network (WSN) [4], [5], [2]. In this paper, we propose the Isovector aggregation, a new aggregation method for continuous region monitoring that can be extended to region and range related queries.

In WSNs, one of the most significant issues is the energy problem. Node-to-node data transmission consumes most of the energy and dwarfs all other energy consumption factors [6], [7]. In-network aggregation has been employed successfully as one of the best ways to save network energy [4], [2]. The idea of in-network data aggregation is to process data and remove redundant data according to the application requirement during the data flows from sensor nodes to the base station. Most previous work does not consider how to generate contours in the network, but rather the base station has to process all the data received to generate the contour map.

In this paper, we describe a novel technique, Isovector aggregation, that targets energy efficient data collection from sensors to generate contours in the network. Energy efficiency is achieved by two approaches.

- Instead of having all sensor nodes send their values to the base station, only a few nodes near the contours need to report. Reporting data contains contours rather than discrete values.
- Instead of having all data transmitted directly, we remove redundant data, generate contours in the network and transmit them to the base station.

Our major contributions include the following:

- A ring data structure that is used to generate local contour vectors between a node and its neighbors.
- Isovector aggregation technique for in-network contour merging and simplification. Each reporting message contains information about a part of or all the contours rather than any single node’s ID and value pair. The total report data size is greatly reduced, which leads to significant energy savings and the base station does not have to do any further interpolation to generate a contour map.
- Analysis which shows that, for region related monitoring, Isovector aggregation is the only technique that incurs $O(\sqrt{n})$ traffic generation and that considers in-network traffic reduction at the same time.
- Comparison of Isovector aggregation with existing methods, including no aggregation and Isolines aggregation [4]. The results show that Isovector aggregation greatly out-performs these methods.

The rest of this paper is organized as follows. We briefly describe related work in section II, and give an overview of our aggregation technique in section III. The preliminaries of our work are given in section IV. We then present our in-network Isovector aggregation in section V. Theoretical analysis is given in section VI. After evaluating performance in section VII, we conclude the paper in section VIII.

II. RELATED WORK

Localized contour/boundary detection is a hot research area in WSNs. In addition, in-network aggregation is extensively researched for data reporting and many aggregation approaches are proposed for different application scenarios.

A. Contour detection in the network

Chintalapudi and Govindan [8] discuss three boundary detection methods and return enough data so that the base station can construct an accurate boundary. They do not consider how to transmit boundary information back. If all boundary nodes report, a mass of network energy will be consumed. Ding et al. [9] proposed localized fault-tolerant boundary and fault sensor detection using spatial data mining techniques. Like [8], [9] do not mention how to report these output nodes to the base station and did not consider continuous boundary changes. If all boundary nodes have to report, a large amount of data will be transmitted. In this paper, contour detection is based on neighboring node value differences: a contour exists between two nodes if they are in different value ranges.

B. Generating contours at the base station

Most previous papers for contour reporting do not consider how to generate contours using in-network approaches. These methods have to relay all or some node information back to the base station, and later the base station uses such information to interpolate values to other nodes and then generate contours.

Event Contour is discussed by Meng et al. in [5] where spatial suppression and temporal suppression are combined together. After the base station receives all reports, values will be interpolated to other non-reporting nodes. Event Contour chooses reporting nodes from all nodes. Therefore, it can never suppress all values in an area. Even when all nodes have the same values, some must always report to the base station.

Solis and Obraczka [4] propose Isolines aggregation. Energy efficiency is achieved by having each node only report to the base station if it detects an isoline between itself and its neighbors; otherwise, no report is generated. The drawback of Isolines is that, in a sparse WSN, nearly half of nodes along both sides of each isoline will report in each sampling round. This may cause a great communication overhead. If sensor nodes are deployed in a dense manner, nodes will only report isoline that they detected between themselves and their children or parents according to the data collection tree. If some parts of an isoline are between two sub-collection trees, then the isoline cannot be detected precisely.

Silberstein et al. [11] propose a method called COUCH that attempts to make full use of spatial suppression and

temporal suppression. COUCH collects history and statistics for minimal cost spanning forest construction and continuous optimization. Ideally, only one node on each side of the contour needs to report a change. COUCH shows good performance if the contours move regularly or the contours seldom change. If the contours change irregularly, historical information may become less useful for the min-cost forest construction and optimization. In the worse case (a vertical contour changes to a horizontal contour), most contour nodes have to report in the next round. What is more, the spanning forest has often to be updated which will consume large amounts of energy. Again, COUCH has to generate other nodes' values at the base station.

In a recent paper [10], Yunhao and Mo propose the Iso-map method, which shares similar ideas with Isolines aggregation. They prove that only letting contour nodes report will decrease network traffic generation from $O(n)$ to $O(\sqrt{n})$. But they do not consider how to reduce data transmission in the network, and they still have to generate contours at the base station.

C. Generating contours in the network

Few papers consider how to generate contours using in-network approaches. They all belong to polygon aggregation.

Hellerstein et al. [2] propose a method called Isobar that uses spatially correlated data aggregation for mapping purposes. Isobar aggregates nodes with the same values into polygons, as the data flows towards the base station. In the Isobar approach, each node has to participate in the aggregation by sending not only its ID and value but also location information, which makes Isobar perform worse than Isolines. Zhao [12] proposed EScan which uses the same approach and has the same problems as Isobar. The difference is that this method constructs contour maps to monitor sensor residual energy rather than physical events in the network.

Xu et al. [13] mention in-network contour generation for contour map matching which actually is an extension of polygon aggregation. Location information from each node still has to be transmitted before the aggregation, which causes huge data transmission. Their method can generate regular polygons (rectangle etc.) in the network. Unlike this approach, Isovector only focuses on the borderlines of polygons and it is not confined to regular polygons.

III. OVERVIEW

In this section, after giving a short description of contour, we use a simple example to explain the basic idea of our Isovector aggregation.

A. Contour Overview [5], [14]

Contour lines are curves connecting points of same particular values. A contour map uses contour lines (often just called a "contour") to join points of equal values. Contours are often given specific names beginning "iso-" according to the nature of the variable being mapped. When we map temperatures, we often call it isotherm. Contour maps have a step-value which separates two adjacent lines. For example, given a step

value of 10, an isotherm contour map indicate regions that differ by 10 units in temperature. The region between two adjacent lines are in the same value range. Contour maps present a simple way of fine tuning the tradeoff between information and the cost of obtaining it by adjusting the step-values to suit situational requirements. There can be many contour maps, each corresponding to a different parameter for e.g., temperature, pressure, wind speed, with different step-values for each parameter, thereby giving a clearer picture of the monitored field. In this paper, we use temperature mapping as an example.

B. Aggregation through contours

Once the value ranges are defined, we use neighboring node value differences to detect contours at each sampling round. If two adjacent nodes are in different value ranges, there is at least one contour between them and we want the two nodes to detect the contour through communication. These nodes are called *contour nodes*. Our goal is to achieve energy efficient contour mapping in WSNs and we aim to minimize the number and size of messages sent in the network. In order to introduce our aggregation technique, we begin with a simple straight contour example. More details will be discussed in the following sections.

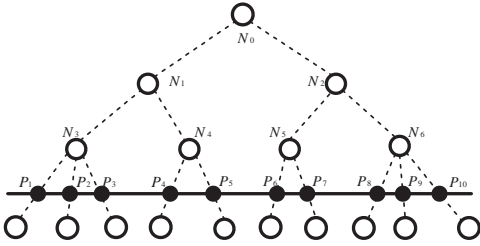


Fig. 1. A straight line contour in a sub-routing tree

Consider a sub-routing tree of the network. Suppose a subset of nodes detect a horizontal contour (figure 1) between them. A few reporting nodes N_3 , N_4 , N_5 and N_6 are chosen to report. Isovector aggregation follows steps shown in figure 2: N_3 detects the contour and generate three contour points P_1 , P_2 and P_3 which are the mid-points between N_3 and the corresponding neighboring nodes. Based on the three contour points, node N_3 then produces a contour vector (P_1, P_3) and reports it to node N_1 following the data routing tree. P_2 is not contained in the vector because it is removed by N_3 as a redundant point. Nodes N_4 , N_5 and N_6 work in the same way as N_3 . After N_1 receives the contour vector (P_1, P_3) from N_3 and (P_4, P_5) from N_4 , it merges the two vectors to a new vector (P_1, P_5) . In this merging process, contour points P_3 and P_4 are removed as redundant points by node N_1 . N_1 then reports the vector (P_1, P_5) to N_0 and N_2 reports the vector (P_6, P_{10}) to node N_0 . Such process will be repeated along the data routing tree. Finally, node N_0 will merge the vector (P_1, P_5) and (P_6, P_{10}) and generate the vector (P_1, P_{10}) . This vector is sufficient to represent the

straight line contour. N_0 then only reports (P_1, P_{10}) to the base station. In this aggregation process, because many redundant data are removed, the total data transmitted is significantly reduced, which decreases network energy consumption.

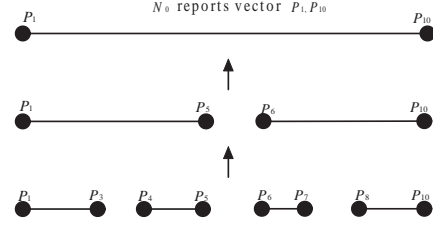


Fig. 2. Isovector aggregation example

IV. PRELIMINARIES

In this section, we first give a relevant definition that will be used later. Then we presents types of messages and time model used by Isovector aggregation. The assumptions we made in this paper are that each node has a unique ID and knows its own location through either a GPS or some GPS-less techniques [15], [16]. Then each node knows its neighboring nodes locations and IDs by simple node-to-neighbor communication. We set the default contour scale to 10 if not illustrated specifically and we also call contours contour vectors because each generated contour is composed of a series of points and it has a start point and an end point. If the start point and the end point of a contour is the same, this contour is a ring.

A. Contour neighborhood ring

We denote the value at node u by $R(u)$ and its value range by $Range(u)$. For any two nodes u and v , if u and v are in the same predefined value range, we have $Range(u) = Range(v)$, otherwise, $Range(u) \neq Range(v)$ (either $Range(u) > Range(v)$ or $Range(u) < Range(v)$). For example: if $R(u) = 42$ and $R(v) = 45$, u and v are both in the value range 40-49 and $Range(u) = Range(v)$.

Definition 1 (Contour neighborhood ring): Let u be a node and v_1, v_2, \dots, v_n be the one-hop neighbors of u , sequenced in counterclockwise cyclic order around u , where a start node v_1 is randomly assigned in advance. The contour neighborhood ring associated with u is a ring data structure $[CN_1, CN_2, \dots, CN_n]$ starting from v_1 where

$$CN_i = \begin{cases} R(v_i), & \text{if } Range(u) \neq Range(v_i) \\ null, & \text{if } Range(u) = Range(v_i) \end{cases}$$

for $1 \leq i \leq n$.

Figure 3 shows an example of a node representation with its corresponding contour neighborhood ring.

Type	Description
<i>Negotiation</i>	Node broadcasting ID, value and location
<i>Query</i>	Base station initiating fetching contours
<i>Notification</i>	Node broadcasting ID and value
<i>Vector</i>	Response message to query
<i>Updating</i>	Response message without contours
<i>Acknowledgement</i>	Response message to children

TABLE I
MESSAGE TYPE

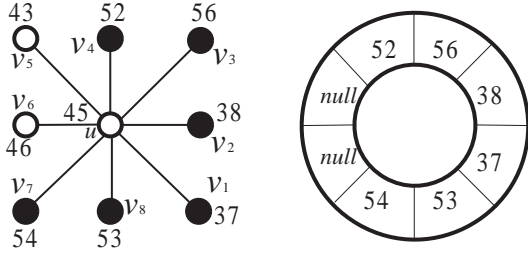


Fig. 3. A node representation of u and the contour neighborhood ring (numeric numbers are values of nodes)

B. Message types

We define six types of messages (table I). *Query* message is broadcasted by the root at the network setup phase. *Negotiation* message tells neighbors ID, initial value and location of the owner nodes, and is only broadcast after receiving *Query* message. *Notification* message will be sent when data changes cause a contour to appear or disappear. The reporting message *Vector* contains the reporting node's ID, contour vectors and contour values. *Updating* is another kind of reporting message that only contains the sending node's ID. If a node receives an *Updating* message from one of its children, it will use the contours previously reported by that child as the current ones. In this way, nodes do not have to send huge contour vector data back at each round if some contours do not change. *Acknowledgement* is used to deal with packet loss. A node can send this message back after receiving a reporting message from a child.

C. Aggregation time model

The time model we use in this paper is based on cascading timer [17]. In cascading timer, instead of having nodes randomly schedule their timeout, i.e., the time interval nodes wait to receive data from their children before forwarding the next data aggregate, a node's timeout is set based on the node's position in the data distribution tree. Thus, a node's timeout will happen right before its parent's. Using cascading timer, a node merges contour vectors after it receives all *Vector* messages from its children. In order to handling packet loss (section V-F), we modify it slightly and set a relatively longer time-out for those nodes near the base station.

V. ISOVECTOR AGGREGATION

In this section, we mainly present our in-network Isovector aggregation which mainly includes four parts: local contour

generation, reporting node selection, contour simplification and contour merging. Temporal suppression for reducing report size is also discussed. We assume the query for continuous mapping is processed repeatedly over a series of rounds, where each node generates a value in each round.

A. Local contour generation

The reporting node readings and the *null* values in the neighborhood ring will work as separators that divide the ring into several partitions. Then a local contour vector will be generated for each partition by counterclockwise order. Each element in a contour vector is the mid-point of the reporting node and the corresponding contour neighbor. Let the *scale* denote the contour scale we defined. If a node u is chosen to report and it is in a higher value range than corresponding neighbors, the value of the reporting contour vector V between this node and those neighbors is set as: $Value(V) = (R(u)/scale) * scale$. Otherwise, if u is chosen to report and it is in a lower value range, the value of the reporting contour vector V between this node and corresponding neighbors is set as: $Value(V) = (R(u)/scale + 1) * scale$.

Figure 4 shows an example of a neighborhood ring of a node and the corresponding local contour vectors. Three partitions $\{37, 38\}$, $\{56, 52\}$ and $\{53, 54\}$ exist in the neighborhood ring and local contour vectors (P_1, P_2) , (P_3, P_4) and (P_7, P_8) will be generated for the three partitions. Each generated vector has a headID (hID) and a tailID (tID) which are equal to the reporting node ID.

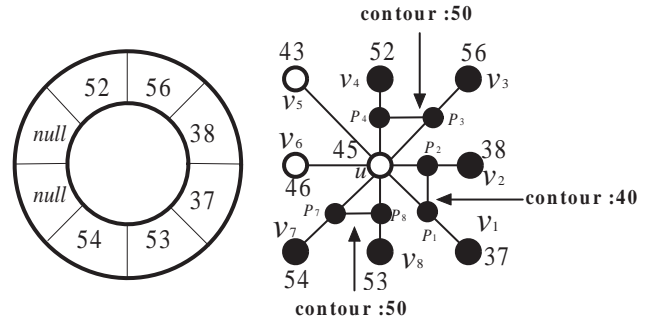


Fig. 4. u 's neighborhood ring and the corresponding local contour vectors

Besides differentiating contours of different values, the partition can be used to recognize narrow contours. In practice, most reporting nodes only have one partition each, and hence each of them only generates one local contour vector. In some rare cases, reporting nodes near a narrow contour will generate two or more partitions. A good example of a narrow contour is shown in figure 5. Our aim is that the final contour is similar to the one shown in figure 6. Suppose node u generates V_1 (P_4, P_6, P_3, P_1) (figure 7) and reports it though node v , and the right part of the broken line also generates a vector V_2 which starts at P_i and ends with P_j . Suppose also that V_2 is reported though node w . When V_1 and V_2 meet at a common ancestor of v and w , it is not easy for this ancestor to merge

V_1 and V_2 together to form a new vector similar to the one shown in figure 6. Suppose u generates two vectors V_0 and V_1 and does not merge them. Then, after the common ancestor receives three vectors, it is straightforward to connect P_3 to P_i , and P_6 to P_j (figure 8), because they are near to each other. Then the contour we get will be similar to the one shown in figure 6. This process is a part of the merging algorithm that will be illustrated in detail in section V-D.

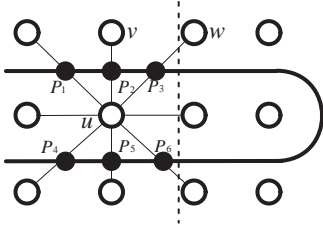


Fig. 5. Narrow contour (1)

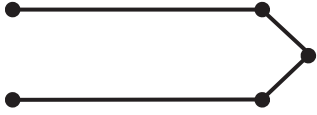


Fig. 6. Narrow contour (2)

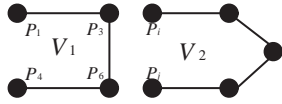


Fig. 7. Narrow contour (3)

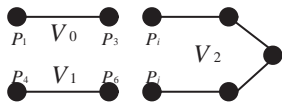


Fig. 8. Narrow contour (4)

B. Reporting node selection

Local contour detection in each round is based on neighborhood information gathered through node-to-neighbor communication. If the ranges of sensed values are changed, nodes broadcast *Notifications* to neighbors in the beginning of each sampling round. Each node that receives *Notifications* compares its own value with neighbors' values. Some neighbors' values might be in the same value range as the reporting node. As specified before, such corresponding entries will be set to *null* and they, together with the *scale* value, act as separators to partition the contour ring. If some neighbor values are on different sides of a contour, then at least one contour exists. When the reporting timer expires, this node

will check if it should report. If it does, a *Vector* message will be constructed and transmitted to the parent.

Contour detection is symmetric. Not all contour nodes are required to generate contour vectors and report. We choose contour nodes in the higher value ranges than neighbors to report. Each vector has a headID and a tailID which is equal to the ID of the node(s) that report(s) them. Therefore, by maintaining such information of a contour, the base station can know which side of the contour is in higher value ranges and which side of the contour is in lower value ranges. In some cases, two adjacent nodes may not exist in consecutive value ranges. In this situation, both node in the lower value range and the higher value range will report. Figure 9 gives an illustration.

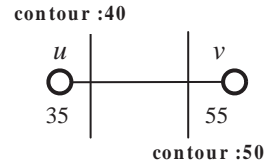


Fig. 9. u will report a contour with value 40 between u and v and v will report a contour with value 50 between u and v .

C. In-network contour simplification

Although a contour vector only contains important contour point information, such information may be still quite large. Without losing contour fidelity greatly, we consider how to weed out redundant points from a contour vector (figure 10). This problem is also called poly-line simplification which has been researched over the years. In this paper, we use the Douglas-Peucker line simplification algorithm [18] for vector simplification in that it was best at choosing critical points when compared with others [19]. We should point out that, although the Douglas-Peucker approach is chosen, any other poly-line simplification method can be adopted if it can retain the shape of the original line.

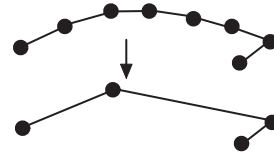


Fig. 10. Simplify a contour vector

D. In-network contour merging

After a node receives *Vector* messages from its children, it saves them in a local contour Vector-Array ($vArr$). If this node is also a reporting node, local generated vectors will also be added to the array. When the reporting time comes, the node tries to merge saved contour vectors in the $vArr$ and removes redundant points from all merged contour vectors.

Let $V.h$ (short name of vector head) denote the first point in the vector and $V.t$ (short name of vector tail) denote the last point in a vector V . We define the distance $D(V_1, V_2)$ between two vectors V_1 and V_2 as the minimal value of $D(V_1.h, V_2.h)$, $D(V_1.h, V_2.t)$, $D(V_1.t, V_2.h)$ and $D(V_1.t, V_2.t)$. If the distance $D(V_1, V_2)$ is shorter than a predefined threshold, they should be connected together. The node continuously merges each pair of adjacent contours until no two contour vectors are near enough to each other. In the merging process, any two vectors sent by the same child node will not be merged no matter how near the distance is between them. This restriction is used to avoid generating wrong contour shapes of narrow contour cases.

By calculating the distance between two vectors, algorithm 5 checks merging conditions. If the distance is less than a predefined tolerance and the merging parts are not reported by the same node, it returns a positive value which will be used by algorithm 2. Algorithm 2 merges two specific contour vectors. For example, when $mgValue$ is equal to 1, V_1 head and V_2 head will be connected together. The new generated V_3 then replaces V_1 in the $vArr$ structure. By calling algorithm 5 and 2, algorithm 3 continuously checks merging status of two vectors and merges them if they meet the merging standard.

Algorithm 1 $ckMerge(V_1, V_2, tol)$

```

1: if ( $V_1$  or  $V_2$  is a ring) then
2:   return 0;
3: if ( $V_1.value \neq V_2.value$ ) then
4:   return 0;
5: calculate  $D(V_1, V_2)$ ;
6: if ( $D(V_1.h, V_2.h) \equiv D(V_1, V_2)$  and  $D(V_1, V_2) < tol$  and
    $V_2.hID \neq V_2.hID$ ) then
7:   return 1;
8: if ( $D(V_1.h, V_2.t) \equiv D(V_1, V_2)$  and  $D(V_1, V_2) < tol$  and
    $V_2.hID \neq V_2.tID$ ) then
9:   return 2;
10: if ( $D(V_1.t, V_2.h) \equiv D(V_1, V_2)$  and  $D(V_1, V_2) < tol$  and
    $V_2.tID \neq V_2.hID$ ) then
11:  return 3;
12: if ( $D(V_1.t, V_2.t) \equiv D(V_1, V_2)$  and  $D(V_1, V_2) < tol$  and
    $V_2.tID \neq V_2.tID$ ) then
13:  return 4;
14: return 0;

```

All new generated vectors are also stored in the local $vArr$ (last line of algorithm 2) structure. After the local time-out, Douglas-Peucker algorithm will be called for vector simplification. Those simplified vectors will be added to the *Vector* message and sent to the parent. Each node along the data routing tree does the same process until the base station receives final results.

E. Updating contours with temporal suppression

In many applications, contours do not change frequently because events are rare, and most node values are unchanged or only slightly changed over time. In such situations, it is not

Algorithm 2 $mgVector(V_1, V_2, mValue)$

```

1: if ( $mValue \equiv 1$ ) then
2:    $V_3 \leftarrow Reverse(V_1) + V_2$ ;
3:    $V_3.hID \leftarrow V_1.tID$ ;
4:    $V_3.tID \leftarrow V_2.tID$ ;
5: if ( $mValue \equiv 2$ ) then
6:    $V_3 \leftarrow Reverse(V_1) + Reverse(V_2)$ ;
7:    $V_3.hID \leftarrow V_1.tID$ ;
8:    $V_3.tID \leftarrow V_2.hID$ ;
9: if ( $mValue \equiv 3$ ) then
10:   $V_3 \leftarrow V_1 + V_2$ ;
11:   $V_3.hID \leftarrow V_1.hID$ ;
12:   $V_3.tID \leftarrow V_2.tID$ ;
13: if ( $mValue \equiv 4$ ) then
14:   $V_3 \leftarrow V_1 + Reverse(V_2)$ ;
15:   $V_3.hID \leftarrow V_1.hID$ ;
16:   $V_3.tID \leftarrow V_2.hID$ ;
17:  $V_3.value \leftarrow V_1.value$ ;
18:  $V_1 \leftarrow V_3$ ;

```

Algorithm 3 $mergeVectors(vArr[], tol)$

```

1: for ( $i \leftarrow 0$  to  $vArr.len - 3$ ) do
2:   for ( $j \leftarrow i + 1$  to  $vArr.len - 3$ ) do
3:      $mValue \leftarrow ckMerge(vArr[i], vArr[j], tol)$ ;
4:     if ( $mValue \neq 0$ ) then
5:        $mgVector(vArr[i], vArr[j], mValue)$ ;
6:       remove  $vArr[j]$  from  $vArr$ ;
7:   if ( $vArr[i]$  changed) then
8:      $i \leftarrow i - 1$ 

```

necessary to let nodes always transmit *Vectors* to the base station allowing the data flow tree. In each sampling round of Isovector aggregation, each node saves a copy of the detected and received contours before merging. In the next sampling round, if a node finds such information is not changed, it only sends a *Updating* to its parent, the parent will use the node's previous contours as the current ones. If a node only receives *Updating* from children in a specific round, it transmits a *Updating* to its parent in the same way.

Combining local contour generation, reporting node selection, contour simplification, contour merging and temporal suppression together, our algorithm for Isovector aggregation is shown in algorithm 4.

F. Dealing with packet loss

Vector messages sent by nodes near the base station may contain a great deal of contour information and their size is relatively large. If such a message is dropped by the network, we may not get a good contour map at the base station. For the nodes near the base station, we introduce simple *Acknowledgement* message which will be sent to children by a parent node after receiving a *Vector*. Then such children can report again if they do not receive *Acknowledgement* messages from parents after a short timeout.

Algorithm 4 Isovector aggregation

```
1: while 1 do
2:   gets sensed value;
3:   if (value range changes) then
4:     broadcasts Notification;
5:   while (monitor events) do
6:     if (receives Notification) then
7:       updates information;
8:     if (receives children's Vector) then
9:       saves Vector;
10:      needReport = true;
11:    if (receives children's Updating) then
12:      marks the corresponding existing contour;
13:      needReport = true;
14:    if (reporting timeout) then
15:      if (in a higher value range) then
16:        generates and saves local contours;
17:        needReport = true;
18:      else if (in non-consecutive value ranges) then
19:        generates and saves local contours;
20:        needReport = true;
21:      if (needReport) then
22:        if (all received msgs are Updating and no new
23:          local contours) then
24:          composes a Updating and reports;
25:        else
26:          merges vectors;
27:          simplifies vectors (Douglas-Peucker);
28:          composes a Vector and reports;
```

Algorithm 5 BSckMerge (V_1, V_2, tol)

```
1: if ( $V_1$  or  $V_2$  is a ring) then
2:   return 0;
3: if ( $V_1.value \neq V_2.value$ ) then
4:   return 0;
5: calculate  $D(V_1, V_2)$ ;
6: if ( $D(V_1.h, V_2.h) \equiv D(V_1, V_2)$  and  $D(V_1, V_2) < tol$ )
7:   then
8:     return 1;
9: if ( $D(V_1.h, V_2.t) \equiv D(V_1, V_2)$  and  $D(V_1, V_2) < tol$ )
10:  then
11:    return 2;
12: if ( $D(V_1.t, V_2.h) \equiv D(V_1, V_2)$  and  $D(V_1, V_2) < tol$ )
13:  then
14:    return 3;
15: if ( $D(V_1.t, V_2.t) \equiv D(V_1, V_2)$  and  $D(V_1, V_2) < tol$ )
16:  then
17:    return 4;
18: return 0;
```

G. The base station

At the end of each sampling round, the base station will receive some contour vectors. It does similar merging operations to the inner nodes, except that contour vectors sent by the same node can be merged together. In this way, we generate

an integrated contour map.

VI. ANALYSIS

Isovector aggregation only chooses a few contour nodes to report and progressively removes unwanted data along the data routing tree. In this section, we analyse the traffic generation and traffic reduction during transmission. Comparison with other techniques will be given in table II which shows the high scalability of Isovector aggregation.

A. Traffic generation

As we can see from section II, many previous techniques require all nodes to generate reports before aggregation. Let the network have n nodes. The complexity of traffic generation for those techniques is $O(n)$. From Iso-map [10], the total contour length in the network is $O(\sqrt{n})$ in a WSN with infinite density ($n \rightarrow \infty$ and $d \rightarrow 0$, where d is the communication range of a node). Because we use mid-points to represent contours and only choose contour nodes to report, the complexity of traffic generation of Isovector aggregation is $O(\sqrt{n})$. This result may indicate that, from the perspective of traffic generation, Isovector aggregation is more scalable than many other techniques. Isolines aggregation and Iso-map have the same property.

B. In-network traffic reduction

In real applications, how much communication overhead can be reduced by Isovector aggregation depends on contour shapes, contour locations, network topology and the simplification ratio of each inner node. We analyse Isovector aggregation and compare it with Isolines aggregation in a simplified m level binary tree network topology. A contour exists between leaf nodes and $n-1$ level nodes (figure 11). All $m-1$ level nodes are chosen to report. So there are 2^{m-1} reporting nodes in total, which have to transmit data to the root of this tree where the integrated contour will be generated. Suppose, after receiving children reports, each inner node can simplify merged contour by fixed $1-k$ percentage. The size of the new generated contour by a inner node is the $k\%$ of the total contour size received by the node. In this binary tree case, k is in the range $[0.5, 1]$. The packet header size is usually a small constant and we omit it in the analysis.

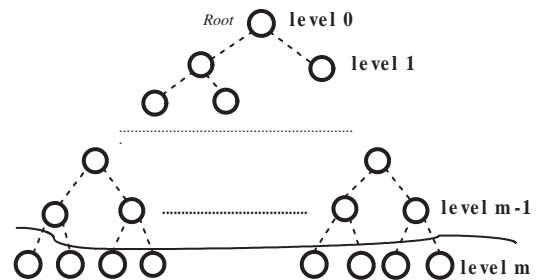


Fig. 11. A simplified binary routing tree and a contour

	Traffic generation	In-network traffic reduction	In-network contour generation	Packet loss handling
no agg.	$O(n)$	No	No	N/A
Polygon agg.	$O(n)$	Yes	Yes	No
Iso-map	$O(\sqrt{n})$	No	No	No
Isolines agg.	$O(\sqrt{n})$	No	No	No
Isovector agg.	$O(\sqrt{n})$	Yes	Yes	Yes

TABLE II
COMPARISONS OF DIFFERENT TECHNIQUES

In Isolines aggregation, each reporting node contains its own information and that of its two children (ID, and Value), which requires 12 bytes in total. In Isovector aggregation, each reporting nodes contains a contour vector start location, end location, head ID, tail ID and the contour value which requires 10 bytes in total. We can see that even when contour vector are not simplified, a reporting node of Isovector aggregation required less bytes than Isolines aggregation, if it has more than one neighbor on the other side of the contour.

Now we count how much data are sent by Isolines aggregation and Isovector aggregation respectively. There is no in-network data reduction mechanism in Isolines aggregation and each reporting message in Isolines aggregation has to travel $m - 1$ hops before reaching the root. So the total size of data transmission is:

$$S_{isolines} = 12 \cdot 2^{m-1} \cdot (m - 1) \quad (1)$$

The complexity of data transmission of Isolines is always $O(m2^m)$.

In Isovector aggregation, after a parent receives 2 reports from its children, it merges them into one. Let the received contour size be s . After merging and simplification, the new contour size will be $2sk$. So the total size of data transmission is:

$$\begin{aligned} S_{isovector} &= 10 \cdot (2^{m-1} + 2k \cdot 2^{m-2} + \dots + 2^{m-1} \cdot k^{m-2}) \\ &= 10 \cdot 2^{m-1} \cdot (1 + k + k^2 + \dots + k^{m-2}) \end{aligned} \quad (2)$$

When $k = 1$, all contours are not simplified after merging and we reach the upper bound of the Isovector aggregation. This situation happens when we want to keep all contour points. In this case, $S_{isovector} = 10 \cdot 2^{m-1} \cdot (m - 1)$. We can see that data transmission by Isovector is less than Isolines aggregation but the complexity is also $O(m2^m)$.

When $0.5 \leq k < 1$, contours are simplified and $S_{isovector} = 10 \cdot 2^{m-1} \cdot \frac{1-k^{m-2}}{1-k}$. In particular, when $k = 0.5$, $S_{isovector} = 10 \cdot 2^{m-1} \cdot \frac{1-0.5^{m-2}}{1-0.5} = 10 \cdot (2^m - 2)$, and this is the lower bound of the Isovector aggregation. The complexity is $O(2^m)$. The total data transmission is reduced by a factor of m compared to Isolines aggregation. When we remove all inner contour points of a contour, we can reach this bound.

VII. EVALUATION

It has been shown that Isolines aggregation performs significantly better than polygon aggregation [4]. In this section, we evaluate Isovector aggregation by simulation in the NS2 [20] environment and compare with Isolines aggregation. The simulation results are also compared with the no aggregation method in which nodes simply send their values to the base station through the routing tree, and negotiations between different nodes are not required. Basically, with the no aggregation method, the base station will get a value from each node and generate the most accurate map. However, the considerable network traffic caused by having all nodes report will bring considerable packet loss, which means the base station can not receive all reports. In the following simulations, Node ID, location information and value are all 2 bytes long.

A. Simulation setup

We use a WSN consisting of 16×16 nodes arranged in a $400m^2$ evenly spaced grid to monitor temperature. The base station is placed in the center of the network. We should point out that, although, in these experiments, nodes are placed according to a grid pattern, similar to Isolines, Isovector is not specific to grid placement. For medium access control, nodes use CSMA at 196Kbps. Their transmission range is set to 40m so each node has 8 neighbors except the outer layer nodes. FLIP [21] is used as the network protocol. The distance tolerance used for the Douglas-Peucker algorithm is 6m and the distance threshold for connecting 2 contour vectors is 14m which is less than the distance between any 2 adjacent nodes. The contour scale is set to 10.

Tree base routing scheme [22], [23] is used for simulations. After the startup of the network, the base station broadcasts a *Query* on its radio. All nodes that hear the query process it and rebroadcast it on to their neighbors. They keep on broadcasting until all nodes in the network have heard the query. In this process, a routing tree will be built. Then each node broadcasts the *Negotiation* message to neighbors and each node initializes its contour ring by *Negotiation* messages received. The neighbor ID and location pair information will be saved. Next time, nodes only have to broadcast *Notification* messages. Until then, the network is fully initialized. Each nodes then start reporting according to *Query* it received.

B. Simulation scenarios and evaluation metric

There are three simulation scenarios for temperature monitoring. The first case includes detecting two straight line contours with value 40 and 50 respectively, and detecting irregular contours. In the second scenario, we change the contour node densities and compare data sent in different contour node ratios. We focus on continuous contour mapping in the third simulation scenario.

We have two criteria for evaluation. (1) Data transmission size which reflects the energy consumption by different approaches. For Isovector Aggregation and Isolines aggregation, data transmission for negotiation between nodes is also included. (2) Contour map similarity which corresponds to the

	Similarity	Data sent (Bytes)
no agg.	96.64% (sd 0.49%)	12111 (sd 534)
Isolines agg.	94.94% (sd 1.16%)	6177 (sd 345)
Isovector agg.	96.03% (sd 0.51%)	4175 (sd 263)

TABLE III
CONTOUR MAP SNAPSHOT

query precision by different approaches. The contour map similarity is calculated as the percentage of points (80*50 points are placed on the map) that are actually in correct value ranges when compared to the baseline map which is generated using all node values. ArcView GIS is used for interpolation and visualization. We should point out that only the no aggregation method and Isolines aggregation need interpolation by ArcView GIS. Isovector does not need to do this since the contour generation is part of the algorithm.

C. Static contours

We have a regular contour map for evaluation. This map includes two vertical contours with value 40 and 50 respectively in the network. The contour node ratio in this case is 25 percent. By Isovector aggregation, the base station only receives two contour vectors and each contour vector only consists of two points. This implies that Isovector aggregation only uses a start point and an end point to represent a straight line. An example of received contour vector is 40:(99, 387):(99, 12). Here 40 means the contour value. (99, 387) is the start point coordinate and (99, 12) is the end point coordinate. In the all 10 run simulations, Isovector aggregation produces exact contours. The data size transmitted by Isovector aggregation is 3102 bytes (sd 199), whereas 5025 bytes (sd 494) are transmitted by Isolines aggregation and 12331 bytes (sd 429) are sent by the no aggregation method.

An irregular contour map which is similar to the one used in Isolines [4] is also used for evaluation. Figure 12 shows the baseline map and examples of maps generated using reporting data of the no aggregation method, Isolines aggregation and Isovector aggregation respectively. Table III gives the simulation results.

As we can see from the table III, after drawing contours generated by Isovector aggregation, we get a contour map which is highly similar to the baseline map. Isovector sends much less data than Isolines aggregation. For the no aggregation method, we find that nearly 40 percent reports, including many reports sent by contour nodes, are automatically dropped by the network. Hence, the no aggregation method does not achieve significantly better map similarity than the other approaches.

D. The impact of contour node densities

In this scenario, we change the contour node percentage by setting different contour tiers in the network. We query all existing contours and measure the total data sent by no aggregation, Isolines aggregation and Isovector aggregation. Figure 13 shows the result. From the figure we know that both aggregation methods send more data as contour node ratio increases, because more nodes can detect contours. As

	Similarity (9s)	Data sent (Bytes)
no agg.	98.0% (sd 0.72%)	110994 (sd 3074)
Isolines agg.	97.7% (sd 0.48%)	26855 (sd 839)
Isovector agg.	98.5% (sd 0.2%)	16617 (sd 807)

TABLE IV
MOVING CONTOUR

more redundant points are removed from the reports, Isovector aggregation sends much less data in the case of a high contour node ratio when compared to Isolines aggregation. This implies that Isovector aggregation is more scalable for mapping dense contours in WSNs. Because the no aggregation method lets all nodes report, the contour node ratio has no impact. This result shows that Isovector aggregation is also applicable for dense contour mapping.

E. Moving contours

In our continuous mapping scenario, we simulate a front moving in from left to right. Temperature increases from the thirties to the fifties in about 50 meters. The front moves to the right in 9 seconds. The starting value of all points is centered at 35 degrees. The base station, which is placed at the center of the map, starts by initializing the network at time 1s. From time 3s to 11s, nodes report their temperature values in each second. The simulation is stopped at time 12s.

We count the total data sent in this moving scenario and take map snapshots at 9s for comparison. Figure 14 shows the examples of contour map snapshots at 9s and table IV gives the simulation results. From table IV we know that, benefiting from contour vector merging and simplification, Isovector aggregation also sends much less data than Isolines aggregation in the continuous monitoring scenario. The contour map similarity by Isovector is slightly better than Isolines. No aggregation technique sends much more data than Isovector and Isolines. This result implies that Isovector aggregation is proper for continuous contour mapping.

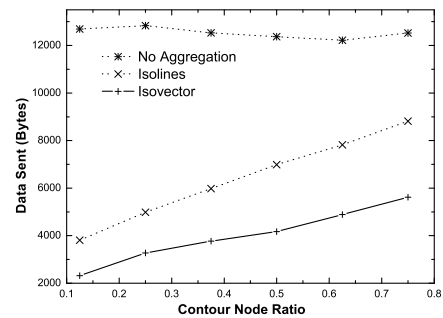


Fig. 13. Data sent at different contour node ratios

In certain cases, it is necessary to increase node density to get a fine contour map. With the \sqrt{n} traffic generation and in-network traffic reduction mechanism, Isovector aggregation

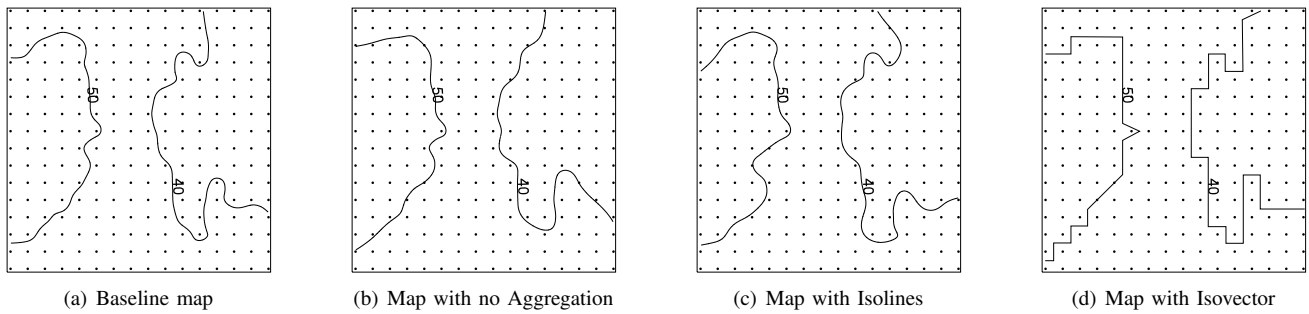


Fig. 12. Map snapshots for the irregular contours

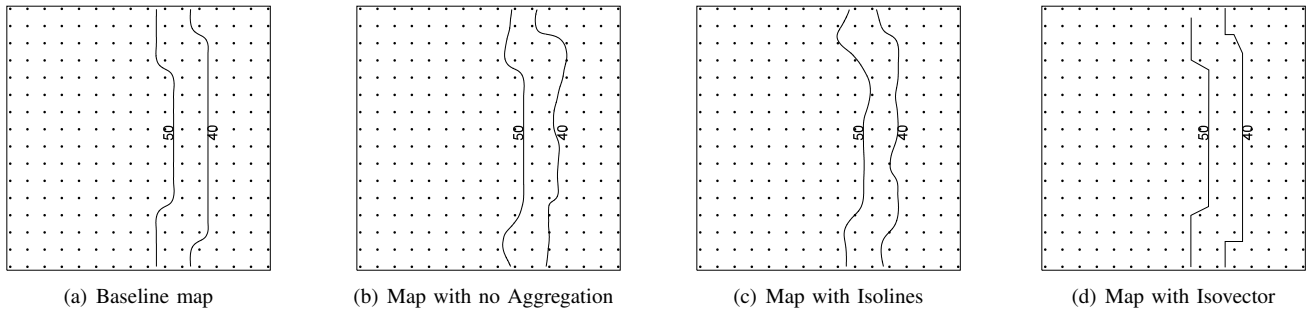


Fig. 14. Map snapshots at 9s

has the better scalability than other methods. It is also important to mention that only lossless no aggregation method can achieve 100 percent map similarity under our map similarity metric. Because we use mid-points between nodes to represent contours, Isovector aggregation might not be able to achieve 100 percent map similarity even in a dense network. In a real sensor network, especially in a dense network, the huge traffic cause by no aggregation method makes the perfect network transmission unrealistic.

VIII. CONCLUSION AND FUTURE WORK

Isovector aggregation is a new approach to achieve energy conservation in WSNs. Its main advantages are its $O(\sqrt{n})$ traffic generation, in-network contour generation and simplification. All these factors make Isovector aggregation energy efficient and very scalable. Our simulation results suggest that Isovector aggregation achieves good performance and energy saving compared to other approaches.

There is a dense contour case that Isovector aggregation cannot handle properly. Isovector aggregation reports at most two contours with different values between two neighboring nodes. If two neighboring nodes have large value differences, then it is possible that some contours between them will not be reported. This is the subject of future work. We also plan to extend Isovector aggregation to support different region and range related queries in the future.

REFERENCES

- [1] D. J. Abadi, S. Madden, and W. Lindner, "Reed: Robust, efficient filtering and event detection in sensor networks." in *VLDB*, 2005, pp. 769–780.
- [2] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek, "Beyond average: Toward sophisticated sensing with queries." in *IPSN*, 2003, pp. 63–79.
- [3] Y. Yao and J. Gehrke, "Query processing in sensor networks." in *CIDR*, 2003.
- [4] I. Solis and K. Obraczka, "Efficient continuous mapping in sensor networks using isolines." in *Proc. of the 2005 MobiQuitous*, 2005, pp. 325–332.
- [5] X. Meng, L. Li, T. Nandagopal, and S. Lu, "Event contour: an efficient and robust mechanism for tasks in sensor networks," *Technical Report, UCLA*, 2004.
- [6] G. J. Pottie and W. J. Kaiser, "Wireless integrated network sensors." *Communications of the ACM*, vol. 43, no. 5, pp. 51–58, 2000.
- [7] V. Shnayder, M. Hempstead, B. rong Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications." in *Sensys*, 2004, pp. 188–200.
- [8] K. Chintalapudi and R. Govindan, "Localized edge detection in sensor fields." *Ad Hoc Networks*, vol. 1, no. 2-3, pp. 273–291, 2003.
- [9] M. Ding, D. Chen, K. Xing, and X. Cheng, "Localized fault-tolerant event boundary detection in sensor networks." in *INFOCOM*, 2005, pp. 902–913.
- [10] Y. Liu and M. Li, "Iso-map: Energy-efficient contour mapping in wireless sensor networks," in *ICDCS*, 2007, p. 36.
- [11] A. Silberstein, R. Braynard, and J. Yang, "Constraint chaining: on energy-efficient continuous monitoring in sensor networks." in *Proc. of the 2006 ACM SIGMOD Intl. Conf. on Management of Data*, 2006, pp. 157–168.
- [12] J. Zhao, R. Govindan, and D. Estrin, "Residual energy scans for monitoring wireless sensor networks." in *IEEE Wireless Communications and Networking Conference*, 2002, pp. 145–156.
- [13] W. Xue, Q. Luo, L. Chen, and Y. Liu, "Contour map matching for event detection in sensor networks." in *SIGMOD Conference*, 2006, pp. 145–156.
- [14] "Contour line," http://en.wikipedia.org/wiki/Contour_line.
- [15] X. Cheng, A. Thaler, G. Xue, and D. Chen, "Tps: A time-based positioning scheme for outdoor wireless sensor networks." in *INFOCOM*, 2004.
- [16] Y. Shang, W. Ruml, Y. Zhang, and M. P. J. Fromherz, "Localization from mere connectivity." in *MobiHoc*, 2003, pp. 201–212.
- [17] I. Solis and K. Obraczka, "The impact of timing in data aggregation for sensor networks." in *Proc. of the IEEE Intl. Conf. on Communication*, 2004.
- [18] D. Douglas and T. Peucker, "Algorithms for the reduction of the number

of points required to represent a digitized line or its caricature." *The Canadian Cartographer*, vol. 10, pp. 112–122, 1973.

- [19] E. White, "Assessment of line-generalization algorithms using characteristic points." *The American Cartographer*, vol. 12, pp. 17–27, 1985.
- [20] "Ns 2," <http://www.isi.edu/nsnam/ns/>.
- [21] I. Solis and K. Obraczka, "Flip: A flexible interconnection protocol for heterogeneous internetworking." *MONET*, vol. 9, no. 4, pp. 347–361, 2004.
- [22] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: A tiny aggregation service for ad-hoc sensor networks." in *Proc. of the 2002 USENIX Symp. on Operating Systems Design and Implementation*, 2002.
- [23] J. M. Hellerstein, W. Hong S. Madden and M. J. Franklin, "The design of an acquisitional query processor for sensor networks." in *SIGMOD Conference*, 2003, pp. 491–502.