Iso-Contour Queries and Gradient Descent with Guaranteed Delivery in Sensor Networks

Rik Sarkar* Xianjin Zhu* Jie Gao* Leonidas J. Guibas† Joseph S. B. Mitchell‡

* Department of Computer Science, Stony Brook University. {rik, xianjin, jgao}@cs.sunysb.edu
† Department of Computer Science, Stanford University. guibas@cs.stanford.edu
‡ Department of Applied Mathematics and Statistics, Stony Brook University. jsbm@ams.sunysb.edu

***Abstract*—We study the problem of data-driven routing and navigation in a distributed sensor network over a continuous scalar field. Specifically, we address the problem of searching for the collection of sensors with readings within a specified range. This is named the *iso-contour query* problem. We develop a gradient based routing scheme such that from any query node, the query message follows the signal field gradient or derived quantities and successfully discovers *all* iso-contours of interest. Due to the existence of local maxima and minima, the guaranteed delivery requires preprocessing of the signal field and the construction of a *contour tree* in a distributed fashion. Our approach has the following properties: (i) the gradient routing uses only local node information and its message complexity is close to optimal, as shown by simulations; (ii) the preprocessing message complexity is linear in the number of nodes and the storage requirement for each node is a small constant. The same preprocessing also facilitates route computation between any pair of nodes where the the route lies within any user supplied range of values.**

I. Introduction

Wireless sensor networks have shown great potential for providing dense monitoring and sensing capabilities with modest cost and management effort. In many typical sensor network applications, sensors are densely deployed in a physical environment to provide good coverage at fine sensing resolutions. Existing work has established many fundamental mechanisms for sensor deployment to ensure coverage [1]–[4] as well as energy efficient networking functions to collect data from these nodes.

There are two fundamental aspects of sensor networks that differentiate them from other types of wireless networks. First, it is the data from the sensor nodes, rather than the network nodes themselves, that is of most interest to the users. While many wireless networks, such as wireless LANs, cellular networks, and ad hoc mobile networks, focus on supporting low-latency end-to-end communications and maximizing the system throughput, sensor network designs are often tailored towards their target application and are bound tightly to the physical environment that they are supposed to monitor/sense. In the most prevailing applications of environmental monitoring, sensors measure readings of the physical space, such as temperature, pressure, chemical concentration, and many others. Such physical quantities often exhibit continuity properties over space and/or time. Thus the smoothness of the physical signal field, and the spatial correlation of discrete sensor data, naturally suggest possibilities for data compression and exploitation for efficient system design.

A second unique property of sensor networks resides in their great potential in allowing seamless interaction between users and the physical world. In many civilian and military applications, the users operate in the same space in which the sensors are embedded. This allows novel applications in which real-time sensor data is quickly delivered to users of interest for appropriate response and actions. All of this eventually leads to a smart environment that could revolutionize the way we observe, interact with and influence the physical world.

In this paper we look at the iso-contours of a scalar signal field represented by sensor data, together with a local gradient descent routing scheme, with which the users can navigate in this signal field with guaranteed success.

Iso-contour related queries. For a continuous field, an iso-contour at an isovalue x is the collection of points with value equal to x. In a discrete sensor network, this is often approximated by the collection of sensors with readings sufficiently close to x. The iso-contours encode spatial structures of the signal field, such as boundaries of the 'hot' regions that indicate overheating or a fire, or pollution dissemination that may require special treatment. The signal field can also be the energy map or traffic load on the networked sensors, and thus the iso-contours are related closely and provide information about the general health of the network or its traffic bottlenecks.

A few papers have studied compression, approximation and aggregation of iso-contours with space-efficient data structures, when sensors report their data along an aggregation structure to the base station [5]–[7]. In this paper we are interested in in-network data processing and the usage of iso-contours for navigation in the signal field. Consider a scenario in which sensors and users (such as rescuers or patrol officers) are embedded in the same physical space. Users with hand-held devices communicate with nearby sensors to obtain directions to places that require attention or service, indicated by the sensor data being within a specified range. We consider the following two routing and navigation functions:

- *Iso-contour query*: from a query node q, find the iso-contours at value x, or count/report iso-contour components at given value/range.
- *Value-restricted routing*: find a path from a source node s to a destination node t with all values on the path within a user-specified range. This can be used for navigation of packets in the network (e.g., avoiding sensor nodes with

low energy level), or navigation of objects in the physical environment (e.g., avoiding traffic jam).

For both problems, we are looking for efficient solutions without flooding all the nodes. The paper is tailored by the iso-contour query as it best demonstrates the basic idea, with which the value-restricted routing can be answered easily.

Gradient descent routing. The most intuitive solution for iso-contour queries is to use gradient descent, by exploiting the natural continuity of the signal field. Starting from the query node q, the query message can be greedily guided either downhill or uphill, depending on the comparison of the value at q and the target value x. This greedy descent routing is simple and requires only local knowledge. Thus it has been explored in a number of settings for low-cost data-centric routing [8]–[12]. Greedy descending/ascending can typically lead the query message to one iso-contour, unless the query message reaches a local minimum or local maximum, in which case the query gets stuck. Indeed, using simple gradient descent for an iso-contour query has a serious defect: the signal field may have multiple peaks and valleys, and greedy descending discovers at most one iso-contour, and is not able to discover all of the iso-contours due to the existence of local optima.

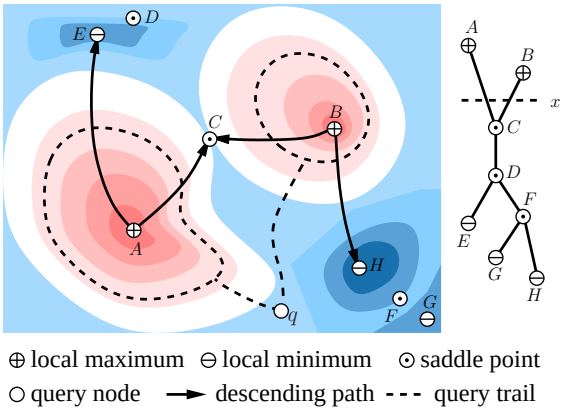

Fig. 1. The level sets of a signal field and the contour tree spanning all the critical points (in the right). The figure also shows *some* descending paths connecting the critical points.

Figure 1 shows an example of a potential field by drawing its level sets. Red colors mean hot and blue colors mean cold. We also show all the local maxima, minima and saddle points. A greedy gradient routing from a query node q looking for a desired level contour will follow the local gradient and climb up the mountain. Once the query reaches the desired level it can locally trace out one contour, e.g, the contour on the left peak in the figure. However with only local information the query does not know whether there are other peaks and if so where they are.

The difficulty here is that the greedy gradient routing is completely local, while iso-contours reflect the global topology of the signal field. This is a general problem in navigation with a potential field, as has also been studied in robotics: with only information about the local potential one lacks the big picture of the signal field which is important for guaranteed success. In particular, the collection of critical points (local maxima, minima and saddle points) represents the global topology of the signal field. Thus, in order to make the local greedy descend algorithm always work, one needs to augment it with a compact representation of the critical points and their relationships.

Our contribution. We propose to investigate distributed algorithms to pre-process the iso-contour structures of the signal field by what is called the *contour tree* [13], using which a gradient routing scheme can successfully discover *all* iso-contours. In short, a contour tree is a tree on all the critical points of the signal filed and captures the topology of the iso-contours. It is a special case of the *Reeb graph* in Morse theory [14]. Take Figure 1 as an example, the right figure shows the topological contour tree consisting of eight vertices, corresponding to two local maxima, three local minima, and three saddle points. A contour tree captures how the connected components of the iso-contours merge/split as we increase/decrease the isovalue.

We propose an algorithm for the construction of the contour tree in a fully distributed manner. The basic idea is similar to the centralized construction [13], [15]–[17]. But we need to account for numerous robustness issues due to local noise and degeneracies, and lack of global coordination. We use distributed sweeps [18], initiated at local maxima and minima to identify the saddle points and nodes on the saddle contour. Next an information dissemination phase following the contour tree structure distributes necessary information for gradient descent routing. The preprocessing involves all together four rounds of sweeps of the signal field and has a linear message complexity.

The invariant we maintain on a node p is the max/min value in the interior and exterior of the iso-contour component through each point p. This represents only a small constant storage requirement at each node. For iso-contour queries, the gradient descent routing alternates between two operations (i) at a node on some saddle contour, it checks the split/merged contours and send one or two (if necessary) messages to the new connected components. (ii) at other nodes, the query message either follows an iso-level or follows gradient ascending/descending path to reach the desired contour. The gradient routing only uses information stored at a node itself and every routing step is justified, in the sense that there will definitely be a contour discovered for each query message. Thus no effort is wasted. Our simulations show that the gradient routing achieves comparable message complexity, when compared with the minimum spanning tree covering the iso-contour components, which is at most twice the length of the minimum Steiner tree, the optimal solution if the global knowledge about the entire signal field were available.

At the same time, the same contour tree permits a scheme for restricted value routing, and a labeling scheme such that validity of a restricted value route request can be determined simply from the labels of the source and destination nodes. Intuitively, the spatial structures of the signal field are entirely captured by the contour tree, and low values paths in the field can be mapped to a low value path on the tree.

Lastly we note here that in this paper we only consider a static signal field, because the problem for a static signal field is

already quite challenging. In practice, as the signal evolves over time we can periodically execute the contour tree construction phase. The maintenance of the contour tree for a time-varying signal field will be future work.

II. Contour Trees and Gradient Routing

Given a continuous signal field $\mathcal{F}$, the *iso-contour* (aka. *level sets*) at an *isovalue* x is the collection of points p with value $\mathcal{F}(p) = x$, and may have multiple connected components. We denote by C one connected component of an iso-contour and by $C(p)$ the connected component containing node p.

As we decrease the isovalues from the global maximum to the global minimum, the connected components on the iso-contours may merge together, split, emerge, or disappear. These changes happen at *critical points*, such as *local minima*, *local maxima* and *saddle points*. The *contour tree* captures such topological changes of the iso-contours. In a contour tree, each node corresponds to a critical point, and an arc in the contour tree connects two critical points. In particular, as we start from $+\infty$ and decrease the isovalue,

- at a local maximum, a contour component emerges;
- at a local minimum, a contour component vanishes;
- at a saddle point, two contour components merge into one or one contour component splits into two (see Figure 2). In the first case, there are two branches of the iso-contour eminating from the saddle point, representing the two components. Such a saddle point is called a *merge saddle* (with respect to decreasing isovalues). The second case with respect to decreasing isovalues is called a *split saddle*. (For increasing isovalues, split saddles and merge saddles are interchanged.)

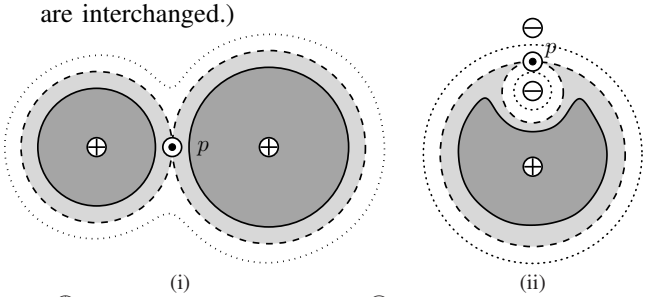

Fig. 2. $\oplus$ indicates a local maximum. $\ominus$ indicates a local minimum. $\odot$ indicates a saddle point. Dark colors mean larger values. When we start from ∞ and decrease the isovalue, at a saddle point, (i) two contour components merge into one; (ii) one contour component splits into two.

It has been proved that the merging and splitting of contour components are indeed represented by a tree. Further, without degeneracy (no two saddle points have the same values), a local maximum or a local minimum has degree 1; a critical point has degree 3. To visualize, we place the vertices of a contour tree, i.e., the critical points, at the height levels of their values. A merge saddle has a 'Y' shape and a split saddle has an inverted 'Y' shape. Then we can map contour components in an iso-contour at value x to the points obtained by cutting the tree at level x. The contour component through a saddle is mapped to the saddle vertex on the tree. Thus, at a point p in the signal field, if its contour component $C(p)$ is mapped to a point on arc α in the contour tree, then we say p is on arc α. In Figure 1, the iso-contour at the query value x has two components, the left contour stays on the arc AC and the right one stays on the arc BC. If we embed the contour tree in the domain by representing each edge with a monotonic path connecting the corresponding critical points, it can be verified that this mapping is continuous and the contour tree is a retract of the original domain under this mapping.

In a sensor network the continuous signal field is sampled by discrete sensors. To compute the contour tree in the discrete setting, we have the following challenges:

Local identification of critical points. In a continuous signal field, a critical point p is a point with all partial derivatives vanishing at p. In a sensor network we can easily identify the local maxima and local minima. A local maximum (minimum) has all the neighboring values no greater (smaller) than itself. However, it is not easy to identify saddle points, which have larger and smaller values in its neighborhood in an alternating way. When we do not have sensor locations or do not have accurate locations (say, the neighbors may switch their angular ordering), identifying a saddle point robustly is not straightforward. In our algorithm, the saddle points are discovered along with the construction of the contour tree, as the nodes where the contour components merge.

Distributed construction of the contour tree. The construction of the contour tree of a piecewise linear mesh has been studied before [13], [15]–[17], [19]. The best algorithm achieves a running time of $O(n \log n)$ on a piece-wise linear surface with n vertices and can even be made to be output sensitive [16], [19]. However, these algorithms are centralized and are not appropriate for low-cost in-network processing in a distributed sensor network. We propose a distributed algorithm that involves four passes of sweeps, to be explained in details in subsection II-B. Thus the construction costs roughly $4n$ message transmissions. After the preprocessing phase gradient-based routing with guaranteed success for iso-contour queries can be performed at *any* node in the network.

Handling noises and plateau regions. An important practical issue regarding contour trees for a sensor network is that the sensor data is a noisy approximation of the underlying smooth signal field, due to sensor inaccuracy, hardware noise, etc. Thus there could be many more local maxima and minima in the sensor data than the the original (unknown, smooth) signal field. We propose two methods to handle this. First, we will locally simplify a contour tree by using topological persistence [20]. Small bumps will be chopped off. Second, we will not keep the entire contour tree at each node but rather only keep enough information for gradient routing. Thus, local optima due to noises in the measurement only influence a small neighborhood and are 'invisible' to queries from far away regions.

A. Notations

Before we describe the algorithm, we first state conceptually what we want to achieve with the contour tree construction and what we want to store at each node. An example of a

contour tree is given in Figure 3 (i). A node w on an arc AB has a contour component $C(w)$ in between $C(A)$ and $C(B)$. The contour component $C(w)$ decomposes the entire signal field into two components, the *interior* and the *exterior*, corresponding to the two subtrees when w is removed. The interior contains the critical point A, which is reachable from $C(w)$ via a gradient ascending path. We call A the *ascending saddle*. The exterior contains the critical point B, which is reachable by a gradient descending path. B is called the *descending saddle*. Not every node has both ascending/descending saddles. Now

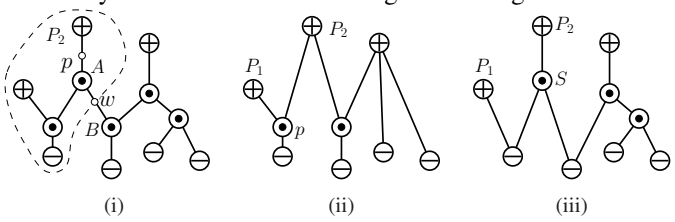

Fig. 3. (i) A contour tree and the interior of $C(w)$ shown in the bounded region; (ii) merge tree; (iii) split tree.

we will state what is needed to store at each node for gradient descent routing with success.

At a node w (not on the contour component of a saddle), we will store four values:

- $I^+(w), I^-(w)$ correspond to the maximum and minimum value in the interior of $C(w)$;
- $E^+(w), E^-(w)$ correspond to the maximum and minimum value in the exterior of $C(w)$.

This information is to guarantee that when we send a query message either uphill or downhill, we know for certain that there exist some contours for which we are looking.

For the consideration of easy navigation with the contour tree, each node also keeps information about the contours that split off/merge together at their *ascending merge saddle* or *descending split saddle*. Take point p on the arc P_2A and its descending split saddle A in Figure 3 (i) as an example. The contour component $C(A)$ is the union of two contours $C_1(A)$ and $C_2(A)$ splitting up soon. Thus we keep at each node $u \in C(p)$,

- the maximum/minimum values of the interior/exterior of both $C_1(A)$ and $C_2(A)$;
- gradient descending pointers leading to $C_1(A)$ and $C_2(A)$.

This information helps us decide before we reach a saddle contour, whether it is worth visiting one or two of the contour components that split off of it and if so, how to get there.

To summarize, each node only keeps a small constant amount of information. Next we will explain how to get this information. In the rest of this paper we assume a dense deployment of sensors in which each node u has an value $\mathcal{F}(u)$. The nodes have a communication graph G that models the pairs of nodes who can directly communicate with each other. We assume no two sensors have the same values, if they do, ties are broken by their IDs (the one with higher ID is considered larger).

B. Sweep to identify saddle points

The construction of the contour tree and the spread of information about the peaks/valleys of the signal field are conducted by a sweep algorithm, similar to the one in [18]. Without loss of generality, we explain the details with the sweep top down. A node has its *higher (lower) neighbors* as the subset of neighbors with value strictly higher (lower) than itself.

Each sweep is initiated and labeled by a critical node (a maximum, minimum or a saddle node). A node identifies itself as a local maximum if it discovers that all its 1-hop neighbors have value no greater than itself. It then initiates a sweep top down. The sweep algorithm runs in a distributed fashion on all the nodes. A node has two possible states, *swept* and *not swept*. Each local maximum node initializes itself as a swept node. When a node has all of its higher neighbors in the swept state, it changes itself to be swept. The nodes who participate in the sweep do not need to be synchronized and advance the sweep frontier with their local knowledge.

In the sweep initiated by a local maximum p, the sweep message carries the tuple $(p, \mathcal{F}(p))$, i.e., the node ID and value of p. Each node being swept will keep this information, as well as from which nodes it received this information. We define a *descending path* as a path in which each node has a value no greater than its precedent. During the sweep the information about a local maximum p is propagated along descending paths from p. In addition, each node swept learns ascending pointers which eventually lead to the local maximum.

If a node gets two sweep messages from different local maxima, this indicates that two contour components start to merge. Thus a saddle should be identified. Since the nodes advance the sweep frontier in a distributed fashion, it may happen that two nodes at the same time both receive the sweep messages from two peaks. Thus we will need to define a saddle rigorously and resolve the ambiguity.

Definition 2.1. *We define a node to be a* merge saddle node *if it is the one with highest iso-value with two descending paths from different critical points (other merge saddles or local maxima), i.e., it receives two sweep messages from different critical points.*

Notice that this definition is recursive in nature. A merge saddle is precisely the first node when two contour components merge, as shown below.

Lemma 2.2. *For a merge saddle q of two critical nodes p_1, p_2, if we remove the sensors with values strictly smaller than $\mathcal{F}(q)$, and obtain a subgraph G', then q is the cut node of G' (removing q will result in two or more disconnected components.).*

Proof: First, define L_1 (L_2) as the set of nodes in G' with ascending paths to p_1 (p_2). We claim that L_1 and L_2 has only node q in common. If otherwise, $q \neq q' \in L_1 \cap L_2$. Since q' is a node in G', q' has a higher value than q. Now this contradicts with the definition that q is the saddle node.

Now we argue that once q is removed from G', then the set of nodes L_1 is disconnected from L_2. Suppose otherwise, that there are two nodes $x_1 \in L_1$, $x_2 \in L_2$ and a path $\mathcal{P}$ connecting them that does not go through q. This path $\mathcal{P}$ must use nodes other than those in $L_1 \cup L_2$. Now take the first node on $\mathcal{P}$ coming out of x_1 that is not in $L_1 \cup L_2$, denoted by y_1. Without loss of generality we can also assume that y_1 is just

next to x_1 (otherwise take x_1 to be the preceding node of y_1). Now we must have $\mathcal{F}(y_1) > \mathcal{F}(x_1)$, since y_1 is not in L_1. Now take an ascending path from y_1, it will lead eventually to either a local maximum or a saddle, denoted by p_3. Thus the node q cannot be a merge saddle with p_1, p_2, since there will be another saddle of p_1 and p_3, which is at least higher than node y_1 and q. This shows a contradiction. ■

We also remark that with a top-down sweep we do not identify the saddle when one contour component splits into two – the split saddles will be discovered when we do a bottom-up sweep from local minima, in a completely symmetric fashion.

Now we show how the merge saddle node is identified in a robust and efficient way. A node who is not a local minimum and first receives two sweep messages from different peaks P_1, P_2 will promote itself to be a *potential merge saddle* $S(P_1, P_2)$. In a distributed setting we need to worry about two issues: (i) two nodes u, v (or more) may become potential merge saddles $S(P_1, P_2)$ for the same two peaks. In this case only the real saddle node (the one with highest isovalue) should survive. (ii) it may happen, if the sweep frontier does not proceed in the same speed, that the lower saddle may be discovered before the higher saddle, as shown in Figure 4.

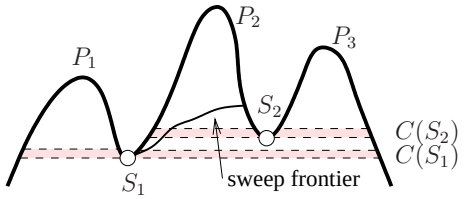

Fig. 4. If the sweep from P_2 proceeds faster and reaches S_1 before it reaches S_2, then S_1 will notice it is a potential saddle for $S(P_1, P_2)$. The correct contour tree should have the saddle S_1 to be the merge saddle for P_1, S_2.

The two problems will be resolved by the traversal of contour component, described below. Once a node u becomes a potential saddle for two peaks p_1, p_2, it starts to traverse the contour component $C(u)$, defined as,

Definition 2.3. *The contour component $C(u)$ for a node u is defined as the set of nodes that have values above $\mathcal{F}(u)$ and have a lower neighbor lower than $\mathcal{F}(u)$. If u is a merge saddle for two critical points p_1, p_2, then $C(u)$ is partitioned into two components $C_1(u)$, $C_2(u)$ (sharing only node u) that have ascending paths to p_1, p_2 respectively.*

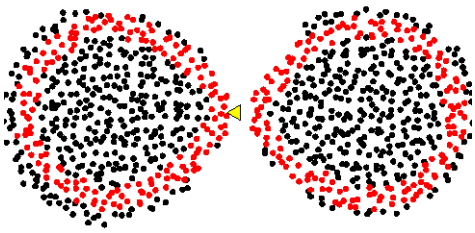

Fig. 5. A merge saddle (shown as the triangle) and its contour component (in red).

Thus a potential saddle node u sends the tuple (p_1, p_2) to the nodes in $C(u)$. To resolve the first issue that several potential saddle nodes compete for the real saddle, the traversal message from u is suppressed if it hits a node x with a traversal message from a winning potential saddle u' (with higher value) for the same two peaks. x stops forwarding the message from u. Thus the traversal from u will stop because it either visits all the nodes in $C(u)$ or if u loses to some other potential saddle.

During the message dissemination, a backward pointer is cached at each node in the traversal. Thus a tree rooted at u, named $T(u)$, is established and used for information aggregation and for u to learn about whether it wins and becomes the real saddle, or whether it loses the competition. In particular, a leaf w in this aggregation tree will return to its parent 'loser' if w has another winning traversal message, otherwise return the sweep message it has received. If a node is not yet swept, it waits for its sweep message before it reports back. An interior node in the aggregation tree returns to its parent the union of the messages from its children. Now the potential saddle node u becomes the real saddle for p_1, p_2, if (i) it does not get the 'loser' message from its aggregation tree; (ii) all nodes in $C(u)$ are swept by p_1 or p_2.

The new saddle q will start with a new top-down sweep and they propagate the tuple $(q, \mathcal{F}(q), M(p_1, p_2))$, where $M(p_1, p_2)$ indicates that q is the merge saddle of two critical points p_1, p_2. All the nodes in $C(q)$ are considered swept by q and the new sweep moves forward.

Notice that the sweep from a merge saddle q is distinct from the sweeps from p_1, p_2. In fact, the merge saddle q and all the nodes who receive the traversal message from q do not forward the sweep from p_1 or p_2 anymore. In the case when a node w has already forwarded the sweep from p_1 or p_2 by the time it gets the traversal message, it simply participates in the new sweep of q. Notice that again we do not require synchronization. The old sweeps from p_1 and p_2 cannot propagate very far from $C(q)$, since q stopped participating; thus, q's lower neighbors cannot possibly be swept, and so on and so forth.

If the merge saddle q also happens to be a local minimum (in a setting with low discrete resolution), q is in fact a merge saddle, a split saddle, and a local minimum all by itself. One trouble this may potentially cause is that the old sweeps from p_1 and p_2 may propagate without being dragged behind by q, since q does not have lower neighbors. The system however, will eventually arrive at the correct state, since the sweep from saddle q will overwrite the old sweeps from p_1, p_2.

The traversal also resolves the second issue mentioned above. In particular, S_1 cannot win before the saddle S_2 successfully identifies itself and proceeds with its sweep — this is because S_1 will only get its aggregated message when all the nodes in $C(S_1)$ have been swept, and S_2 and its descendants cannot be possibly swept before the saddle S_2 is done. During the aggregation phase for S_1, S_1 will learn about the sweep messages on $C(S_1)$. A subtle issue is that some nodes in $C(S_1)$ may considered them swept by P_2 and report P_2 back to S_1. Thus S_1 learns that some of the nodes are swept by $S_2 = M(P_2, P_3)$ and some nodes are swept by P_2 or P_3 alone. Now S_1 un-sweeps the nodes only swept by P_2 or P_3 and will

only proceed to be the real saddle for P_1, S_2 when all the nodes are swept by P_1, S_2. In this case S_1 is initially proposed to be a saddle for P_1, P_2 but eventually becomes a saddle of P_1, S_2 when it wins.

To summarize, If there are two nodes both identifying themselves as a merge saddle, then the one with lower value will be swept and corrected (i.e., removed) eventually.

Lemma 2.4. *With the algorithm above, there cannot be two nodes both identifying themselves as the merge saddle of two critical nodes. Thus the algorithm defines a unique contour tree structure.*

After the top-down sweep, we have identified all merge saddles. By symmetry, we perform another sweep bottom-up initiated by local minima. Thus, after both sweeps we identify all saddle points and all nodes on the contour components of these saddles, thereby obtaining inherently the entire contour tree structure.

C. Construction of the contour tree

In this section we will extract the combinatorial contour tree after the saddles are identified. Notice that during top-down and bottom-up sweeps we have identified the merge tree (on all local maxima/minima and merge saddles) and the split tree (on all local minima/minima and split saddles). We will combine them to the contour tree such that each critical node learns its parent/child on the tree. Figure 3 (ii) (iii) shows the merge tree and split tree, respectively.

We use descending and ascending paths to discover the contour tree. Starting from a merge saddle $p = M(P_1, P_2)$, we follow ascending paths towards P_1, P_2 respectively. If the ascending path towards P_1 reaches P_1 before it hits any other critical contour level, then p will consider P_1 its parent in the contour tree. If the ascending path towards P_2 hits a split saddle contour S, then p will consider S as its other parent in the contour tree. Similarly p also sends a descending path and identify its child in the contour tree. The operations for a split saddle, maximum/minimum are very similar and not repeated.

The operations require that an ascending path does not cross a split saddle contour without noticing it. This is guaranteed by the definition of a contour component. Suppose that in an ascending path x has value $\mathcal{F}(x) < \mathcal{F}(q)$, with q as a split saddle node, and the next node on the path y has value $\mathcal{F}(y) \geq \mathcal{F}(q)$. Thus x must be on the saddle component $C(q)$, because x has a value below $\mathcal{F}(q)$ and has a neighbor y above it. This guarantees that the contour tree will be detected precisely as the combination of the merge tree and the split tree.

D. Information dissemination

With the contour tree constructed, we will need to disseminate information such that each node w learns

1) the maximum/minimum value, $I^+(w), I^-(w)$, inside the *interior* of its contour component $C(w)$;
2) the maximum/minimum value, $E^+(w), E^-(w)$, inside the *exterior* of $C(w)$.

This is done by information dissemination along the contour tree. By symmetry, we first explain how a node w learns about the maximum value inside the interior/exterior of its contour component. Suppose that w is on an arc AB. Recall that the interior of $C(w)$ corresponds to the subtree containing the ascending neighbor A, when $C(w)$ is removed. Thus the maximum of the exterior (interior) of $C(p)$ for a local minimum (maximum) p is its own value.

We explain the basic operation by using the contour tree. For an arc e, the removal of $w \in e$ leaves two subtrees T_1 and T_2, the maximum value in T_1 is sent through the arc, by a sweep, to T_2, and vice versa. In particular, we specify the dissemination rules at saddle points. See Figure 6. First, examine a merge

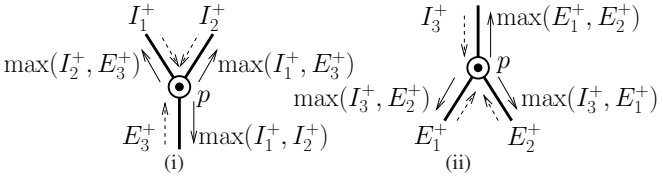

Fig. 6. Information dissemination on (i) merge saddle; (ii) split saddle.

saddle p with two incoming arcs e_1, e_2 and one outgoing arc e_3. Suppose by induction that the maximum is already learned and propagated along the arcs e_1, e_2, e_3 to saddle p, as shown by I_1^+, I_2^+, E_3^+ in the figure. The contour component $C(p)$ has two components $C_1(p)$, $C_2(p)$, corresponding to the nodes with ascending paths along e_1 and e_2 respectively. Now for a node w,

- if $w \in C_1(p)$, w sends $E_1^+ = \max(I_2^+, E_3^+, \mathcal{F}(w))$ along the bottom-up sweep of e_1;
- if $w \in C_2(p)$, w sends $E_2^+ = \max(I_1^+, E_3^+, \mathcal{F}(w))$ along the bottom-up sweep of e_2;
- if $w \in C(p)$, w sends $I_3^+ = \max(I_1^+, I_2^+)$ along the top-down sweep of e_3.

This says that the nodes on $C(q)$ will initiate a sweep bottom-up along e_1 and e_2, and a top-down sweep along e_3 and propagate information as shown in Figure 6(i) accordingly. At a split saddle, information propagates in a similar way as shown in Figure 6(ii). We do not repeat here.

Notice that we do not need close synchronization among these sweeps. In particular, the bottom-up sweep on e_2 in Figure 6(i) can start when both I_1^+ and E_3^+ are done, even if the sweep I_2^+ is not finished yet.

The information dissemination phase is initiated by the local minima and local maxima. A local minimum p initiates a bottom-up sweep with value $E^+(p) = \mathcal{F}(p)$. A local maximum p initiates a top-down sweep with value $I^+(p) = \mathcal{F}(p)$. Along each arc there are at most two sweeps in different directions. In addition, the dissemination of both the minimum and the maximum can be integrated in the same sweep so that the total cost for this phase is roughly $2n$.

For navigation purposes, we will also disseminate information such that a node traveling in a gradient ascend path can easily find ways to each of the two peaks that will split up on the upcoming merge saddle p (so that we do not need to reach the saddle to decide). Specifically, each node on $C_1(p)$ records

its hop count within $C_1(p)$ from the saddle p. This is called its index. For a node w with p as its ascending merge saddle, if w has higher neighbors with ascending pointers to p_1, then w has an ascending pointer to p_1 and its index is the minimum of the indices of those higher neighbors. A node may have ascending pointers to both p_1, p_2, for example, the saddle node p itself and all the nodes with ascending paths to p. Similarly we disseminate the descending pointers along the ascending paths from a split saddle until the next critical contour. This information sweep can be combined with the previous sweep thus it does not incur extra cost.

To summarize, the total communication cost is bounded by the cost of sweeps, and the cost of traversing the saddle contours. In the ideal case when the saddle contours do not severely overlap and the sweeps are stopped in time by the saddle contour traversal, both the sweep cost and the saddle contour traversal cost are a constant factor of the network size. The construction cost in practice is evaluated in simulations.

E. Handing noises

With real sensor data, the signal field may have noises, causing lots of local optima. In practice we will de-noise the signal field by simplifying the contour tree during construction, to improve the construction efficiency. At a saddle node q, we will check the values of the two peaks p_1, p_2. Say $\mathcal{F}(p_1) > \mathcal{F}(p_2)$. If $\mathcal{F}(p_2) - \mathcal{F}(q) < \varepsilon$ and q is at least γ hops away from p_2, with ε and γ as upper bounds on the height and size of a bump to be considered as noises, we consider p_2 insignificant and chop it off. See Figure 7 (i). At the saddle q, q will detect that p_2 is too small, thus it will be chopped at the value of $\mathcal{F}(q)$ and the sweep of p_1 will take over.

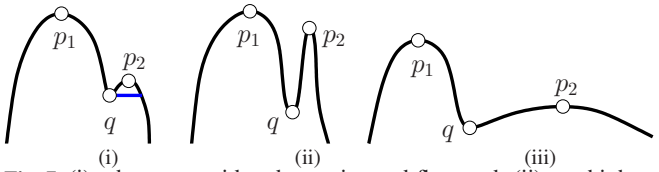

Fig. 7. (i) a bump considered as noise and flattened; (ii) too high to be a noisy bump; (iii) too wide to be a noisy bump;

The above operations effectively 'smooth out' the signal field, guided by local geometric measures. This can substantially simplify the contour tree in a noisy data field. The gradient routing for iso-contour queries will miss at most some small components, whose sizes are controlled by ε and γ.

III. Iso-contour Queries

A. Gradient Descent Routing for Iso-contour Queries

The invariant we constructed so far enables an efficient gradient routing for iso-contour queries with guaranteed success. The gradient descent algorithm uses only the information stored at a node and its immediate neighbors.

Starting at q we first check whether x is beyond the range of the signal field, in which case we do not travel even one step and immediately return $\emptyset$. Effectively, this is by checking whether $I^+(q) < x$ and $E^+(q) < x$, or $I^-(q) > x$ and $E^-(q) > x$. If not, we know that there must be some non-empty iso-contours at level x and we use a greedy gradient algorithm to find them. The main idea is to send the query message along the contour tree, possibly splitting at internal branches, and discover all components of the iso-contour of interest. At the query node q,

- If $I^+(q) \geq x \geq I^-(q)$, then q initiates a query message to follow the gradient uphill.
- If $E^+(q) \geq x \geq E^-(q)$, then q initiates a query message to follow the gradient downhill.

We first explain the ascending query message from q. If a query message hits a node w with isovalue x, it will then start a traversal along the contour component $C(w)$. This is done by the same algorithm as explained earlier. At the same time, we also need to check at w whether it is worth getting even higher up — it is possible that at the interior of $C(w)$ there are still contours of value x. Again this is done by checking a higher neighbor of w, say v, whether $I^+(v) \geq x \geq I^-(v)$.

For an ascending query message at a node w, suppose w stays on an arc with p being an ascending merge saddle. Then we will check for two parents of p, denoted by p_1, p_2, whether we will need to ascend on one peak or both of them. Luckily this information has been disseminated for all the nodes on this arc. Thus w will check the value range within the interior of $C_1(p), C_2(p)$ respectively. If the query value x falls in the range, w will initiate an ascending query message for it. See the red query in Figure 8 as an example of two query messages, one for each peak.

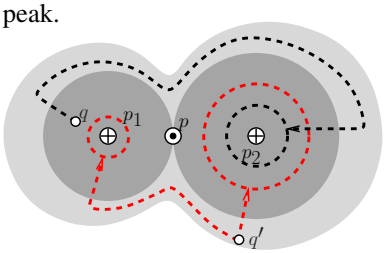

Fig. 8. Examples of two queries.

For an ascending query message towards say peak p_2, if w has ascending pointers to p_2, this query message is simply delivered by gradient ascent routing, as the query from q' shown in Figure 8. If not, then the query message will follow a contour at a random value (below $\mathcal{F}(p)$ and above $\mathcal{F}(w)$) and follow the index-decreasing path, in order to cross the ridge and discover some ascending paths to p_1. The descending query message is delivered in a symmetric manner looking for contour components at x. We may go a random number of hops further after ascending pointers are discovered, in order to avoid always using the nodes on the ridge. To summarize,

- The gradient routing algorithm is completely local and distributed and successfully finds *all* contour components at a given query level.
- Every step of the routing algorithm is *justified*, we send a query message only when we are sure there is something to be found. So no message will end up in vain.
- The routing scheme does not have to go through the saddles or follow critical contours, thus does not overload those nodes.

We note that this iso-contour query is the most basic query of a family of queries on iso-contours. Other iso-contour queries include: reporting the number of contours at value x, in particular, is there a single contour component? Range-limited queries (count/report contours within a value range)? These can be handled with either the iso-contour query as a subroutine, or by using a similar gradient routing algorithm. We omit the details here as the extension is relatively straight-forward.

B. Value restricted routing

The contour tree can be used for *value restricted routing*: given a source s and destination t, find a path $\mathcal{P}$ from s to t such that at every node x on $\mathcal{P}$, $a \leq \mathcal{F}(x) \leq b$, abbreviated as $a \leq \mathcal{F}(\mathcal{P}) \leq b$. Recall that the contour tree can be considered as a retraction $\mathcal{R}$ which maps every point on a contour component to a point on the arc in the contour tree. Thus we have:

Lemma 3.1. *For any path $\mathcal{P}$ between points s and t, the image $\mathcal{R}(\mathcal{P})$ in the tree contains the unique path $\mathcal{P}'$ in the tree between $\mathcal{R}(s)$ and $\mathcal{R}(t)$.*

Theorem 3.2. *A value restricted path exists in the network if and only if a value restricted path exists in the contour tree.*

Proof: In the following, we assume $\mathcal{F}(s), \mathcal{F}(t) \in [a, b]$, since otherwise the request is clearly invalid. First, a path $\mathcal{P}'$ in the contour tree implies a path in the network. Since the tree is a retract of the domain, a path $\mathcal{P}'$ in the tree is also a path in the network. Also it is possible to traverse from s to $\mathcal{R}(s)$ and from t to $\mathcal{R}(t)$ along $C(s)$ and $C(t)$ respectively. Appending these to $\mathcal{P}'$ gives the required path.

For the other direction, a path $\mathcal{P}$ in the network implies a path in the tree. Let $\mathcal{P}'$ be the unique path between $\mathcal{R}(s)$ and $\mathcal{R}(t)$. Then by lemma 3.1, $\mathcal{P}' \subseteq \mathcal{R}(\mathcal{P})$. Since $\forall p, \mathcal{F}(\mathcal{R}(p)) = \mathcal{F}(p)$, we have $\max(P') \leq \max(\mathcal{R}(\mathcal{P})) = \max(P)$ and $\min(P') \geq \min(\mathcal{R}(\mathcal{P})) = \min(P)$. ■

The results have a number of implications. The contour components on path $\mathcal{P}'$ on the contour tree are ones that any path from s to t in the network must intersect. In moving from s to t along any path, we can keep record of number of times each component appears, or simply push and pop them on a stack. The ones remaining in the stack at the end constitute the path $\mathcal{P}'$. Thus, a value restricted path can be obtained by deforming any path connecting source and destination.

To answer the value restricted routing problem in a sensor network, if we disseminate the entire contour tree to every node, the a route in the network can be found in a greedy manner by following the contour components connecting the source and destination. If we do not store the entire tree at every sensor, we can develop a node labeling scheme, such that by using the labels of source and destination we can tell whether a path exists or not.

Given a contour tree with m vertices, we first do a balanced decomposition of the tree. For any tree there is a cut node, whose removal will leave subtrees each of size no more than 2/3 of the total vertices. Repeatedly partition each subtree to get a balanced decomposition of depth $\log m$. The label of a node u will be the concatenation of the IDs of all the cut nodes along the path from u to the root of the decomposition tree, as well as the max/min value of the paths from u to these cut nodes. Thus the size of the label is $O(\log m)$. For any two nodes s and t, by their labels, we can immediately find the lowest level cut node w shared by them. The path between them will necessarily go through w. Thus by taking the union of the range of the paths from s, t to w, we get the value range of the path connecting s, t in the contour tree.

With the labels pre-computed, each node p in the contour component will store the labels of its ascending and descending critical points. Thus one can use the labels of source and destinations to answer the value restricted routing requests.

IV. Simulations

We implemented the algorithm for constructing contour tree and for answering iso-contour queries with gradient routing. Our simulations do not take into consideration many important networking details, e.g., packet loss, delay and channel contention. This set of simulations is a proof of concept and aims to verify the correctness of the algorithm and evaluate the feasibility of the approach on the algorithmic level.

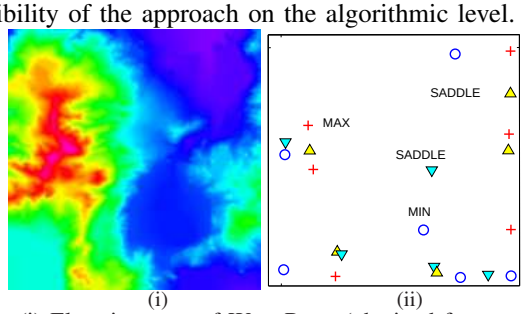

Fig. 9. (i) Elevation map of West Reno (obtained from usgs.gov). (ii) The critical points discovered by our contour tree algorithm with a 2500 node sampling.

Unless specified otherwise, the simulation setup consists of 1600 nodes, deployed in a 16 units by 16 units square region with unit disk graph as the communication model. Nodes are deployed in a perturbed-grid distribution, where each node is assigned a random position within its grid square. The average number of neighbors per node is about 21. The sensors sample from a continuous signal field shown as in Figure 10 (i).

A. Preprocessing cost for contour tree construction

We first evaluate the cost of contour tree construction. We vary the number of nodes with the same signal field and count the total number of messages, assuming a broadcast medium. In our implementation, a random node on the sweep frontier is selected to become swept. The number of messages grows linearly in the number of nodes as shown in Figure 10 (ii). The constant factor is about $6 \sim 7$.

B. Cost of iso-contour queries

We compare the cost of gradient routing versus a global solution of using the minimum spanning tree to connect the query node q and all the nodes on the iso-contour at value x, which is a 2-approximation of the minimum Steiner tree, the optimal (minimum cost) solution if the full knowledge of the signal field is available. We take 300 random queries with

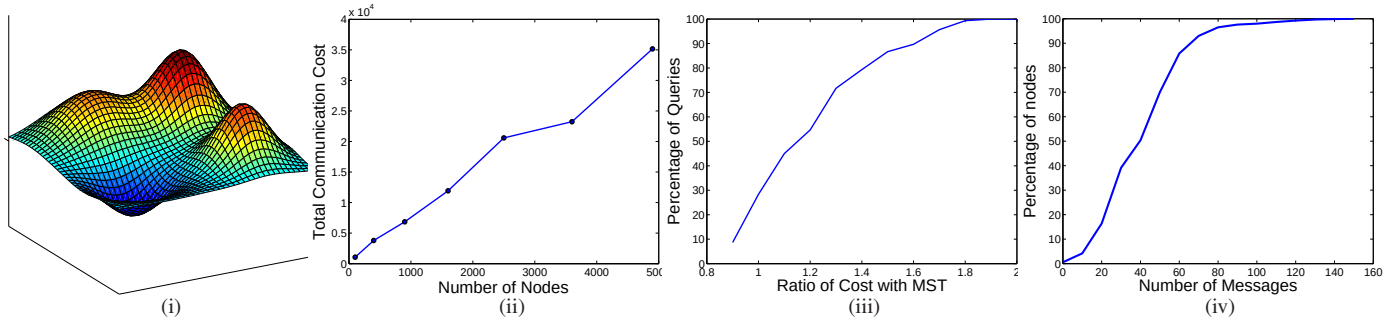

Fig. 10. (i) The continuous signal field sampled by the distributed sensors; (ii) The message complexity of contour tree construction; (iii) The CDF of the ratio of our query cost v.s. the cost of MST; (iii) The CDF of the node load distribution.

q randomly selected within the field of deployment and the query value x randomly chosen between the global minimum and global maximum values. For each query, we take the ratio of our query cost versus the cost of MST (both in terms of number of hops). We calculated the cumulative distribution, i.e., the percentage of queries for which the ratio is below x, in Figure 10 (iii). Roughly all cases have a ratio below 2 and 80% of the queries have a ratio below 1.4.

C. Load balancing

In the same setup as the previous section, we plot the load on every node involved in the query procedure. A node is involved if it is on the routing path or is on the iso-contour to be queried. The maximum message load on any node is 148, the average message load is about 48.7. The load distribution is shown in Figure 10 (iv), in which 90% of the nodes have a message load of below 70.

V. Conclusion and Future Work

In this paper we presented the distributed construction of a contour tree and its application in iso-contour queries by gradient routing with guaranteed delivery. Our future work is to update and maintain the contour tree for a time-varying signal field [21].

Acknowledgement. We would like to thank the support from Stony Brook CEWIT center. X. Zhu, R. Sarkar and J. Gao would like to acknowledge support though NSF CAREER Award CNS-0643687. J. Mitchell would like to acknowledge support by NSF CCF-0431030, CCF-0528209, CCF-0729019 and by Metron Aviation, under subcontracts from NASA Ames. L. Guibas would like to acknowledge NSF grants CNS-0626151, CCF-0634803, and Army grant W911NF-07-2-0027-1.

References

[1] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. B. Srivastava, "Coverage problems in wireless ad-hoc sensor networks," in *Proc. of INFOCOM 2001*, vol. 3, 2001, pp. 1380–1387.

[2] S. Kumar, T. H. Lai, and J. Balogh, "On k-coverage in a mostly sleeping sensor network," in *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, 2004, pp. 144–158.

[3] X. Bai, S. Kuma, D. Xua, Z. Yun, and T. H. La, "Deploying wireless sensors to achieve both coverage and connectivity," in *MobiHoc '06: Proceedings of the seventh ACM international symposium on Mobile ad hoc networking and computing*, 2006, pp. 131–142.

[4] H. Zhang and J. C. Hou, "Maintaining sensing coverage and connectivity in large sensor networks," *Wireless Ad Hoc and Sensor Networks: An International Journal*, vol. 1, no. 1-2, pp. 89–123, January 2005.

[5] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek, "Beyond average: Toward sophisticated sensing with queries," in *Proc. Information Processing in Sensor Networks (IPSN)*, April 2003, pp. 63–79.

[6] X. Meng, L. Li, T. Nandagopal, and S. Lu, "Contour maps: Monitoring and diagnosis in sensor networks," *Computer Networks*.

[7] S. Gandhi, J. Hershberger, and S. Suri, "Approximate isocontours and spatial summaries for sensor networks," in *IPSN'07: Proceedings of the 6th international conference on Information processing in sensor networks*, 2007, pp. 400–409.

[8] M. Chu, H. Haussecker, and F. Zhao, "Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks," *Int'l J. High Performance Computing Applications*, vol. 16, no. 3, pp. 90–110, 2002.

[9] J. Liu, F. Zhao, and D. Petrovic, "Information-directed routing in ad hoc sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 4, pp. 851–861, April 2005.

[10] J. Faruque and A. Helmy, "RUGGED: RoUting on finGerprint Gradients in sEnsor Networks," in *IEEE Int'l Conf. on Pervasive Services (ICPS)*, July 2004.

[11] J. Faruque, K. Psounis, and A. Helmy, "Analysis of gradient-based routing protocols in sensor networks," in *IEEE/ACM Int'l Conference on Distributed Computing in Sensor Systems (DCOSS)*, June 2005.

[12] F. Ye, G. Zhong, S. Lu, and L. Zhang, "GRAdient broadcast: A robust data delivery protocol for large scale sensor networks," *ACM Wireless Networks (WINET)*, vol. 11, no. 2, March 2005.

[13] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore, "Contour trees and small seed sets for isosurface traversal," in *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, 1997, pp. 212–220.

[14] J. W. Milnor, *Morse Theory*. Princeton, NJ: Princeton University Press, 1963.

[15] M. de Berg and M. van Kreveld, "Trekking in the alps without freezing or getting tired," *Algorithmica*, vol. 18, pp. 306–323, 1997.

[16] H. Carr, J. Snoeyink, and U. Axen, "Computing contour trees in all dimensions," in *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms (SODA)*, 2000, pp. 918–926.

[17] S. P. Tarasov and M. N. Vyalyi, "Construction of contour trees in 3D in $O(n \log n)$ steps," in *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, 1998, pp. 68–75.

[18] P. Skraba, Q. Fang, A. Nguyen, and L. Guibas, "Sweeps over wireless sensor networks," in *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, 2006, pp. 143–151.

[19] Y.-J. Chiang, T. Lenz, X. Lu, and G. Rote, "Simple and optimal output-sensitive construction of contour trees using monotone paths," *Comput. Geom. Theory Appl.*, vol. 30, no. 2, pp. 165–195, 2005.

[20] H. Carr, J. Snoeyink, and M. van de Panne, "Simplifying flexible isosurfaces using local geometric measures," in *VIS '04: Proceedings of the conference on Visualization '04*, 2004, pp. 497–504.

[21] H. Edelsbrunner, J. Harer, A. Mascarenhas, and V. Pascucci, "Time-varying reeb graphs for continuous space-time data," in *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, 2004, pp. 366–372.