A coordinate-free, decentralized algorithm for monitoring events occurring to peaks in a dynamic scalar field

Myeong-Hun Jeong [#1], Matt Duckham [#2]

[#] *Infrastructure Engineering, University of Melbourne, VIC, Australia*

[1] mh.jeong@student.unimelb.edu.au

[2] mduckham@unimelb.edu.au

***Abstract*—This paper proposes a decentralized and coordinate-free algorithm to monitor spatial events in a dynamic scalar field. The events that are the focus of this paper are the appearance, disappearance, and movement of "peaks" (local maxima). However, ongoing work is extending the approach to monitor the full set of critical points (peaks, pits, and passes). Our approach is based on the gradient flow between immediate neighbors in a geosensor network. Experimental investigations demonstrate that our algorithm is scalable, with $O(n)$ overall communication complexity (where n is the number of nodes in the geosensor network), and even load distribution. Further, our algorithm can improve the accuracy of the identification of peaks in a limited spatial granularity sensor network when compared with an alternative approach that does not account for granularity issues. The results of this research have wide application to decentralized monitoring of dynamic fields, such as changing temperature, pollution levels, or gas concentration.**

I. Introduction

This paper presents an algorithm for monitoring spatial events in a dynamic scalar field, specifically events occurring to "peaks" (local maxima). This algorithm has application to wireless sensor network monitoring of the physical world, in particular geosensor networks monitoring phenomena in geographic environments [1]. The problem of monitoring qualitative events in a dynamic scalar field is challenging both because of the finite granularity of geosensor networks as well as their unique resource constraints. Limited energy resources make it advantageous to investigate algorithms that are able to operate in-network without centralized control and without precise coordinate positioning information (i.e., GPS).

This paper addresses these challenges by proposing a decentralized and coordinate-free algorithm for accurately inferring events occurring to peaks in a dynamic field, specifically appearance, disappearance, and movement events. The approach is based on monitoring the gradient vectors between immediate network neighbors acknowledging the limited spatial granularity of geosensor networks.

The following section gives a brief overview of the recent history of the identification of critical points in a geosensor network. Section III specifies a decentralized algorithm for monitoring peaks events in a dynamic scalar field. Section IV presents an experimental evaluation in terms of communication complexity and accuracy. Finally, the paper concludes with a discussion of future work in Section V.

II. Background

There exist two fundamentally different perspectives of monitoring dynamic geographic phenomena: *histories* and *chronicles*. Histories are sequences of states of the world over time; chronicles are sequences of occurrences or "happenings" over time [2], [3]. For example, [4] provides a framework for the detection of spatial events (e.g., merging, splitting, or movement) in dynamic regions. [5] determines the changes in topological relations between evolving regions. These examples therefore focus on chronicles, sequences of qualitative events. By contrast, [6] adopt a history view: monitoring over time the existence (i.e., state) of "flocks" among mobile geosensor nodes. In this paper we are concerned with the former perspective (chronicle): that of identifying the sequence of events that occur to a peak over time.

Monitoring events in dynamic fields based on critical points (i.e., peaks, pits, and passes) has already received some attention in the literature. In particular, [7] analyzes the changes in retail clustering in a city (e.g., appearance, disappearance, movement) based on a *surface network* (i.e., critical points and their connecting ridges and channels). In terms of geosensor networks, there are a few decentralized algorithms capable of identifying critical points in a static scalar field (e.g., [8], [9]). However, to the best of our knowledge no study has addressed the monitoring of chronicles for critical points in a dynamic scalar field.

This study, therefore, seeks to address this gap. The following section introduces the basic definitions for critical points in a limited granularity geosensor networks and an algorithm, suitable for deployment in a resource-constrained geosensor network, for identifying events to critical points based on an analysis of the gradient flow of the dynamic field.

III. Algorithm

This section outlines the computational environment assumed for our algorithm, defines the critical points of interest, and presents our algorithm for monitoring events occurring to peaks in a dynamic scalar field.

A. Preliminaries

Following previous work [10], [11], a geosensor networks can be modeled as follows:

- a geosensor network, undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of communication links between neighboring nodes;
- the neighborhood of a node, $nbr : V \rightarrow 2^V$, where $nbr(v) = \{v' \in E | (v, v') \in E\}$;
- a sensor function, $s : V \rightarrow \mathbb{R}$; and
- an identifier function for nodes, $id : V \rightarrow \mathbb{N}$.

We use the over-dot notation in the algorithm to emphasize a node's local knowledge of a global function (e.g., $\mathring{s}$ is equivalent to $s(v)$ where node v is clear from the context). We also adopt the common assumption in decentralized algorithm design of reliable communication (i.e., every message will be delivered within a finite amount of time) [12]. Further, we assume each node senses a different scalar field (z-value) from its neighbors. This assumption is the discrete analog of the assumption of smooth surfaces with non-degenerate critical points commonly made in connection with continuous fields in Morse theory [13].

The algorithm specification in the following subsection follows the style in [12], [11]. In summary, each node must be in exactly one from a set of states (e.g., IDLE or PEAK in an Algorithm 1). In any particular state, a node reacts to system events (e.g., *Receiving* a message in an Algorithm 1) by performing a finite sequence of atomic operations, termed an action. In this way, algorithm specification essentially involves the specification of states, system events, and a unique (possibly empty) action for each state/system event pair.

B. The identification of peaks in a static field

This paper identifies peaks in a scalar field based on gradient vectors deduced from immediate (one-hop) neighbors' communication. The identification of critical points is strikingly reminiscent of Morse theory [13]. Morse theory is commonly used to capture the topological structure of a continuous function (surface) based on critical points. Discrete analogs of Morse theory have been proposed. For example, [14] proposes a discrete Morse theory based on a simplicial complexes, identifying critical points by tracing a gradient vector. Unfortunately, this approach is not directly applicable to the context of a geosensor network because of the irregular spatial structure of geosensor networks, and its resulting network granularity effects (network holes and nodes with limited connectivity).

This finding therefore leads to investigating different approaches for identifying critical points. Although we assume no coordinate positions for nodes, each node can still construct qualitative spatial knowledge of the gradient field by comparing its own value with that of its one-hop neighbors. A node's ascent (descent) vector can be defined as pointing to the neighboring node that has the highest (lowest) value of all its neighbors. For example, a node v's ascent vector $av(v) = (v, v')$ where v' is the neighbor such that $s(v') = \max_{v_i \in nbr(v)}(s(v_i))$ and $s(v') > s(v)$. Descent vectors can be defined in a similar manner.

By routing a message along ascent (or descent) vectors, any node identify it's peak (or pit). Limited network granularity, however, may mean that some of these peaks (and pits) do not reflect broader-scale peaks (pits) in the surface. Such a "false" peak is shown in Fig. 1, a common occurrence in limited spatial granularity geosensor networks. It is possible to define a *true* peak as a peak where all its neighbors' ascent vectors point to it. Fig. 1 also highlights a "true" peak under this definition.

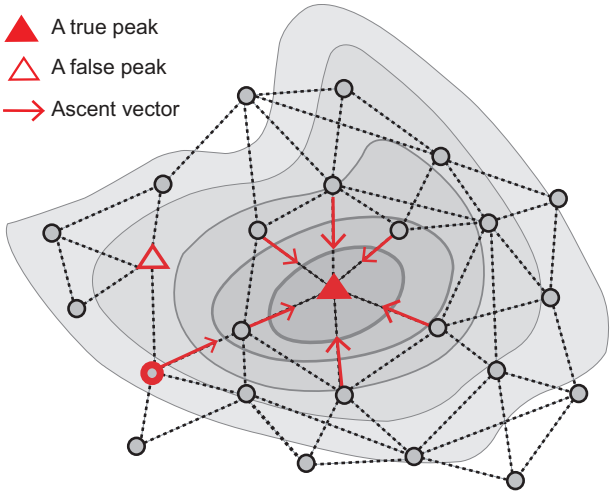


Fig. 1. The identification of a "true" peak, where its all neighbors' ascent vector point to it. Conversely, a "false" peak can be detected when at least one of its neighbors' ascent vectors (thick-boundary circle) points to another node.

Based on these definitions, our approach can identify"true" peaks in a scalar field using the following outline steps:

1) Each node broadcasts a `ping` message containing its sensed value. Nodes with no ascent vector (i.e., where no neighbors have a higher sensed value) are potential peaks.
2) Potential peaks can then poll their two-hop neighbors' sensed value, from which they can deduce whether they are a "true" peak (i.e., where all two-hop neighbors' sensed values are less than their own sensed value).
3) A "true" peak can then initiate a top-down invitation message (`invt`) to its neighbors. Nodes receiving an invitation from their maximum ascent vector neighbors store information about the identifier of their "true" peak. "False" peaks can also be included in this process by adjusting their ascent vector in the "false" peak identification stage in 2 above.

The result of this initialization process is that each node becomes aware of the identifier of its unique "true" peak: the process essentially results in clustering of all nodes according

to catchment areas. Our dynamic algorithm for monitoring events to peaks operates by monitoring changes to nodes' associated "true" peak.

C. Monitoring chronicles for peaks in a dynamic field

Following the identification of "true" peaks in an initial scalar field, any events that occur to peaks (appearance, disappearance, and movement) can be monitored as a consequence of inferring previous/current ascent vectors between neighbors in a geosensor network.

First, Algorithm 1 presents a mechanism to detect these events. In summary:

- Every node sensing a new peak value, compared with previously, broadcasts this data to its neighbors (see Algorithm 1, line 7) as an `upd8` message.
- When a node receives an `upd8` message, it stores this information (see Algorithm 1, line 10).
- When changes are detected by a node (either its sensed value changes or an update message is received from neighbors), this node initiates a timer. The node then waits for further update messages from neighbors (see Algorithm 1, line 6 and 12) before committing any changes, because happenings (events) are usually correlated such that a brief wait can substantially reduce redundant communications.

By maintaining this change information, nodes can locally deduce when certain events (peak appearance, disappearance, or movement) have occurred.

For example, Algorithm 2 is able to monitor the appearance of peaks by tracing previous/current ascent vectors. For example:

- If a node in a IDLE state has highest value of all its neighbors, this node sends a `trpk` message to its past ascent vector (see Algorithm 1, line 19). This message is then relayed by each node based on past ascent vectors (see Algorithm 2, line 6).
- If a `trpk` message reaches a peak with the initiating node's original peak id, it can be deduced that a peak appearance has occurred (see Algorithm 2, line 24);
- If a `trpk` message reaches a node that previously a peak (now just a regular IDLE node) then there are two possibilities: a peak has appeared or one has moved.
- To determine which, the ex-peak sends a `trav` message to its current ascent vector (see Algorithm 2, line 4).
- If this `trav` message encounters another node with different peak id, it can be deduced that the a new peak has appeared (see Algorithm 2, line 10).
- Lastly, a node receiving a confirmation `rtpk` message will check whether it is a "true" peak or a "false" peak. If a "true" peak, this node disseminates a `swpk` message to swap relevant nodes' previous peak id with the new peak id (see Algorithm 2, line 21).

Monitoring changes for peak movement and peak disappearance follows a similar pattern. For example, if a node in a PEAK state makes a transition to a node in an IDLE state, this node forwards a message to infer peak disappearance or movement by tracing ascent vectors. If this message meets a peak with different a peak id, the old peak node (now in a IDLE state) can be deduced to have disappeared. Conversely, if this message meets a peak with the same peak id, this identification indicates that the peak moved. These explanations have been eliminated from the algorithms mentioned above for conciseness.

Algorithm 1 The detection of peak changes in a dynamic field

1: Restrictions: Geosensor network, $G = (V, E)$; sensor function $s : V \to \mathbb{R}$; communication neighborhood $nbr : V \to 2^V$; identifier function $id : V \to \mathbb{N}$; reliable communication.
2: State Trans. Sys.: $\langle \{\text{INIT}, \text{LSTN}, \text{IDLE}, \text{PEAK}\}, \{(\text{IDLE}, \text{PEAK})\}\rangle$
3: Local variables: the last sensing value, s_l, initialized empty; list of upstream neighbors, N_u, initialized empty; an ascent vector, *av*, initialized empty; the last ascent vector, av_l, initialized empty; a true peak id, *pkid*, initialized empty; a last peak id, $pkid_l$, initialized empty; a timer, *timer*, initialized zero

IDLE, PEAK
4: *When* $\mathring{s}$ changes
5: **set** $s_l := s$ {Save the last sensed value }
6: **set** $timer := now$ {Start timer }
7: **broadcast** (`upd8`, $\mathring{id}$, $\mathring{s}$)
8: *Receiving* (`upd8`, i, s)
9: **if** $s > \mathring{s}$ **then** {Check upstream neighbors}
10: **update** N_u
11: **if** *timer* $= 0$ **then**
12: **set** $timer :=$ *now* {Start timer }

IDLE
13: *When timer* elapsed {Wait for predefined threshold }
14: **set** $av_l :=$ *av* {Save the last ascent vector }
15: **if** $|N_u| > 0$ **then** {Update ascent vector}
16: **set** $av := max(N_u)$
17: **if** $|N_u| = 0$ **then**
18: **if** $|av| > 0$ **then**
19: **send** (`trpk`, $\mathring{id}$, *pkid*, $\mathring{id}$) to av_l {Check for event }
20: **set** *timer* $:= 0$ {Initialize timer }

PEAK
21: *When timer* elapsed {Wait for predefined threshold }
22: **if** $|N_u| > 0$ **then** {Update ascent vector}
23: **set** $av := \max(N_u)$
24: **send** (`wipk`, $\mathring{id}$, $\mathring{id}$, *pkid*) to *av* {Check for event }
25: **set** $av_l := \emptyset$ {Save the last ascent vector }
26: **set** $pkid_l :=$ *pkid* {Save the last peak id }
27: **set** *timer* $:= 0$ {Initialize timer }

In analyzing the efficiency of the algorithm, the messages sent fall into two distinct stages: initialization and monitoring of ongoing changes. During initialization, each node must send at least two messages (i.e., `ping`, `invt`); a few nodes require one additional message for identifying "true" peaks. During ongoing monitoring, a node detecting a change sends one message (`upd8`), initiating a (linear) chain of update messages. Further, some nodes send other messages (i.e., `trpk`, `trav`, `rtpk`, `swpk`) to infer changes for peaks.

This algorithm therefore is expected to have communication complexity $O(n+m)$ per "round", where m is the number of messages for identifying "true" peaks during initialization or inferring events occurring to peaks during ongoing monitoring.

Algorithm 2 Monitoring peak appearance

1: Fragment extend : Algorithm 1

IDLE

2: *Receiving* (trpk, p_s, p_p, i)
3: **if** $pkid_l = p_p$ **then** {Previous peak node}
4: **send** (trav, p_s, $\mathring{id}$, $pkid_l$) to av {Forward a message to current ascent vector }
5: **else**
6: **send** (trpk, p_s, p_p, $\mathring{id}$) to av_l {Forward message to the previous ascent vector }
7: *Receiving* (trav, p_s, i, $pkid_l$)
8: **if** $|N_u| = 0$ **then**
9: **if** $\mathring{id} \neq p_s$ **then**
10: Peak appearance (cf. Algorithm 1, line 19)
11: **else**
12: **send** (trav, p_s, $\mathring{id}$, $pkid_l$) to av {Forward message to current ascent vector }
13: *Receiving* (rtpk, p_s, e, p_p)
14: **if** $\mathring{id} = p_s$ **then**
15: **if** This node is not a "false" peak **then**
16: **if** $e =$ "appearance" **then**
17: **set** $av := -1$ {No ascent vector }
18: **set** $pkid_l := pkid$ {Save the previous peak id }
19: **set** $pkid := \mathring{id}$ {Set current peak id }
20: **become** PEAK
21: **broadcast** (swpk, "appearance", $\mathring{id}$, $pkid$, $pkid_l$){Swap peak id }

PEAK

22: *Receiving* (trpk, p_s, p_p, i)
23: **if** $pkid = p_p$ **then**
24: **send** (rtpk, p_s, "appearance", $pkid$) to i {Forward appearance message to a source node }

Further, if this algorithm runs for $t = |T|$ time steps, the communication complexity is $O(t(n+m))$. As a result, the overall communication complexity of the algorithm is linear in the number of nodes, $O(n)$. This expectation was tested through empirical simulation in the following section.

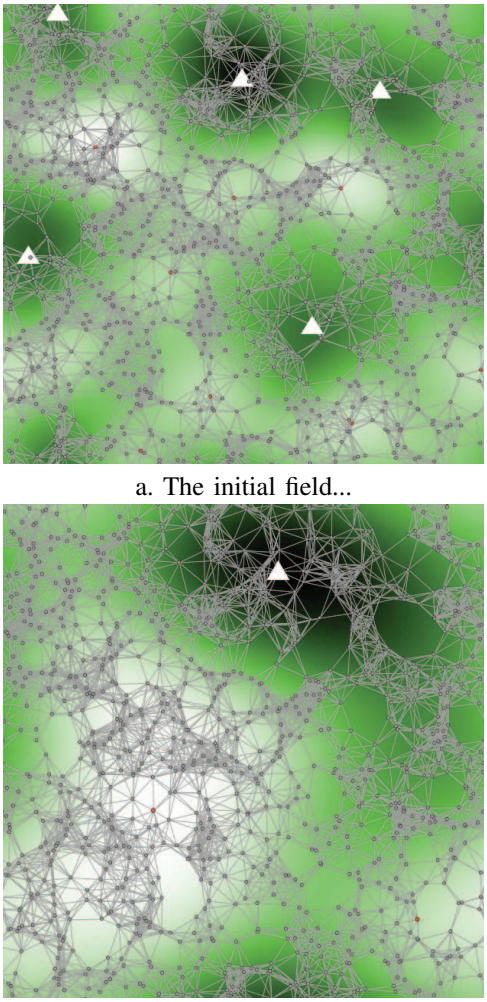
IV. Experiments

The performance of the presented algorithm was evaluated in terms of overall scalability, load balance, and accuracy. The algorithm was implemented in an agent-based simulation system, Netlogo [15].

For each simulation run, nodes were randomly distributed and connected by the unit distance graph (UDG). To guarantee comparability, a constant level of network connectivity was maintained across different network sizes tested. A randomized dynamic scalar field was also simulated in Netlogo. Fig. 2 shows an example of monitored peaks based on randomly generated scalar field and its evolution (i.e., several peaks disappeared after short period of evolution of the field).

A. Overall scalability

The first set of experiments investigated the efficiency of the algorithm in terms of its overall scalability. The algorithm was simulated on 10 randomized networks across a range of five different sizes, from 1000 nodes to 16000 nodes. Further, to each experimental run was replicated 11 times. Thus, the



a. The initial field...

b. ... and its evolution after a short period of time

Fig. 2. Example of dynamic field evolution monitored by randomly distributed 1000 nodes network (a triangle is a detected "true" peak).

number of total simulations is $5 \times 10 \times 11 = 550$. In the one-off initialization step, the n initialization messages (i.e., ping) are ignored in the following analysis.

Fig. 3 shows that the total number messages generated by the algorithm, averaged over 10 randomized networks, where the entire field changes in each evolution step (worst case for our algorithm, requiring every node in the network to respond to a change). A linear regression analysis achieved good fit (R^2 value 0.9962), indicating as expected $O(n)$ overall communication complexity, with about 12 messages generated per timestep. Therefore, this result is in accordance with our expectation.

B. Load balance

In a decentralized sensor network system, uneven load balance can cause some nodes to rapidly deplete their battery power. This phenomenon can result in holes in network coverage. Fig. 4 shows the load balance for our algorithm. The figure shows a frequency histogram of the number of nodes sending a particular number of messages over 11 timesteps.

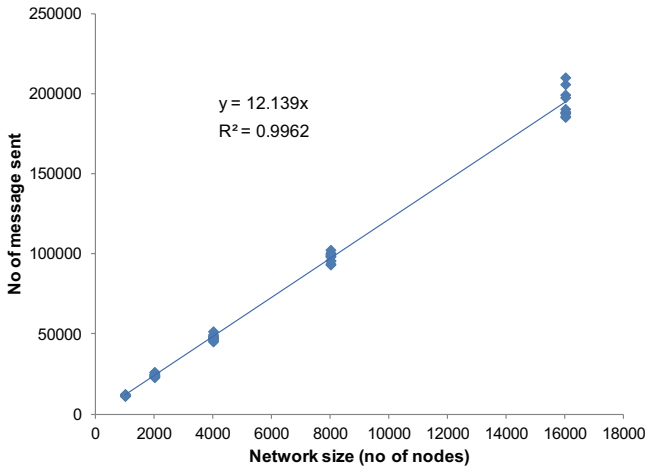


Fig. 3. Overall scalability for monitoring changes for peaks (averaged over 10 randomized networks and 11 consecutive evolution)

It is evident that each node transmitted at least 11 messages (one for the initial field and ten for the evolved field). However, 96% of nodes transmitted fewer than 15 messages. This result provides evidence of even load balance, potentially avoiding rapid depletion of nodes' resources.

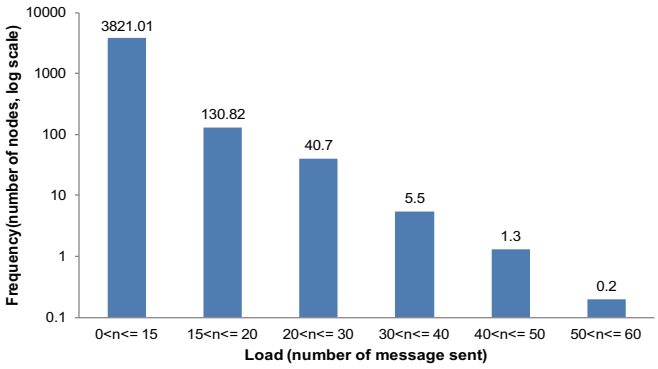


Fig. 4. Load balance (averaged over 10 networks of 4000 nodes and 11 consecutive timesteps)

Although the analysis of communication complexity is worthwhile activity, this analysis cannot guarantee the correctness of an algorithm. The following section investigates the accuracy of our algorithm.

C. Accuracy

The accuracy of our algorithm was analyzed in terms of precision, recall, and F1-score for the identification of peaks. We also compared our algorithm with an alternative algorithm that does not account for the limited spatial granularity of the geosensor network. While our algorithm identifies a peak by investigating its highest value among its neighbors' and neighbors' ascent vectors, the counterpart algorithm only regards a peak as a node that has highest value among its neighbors. Thus the comparison between the two algorithms tests the benefit of our algorithm's refined definition of "true" peaks. In comparing the accuracy of these algorithms, the F1-score provides the best indicator, as it is a combination of precision and recall.

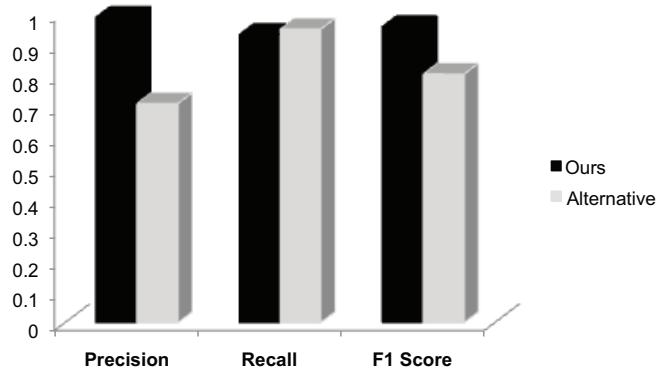


Fig. 5. Precision, recall, and F1-score for the identification of peaks on 11 consecutive evolution over 4000 nodes

Fig. 5 shows the algorithm achieved high accuracy. Further the results shown in Fig. 5 indicated that there was a significant improvement in our algorithm, when compared with the naive peak definition, indicated by the improved F1-score of our algorithm.

V. Discussion and Conclusions

This paper has presented a decentralized and coordinate-free algorithm to monitor chronicles of events in a dynamic field, specifically appearance, disappearance, and movement of peaks. The approach accounts for the inherently limited spatial granularity of any geosensor networks by using the gradient vectors between immediate one-hop neighbors.

Empirical evaluations have demonstrated that the algorithm is scalable, with overall $O(n)$ communication complexity. Our simulations adopted the worst case (in terms of efficiency), where the dynamic field changed its value at each timestep, so that each node must update and transmit its value to neighbors most frequently. It is reasonable, therefore, to suppose that in practical application, where sensed values are strongly temporally-autocorrelated, the communication overheads would be further decreased.

In terms of load balance, the majority of nodes sent only one message per timestep. In the worst case nodes sent approximately five messages per evolution timestep. Thus, the algorithm achieved relatively even load balance. Here, we expect our algorithm to again be aided by the inherent spatiotemporal autocorrelation of dynamic fields. In cases of a highly dynamic field, although this may lead to the highest overall numbers of messages, because the location critical points must be changing it is expected that the nodes bearing the higher load will necessarily change over time. By contrast, for the locations of critical points to remain static, relatively few changes to the underlying dynamic field are

typically required, so again we expect relatively few messages generated, and so an even load balance.

Turning to the accuracy of the algorithm, our algorithm performed at high levels of accuracy and better than an alternative that does not distinguish between "true" and "false" peaks. With the improved accuracy, it is possible better to infer events for critical points in a dynamic field. Conversely, the misidentification of peaks is certain to lead to difficulty correctly identifying events to critical points.

Finally, the algorithms in this paper is specifically designed to monitor chronicles of events for peaks in a dynamic field. One assumption is that when a node's sensed value changes, it will wait a short period for notification of neighbors' value changes. To relax this assumption, future will address the design of a robust algorithm for a rapidly evolving field (e.g., another event occurs during this short wait time, potentially ignored by our algorithm). Further, ongoing work is investigating other types of spatial events, such as merges, and particularly the challenging issues of identifying events for passes in surface networks.

ACKNOWLEDGMENTS

Matt Duckham's work is supported under the Australian Research Council (ARC) Future Fellowship funding scheme (grant number FT0990531).

REFERENCES

[1] S. Nittel, "A survey of geosensor networks: Advances in dynamic environmental monitoring," *Sensors*, vol. 9, no. 7, pp. 5664–5678, 2009.

[2] A. Galton, "Fields and objects in space, time, and space-time," *Spatial Cognition & Computation*, vol. 4, no. 1, pp. 39–68, 2004.

[3] M. Worboys, "Event-oriented approaches to geographic phenomena," *International Journal of Geographical Information Science*, vol. 19, no. 1, pp. 1–28, 2005.

[4] M. F. Worboys and M. Duckham, "Monitoring qualitative spatiotemporal change for geosensor networks," *International Journal of Geographical Information Science*, vol. 20, no. 10, pp. 1087–1108, 2006.

[5] L.-J. Guan and M. Duckham, "Decentralized reasoning about gradual changes of topological relationships between continuously evolving regions," in *Conference on Spatial Information Theory (COSIT '11)*, ser. Lecture Notes in Computer Science, M. J. Egenhofer, N. A. Giudice, R. Moratz, and M. F. Worboys, Eds., vol. 6899. Berlin: Springer, 2011, pp. 126–147.

[6] P. Laube, M. Duckham, and M. Palaniswami, "Deferred decentralized movement pattern mining for geosensor networks," *International Journal of Geographical Information Science*, vol. 25, no. 2, pp. 273–292, Feb. 2011.

[7] Y. Sadahiro, "Analysis of surface changes using primitive events," *International Journal of Geographical Information Science*, vol. 15, no. 6, pp. 523–538, September 2001.

[8] R. Sarkar, X. Zhu, J. Gao, L. Guibas, and J. Mitchell, "Iso-contour queries and gradient descent with guaranteed delivery in sensor networks," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, Phoenix, AZ, April 2008, pp. 960–967.

[9] X. Zhu, R. Sarkar, and J. Gao, "Topological data processing for distributed sensor networks with morse-smale decomposition," in *INFOCOM 2009, IEEE*, april 2009, pp. 2911 –2915.

[10] M. Duckham, D. Nussbaum, J.-R. Sack, and N. Santoro, "Efficient, decentralized computation of the topology of spatial regions," *IEEE Transactions on Computers*, vol. 60, no. 8, pp. 1100–1113, 2011.

[11] M. Duckham, *Decentralized Spatial Computing: Foundations of Geosensor Networks*. Springer, Berlin, 2013.

[12] N. Santoro, *Design and Analysis of Distributed Algorithms*. New Jersey: John Wiley & Sons, Inc., 2007.

[13] J. Milnor, *Morse Theory*. Princeton University Press, 1969.

[14] R. Forman, "A user's guide to discrete morse theory," *Sém. Lothar. Combin*, vol. 48, p. B48c, 2002.

[15] U. Wilensky, *NetLogo. http://ccl.northwestern.edu/netlogo/*. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL., 1999.