Energy-Efficient Continuous Isoline Queries in Sensor Networks[*]

Adam Silberstein Rebecca Braynard Jun Yang
Department of Computer Science, Duke University, Durham, NC 27708, USA
{adam,rebecca,junyang}@cs.duke.edu

Abstract

Environmental monitoring is a promising application for sensor networks. Many scenarios produce geographically correlated readings, making them visually interesting and good targets for the isoline query. This query depicts boundaries showing how values change in the network. Temporal and spatial suppression provide opportunities for reducing the cost of maintaining the query result. We combine both techniques for maximal benefit by monitoring node and edge constraints. A monitored node triggers a report if its value changes. A monitored edge triggers a report if the difference between its nodes' values changes. The root collects reports and derives all node values, from which the query result is generated. We fully exploit this strategy in our algorithm, CONCH*, which maintains the set of node and edge constraints that minimizes message traffic.*

1 Introduction

Conserving battery power is paramount to prolonging the life of a wireless sensor network. Network nodes communicate between each other and to the base station, or root, via radio. Radio dominates energy consumption. Query algorithms must be sensitive to this cost, and attempt to minimize communication during execution. A primary strategy for minimizing communication is *suppression*, which, in general, keeps nodes from transmitting data unless monitored conditions change. This strategy is effective in many scenarios, such as environmental, where conditions change slowly, or industrial, where change reflects abnormal behavior.

We now examine the *isoline query*, which maintains a map of the sensor network with lines drawn bounding areas of nodes with identical or similar measured values, such as temperature. If two nodes are close enough in value, we say they are in the same *tier*. We increase query resolution by decreasing tier width. The isoline query can be used to visualize the moving trends affecting the monitored area. It is particularly useful when data is geographically correlated, and nearby nodes have a high likelihood of being in the same tier. The resulting map is then likely to be visually interesting with longer isolines, rather than disjoint points or short line segments. Fundamentally, the query maintains an estimate of each node value.

Temporal Suppression The most basic version of suppression is temporal, where each node only transmits its monitored value to the root if its tier has changed since its last transmission. The root, in turn, fills in all unreported values with their previous ones. If most nodes are unchanging, this scheme is effective. When all nodes change, though, all transmit, incurring a high cost.

Spatial Suppression This scheme lets nodes suppress if their values are the same as their neighbors'. The scheme exploits local data correlation, since nearby nodes are most likely to have similar values. Spatial suppression can be done by having all nodes attempt to report their values at different slots during a timestep. Nodes overhear reports sent by neighbors. When a node's slot comes up, it first computes the average of values overheard so far. If its value is close to this average, it suppresses. The root then fills in missing values with the average of neighbors. This approach has a significant flaw. To accurately derive a node's value, the root must know which of its neighbors were averaged to trigger suppression. The average of all neighbors may differ from the average of that subset. Rectifying the problem requires a global reporting order of all nodes, known by the root so it can correctly derive a node's value by averaging only the values that precede it in the order. Due to clock skew among nodes, holding to a global order is likely expensive in practice.

Spatio-Temporal Suppression This policy attempts to combine the advantages of both schemes. The obvious policy is to suppress nodes if they qualify for either type. This approach is problematic, however, because this "or"ing creates ambiguity; the root has no way of knowing if a missing value is temporally or spatially suppressed. If these are in conflict, the correct value cannot be derived. One solution is to "and" the schemes and suppress only when a node qualifies for both. The resulting approach would be less effective than using only one suppression type, since it can only restrict suppression chances. Therefore, there is no benefit to this combination.

Nevertheless, there is potential in combined suppression. If a node's value does not change, it should not report. Additionally, if a node's value does change, but its relationship to

[*]The authors are supported by NSF CAREER award IIS-0238386 and grant CNS-0540347.

its neighbors does not, it should not report. Consider the network in Figure 1. In one timestep, a band of nodes all rise in value (shift from white to black), causing the isoline bounding the high value area to move from left to right. Such a shift is a typical example of nearby nodes acting in a correlated fashion. The nodes in the interior of the band change value, but maintain the same relative relationships with their neighbors. Ideally, only one node, on the boundary of the band, should need to report to detect the band's movement. This scheme would achieve a dramatic improvement over requiring all nodes to report, as with temporal, or one node from every local cluster of nodes within the band, as with spatial.

Our Contribution As we have shown, neither temporal nor spatial are alone sufficient to exploit all suppression opportunities. It is not obvious, however, how to implement effective spatio-temporal suppression. To meet this challenge, we have developed the monitoring algorithm CONCH, short for "constraint chaining." It employs spatio-temporal suppression by monitoring changes to both node values and value differences along edges. We view each monitored node or edge as a constraint. By maintaining a chain of constraints, we can globally derive all values in order to support the isoline query.

2 Related Work

The isoline query is targeted in a number of papers. One approach similar to ours is *event contour* [7]. This method combines temporal and spatial suppression, but in the problematic way where the root cannot be sure which suppression type is used. In addition, their overhearing-driven spatial suppression means that even when all nodes have the same value, some must always report. Further, if two nodes have correlated but very different values, both will be reported, even though one is sufficient to derive the other. Other isoline-related problems are discussed in [4] and [1].

Temporal suppression based on value changes is used for continuous aggregation queries in [8]. A more sophisticated form of temporal suppression, proposed in [6], uses Kalman Filters to prevent a node from reporting only if a change in its value diverges from what a model would predict. The general idea of model-based suppression is orthogonal to ours; instead of using value-based constraints, we can use constraints on the parameter values of models that predict how values change.

3 Preliminaries

Our network consists of n fixed-location nodes $u_1, \ldots, u_n$, all measuring some feature such as temperature. An edge, e_{ij}, exists for each pair of nodes, u_i and u_j, within radio communication range of each other. The network is rooted at a base station node, and messages can be sent using some existing routing protocol [5, 10].

The primary source of energy usage is radio communication. We optimize for energy efficiency by minimizing the number and size of messages sent through the network, and evaluate algorithms accordingly. We use the energy specifications of MICA2 motes [2], and account for both per-byte sending/receiving costs and per-message startup costs.

The isoline query is essentially a continuous query requesting the tier values of all nodes. We simplify the notion of a continuous query with *rounds*. Each round is sufficiently long for all nodes to communicate among themselves and to the root. We assume the root initially disseminates the query plan into the network, but nodes act autonomously in each round. The root may infrequently update the plan in the network.

3.1 A First Cut: Neighborhood Approach

Before presenting CONCH, we introduce a neighborhood-based algorithm that supports an improved spatio-temporal suppression scheme. This algorithm returns the correct result, but has performance shortcomings we will further address in CONCH. In the neighborhood approach, each node maintains value information for each node within its communication distance. This set of nodes is the node's *neighborhood*.

We apply temporal suppression by giving all nodes the opportunity to regularly broadcast their values, but only having them do so if their tiers have changed since last broadcast. If u_i broadcasts a change in tier, all nodes that hear the message update their values for u_i in their list of neighbors. We apply spatial suppression as follows. u_i tracks the difference between its own tier, v_i and the tiers of each of its neighborhood nodes. Periodically, u_i has the opportunity to send an update report to the root. u_i remembers the last reported tier difference between itself and u_j, $(v_i^{old} - v_j^{old})$ and calculates the current difference, $(v_i - v_j)$. If these calculations differ, u_i includes the new difference in its report. This process repeats for each neighbor.

This suppression policy captures the features we sought in Section 1 to improve spatial suppression. Now, whether two neighbors are in the same or different tiers, if they move together, the policy suppresses. We expect such behavior due to local correlation. The scheme is spatio-temporal: spatial because it suppresses neighbors when their relationship stays the same, and temporal because it suppresses the need to constantly report neighbor relationships as time progresses.

Shortcomings The neighborhood approach has some key shortcomings. Say we have a collection of 8 nodes all in the same tier, v_l, but at the next timestep, 4 nodes jump to a higher tier, v_h. This example is depicted in Figure 2(a), with the black nodes having higher values. A horizontal axis is drawn to show the split between low and high values. The dotted lines indicate neighbor relationships between pairs of nodes. When the black nodes rise in value, many tier differences are detected between black and white nodes. These differences correspond to the dotted edges crossing the axis. According to the neighborhood approach, each node incident to such a dotted edge sends a message to the base station. The size of the message is proportional to the number of incident edges.

This illustration reveals the redundancy inherent in this approach. Each node contributes to several difference calculations, and is sent several times over. Although there are a lot of changes, the overall effect can be described succinctly: all nodes above the axis move to v_h. Our intuition is we should only need a single message to capture these changes.

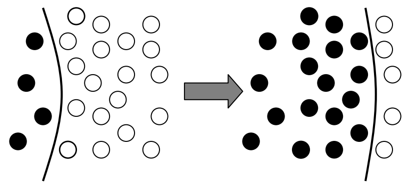
Figure 1. Moving isoline.

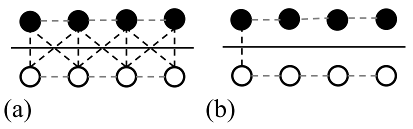

Figure 2. Edge monitoring.

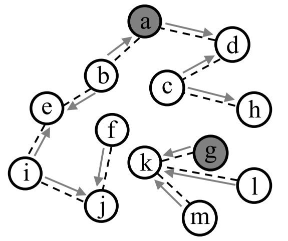

Figure 3. CONCH forest.

The problem can be explained by considering edges. The neighborhood approach monitors all edges between nodes within communication distance. This scheme causes the large number of messages triggered in our example, instead of our goal of only one to describe the change.

4 CONCH

We now present CONCH, the main contribution of this paper. The crux of CONCH is that we can support the spatio-temporal suppression strategy by monitoring far fewer edges than the neighborhood scheme, with a minimal spanning forest whose edges correspond to monitored constraints. We refer to Figure 2(b), where a spanning tree now connects the nodes. We notice that far fewer edges are monitored compared to the neighborhood approach. When the nodes above the axis move to v_h, only one edge observes a change in tier difference between its nodes, so only one message is sent. The black nodes' new values can be derived from this one message. Because none of the edges between the black nodes were reported, we know that each connected pair must continue to be in the same tier. If the left-most black node is now in v_h, then all the black nodes must be in v_h. We achieve energy savings by reducing the number of monitored edges while, through constraint chaining, being able to derive all node values.

4.1 CONCH Plan

Formally, a CONCH plan is a spanning forest SF over the sensor network. All nodes are covered by some tree in SF. Some subset of nodes (always including the base station) are designated as tree roots, and are the set of monitored nodes in the network, $roots(SF)$. The edges in the forest determine the set of monitored edges in the network, $edges(SF)$. For each $e \in edges(SF)$, one incident node is designated the *updater* and the other the *reporter*. By convention, for e_{ij}, u_i is the updater. Nothing prevents a node from serving either role on behalf of any of its incident edges, but each node serves only one role per edge.

The plan maintains constraints on $roots(SF)$ and $edges(SF)$. For each $u_i \in roots(SF)$, if v_i changes, v_i is sent to the base station. For each $e_{ij} \in edges(SF)$, if v_i changes, u_i sends an update to u_j. If the difference $(v_i - v_j)$ changes, u_j sends the new difference to the base station. Hence, CONCH combines two suppression strategies. Tree roots are maintained temporally, while edges are maintained spatio-temporally.

It is important to note that the spanning forest used in CONCH is usually different from the spanning tree used for routing. A CONCH forest is constructed in a way (discussed in detail in [9]) to minimize the expected cost of monitoring the constraints implied by the forest. On the other hand, a routing tree is constructed in a way to allow messages to reach the base station in as few hops as possible. For example, consider an example CONCH forest in Figure 3. The part of the forest rooted at node a contains a very long path to f, and would serve poorly as a routing tree. This separation of the CONCH forest and routing tree does not impact communication efficiency, however, because we still route all messages using the routing tree.

4.2 CONCH Algorithm

CONCH is divided into a start-up phase and continual phase.

Start-up A spanning forest SF covering the network is constructed from the set of all possible nodes and edges. It is built and maintained either by the base station, or in a distributed manner using techniques such as those in [3]. In either case, the base station is always informed of updates to SF. Each $e \in SF$ is assigned a reporter and updater. Figure 3 shows an example CONCH forest. Gray nodes are tree roots. The edge roles are depicted with gray arrows pointing from updater to reporter. For example, node c is an updater for edges e_{cd} and e_{ch}. The details of forest construction and edge role assignment are in [9]. Once SF is built and roles assigned, the base station disseminates this information into the network.

Each node, u_i, builds two lists of edges: those for which it serves as updater, Υ_i and those for which it serves as reporter, P_i. Υ_i stores only the counterpart node id of each of its edges. P_i stores both counterpart node ids and the last reported tier difference for each edge in it. For each edge e_{hi} where u_i is reporter, P_i maps u_h to the last reported tier difference on e_{hi}, $d_{hi} = (v_h - v_i)$. Initially, each node transmits its tier to all nodes in Υ, and likewise uses incoming transmissions to initialize d values in P.

Continual The continual phase occurs simultaneously at all nodes in response to tier changes. All operations occur at periodic timesteps. At each timestep a node u_i takes a sensor measurement. It calculates its tier value, v_i, and compares it with its previous tier value, v_i^{old}. If $\Delta v = v_i - v_i^{old} \neq 0$, u_i broadcasts v_i to all nodes in Υ_i. Additionally, if u_i is a tree root, it sends v_i to the base station.

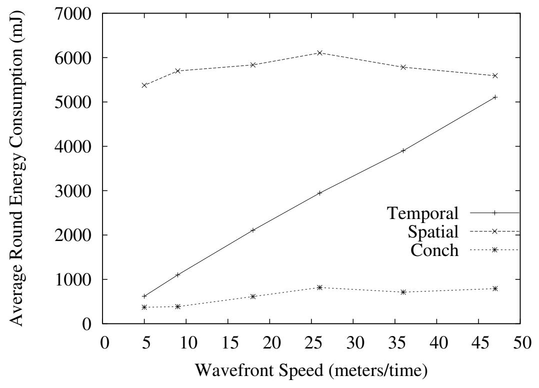

Figure 4. Wavefront scenario.

When a node u_i receives an update message from u_h, it immediately calculates d_{hi} and compares it to the stored result in P_i, d_{hi}^{old}. If $\Delta d_{hi} = (d_{hi} - d_{hi}^{old}) \neq 0$, u_i sends a report to the base station listing the edge, e_{hi}, and the updated difference, d_{hi}. It then stores d_{hi} in P_i.

The base station receives node and edge reports, and derives updates to all node tier values from the network of constraints. The base station tier value v_{bs} is always known. All tree root tiers and edge tier differences are known, whether left over from the previous timestep or just updated. The base station uses this information while it traverses its stored representations (not the actual network) of each tree in SF. Each traversal starts at a tree root r_i. All node tier values in the tree rooted at r_i are derived recursively: If node u_i has parent u_h, u_i then has value $(v_h + d_{hi})$. This approach correctly derives all node values; a proof is given in [9].

5 Evaluation

We give a preview of our results, drawn from our own network simulator. The network is a rectangular grid, with node density set at one per $200\ \mathrm{m}^2$. A full list of experiments can be found in [9]. We present here a wavefront scenario, where waves pass as vertical lines from left to right over the network. As soon as one wave reaches the end, another starts. Wave speed determines how often that happens. A node's value is tied to the distance covered by the recurring wavefront since it last passed over; value is highest when the wave is over it and lowest just before. Maximum value is 40 and tier width is 10, so nodes are in one of four tiers. We vary wave speed to see how each algorithm performs as the number of nodes changing tiers increases.

We compare three algorithms in Figure 4: temporal suppression, spatial suppression, and CONCH. The CONCH forest is built automatically using the techniques described in [9]. As wave speed increases, temporal suppression consumes more energy, since more nodes change tiers. Spatial suppression, with no notion of prior state, is unaffected by wave speed. It performs poorly simply because too many nodes report. These include nodes along the tier boundaries whose neighbors are not all in the same tier, and nodes that are the first in their neighborhoods to report. CONCH consistently outperforms the other algorithms by a large margin. Like spatial suppression, CONCH is oblivious to wave speed and is only affected by the number of tier borders. In particular, the only edges that ever must report to the root are those that cross tier boundaries.

6 Conclusion

We have presented the continuous isoline query for sensor networks, with applications in environmental and industrial monitoring. Due to slow-changing values and high geographic correlation in these scenarios, the query benefits from both temporal and spatial suppression. Our CONCH algorithm effectively and efficiently exploits both types of suppression. It builds a spanning forest over the network and monitors the values of root nodes and value differences along edges. This monitoring scheme significantly reduces the number of updates sent to the root, especially when changes affect large areas of nodes consistently. The set of updates collected at the root are used to derive all node values.

References

[1] K. Chintalapudi and R. Govindan. Localized edge detection in sensor fields. In *Proc. of the 2003 IEEE ICC Workshop on Sensor Network Protocols and Applications*, Anchorage, Alaska, USA, May 2003.

[2] Crossbow Inc. *MPR-Mote Processor Radio Board User's Manual.*

[3] R. Gallager, P. Humblet, and P. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. on Programming Languages and Systems*, 5(1):66–77, 1983.

[4] J. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Toward sophisticated sensing with queries. In *Proc. of the 2003 Intl. Conf. on Information Processing in Sensor Networks*, Palo Alto, California, USA, Apr. 2004.

[5] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *ACM/IEEE Trans. on Networking*, 11(1):2–16, 2002.

[6] A. Jain, E. Chang, and Y. Wang. Adaptive stream resource management using kalman filters. In *Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of Data*, Paris, France, June 2004.

[7] 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.

[8] M. Sharaf, J. Beaver, A. Labrinidis, and P. Chryanthis. Tina: A scheme for temporal coherency-aware in-network aggregation. In *Proc. of the 2003 ACM Workshop on Data Engineering for Wireless and Mobile Access*, San Diego, California, USA, Sept. 2003.

[9] A. Silberstein, R. Braynard, and J. Yang. Constraint-chaining: Energy-efficient continous queries in sensor networks. Technical report, Duke University, Durham, North Carolina, USA, Oct. 2005. `http://www.cs.duke.edu/dbgroup/papers/2005-sby-conch.pdf`.

[10] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proc. of the 2003 ACM Conf. on Embedded Networked Sensor Systems*, Los Angeles, California, USA, Nov. 2003.