Distributed Spatial Clustering in Sensor Networks

Anand Meka and Ambuj K. Singh

Department of Computer Science, University of California, Santa Barbara
Santa Barbara, CA.
{meka,ambuj}@cs.ucsb.edu

Abstract. Sensor networks monitor physical phenomena over large geographic regions. Scientists can gain valuable insight into these phenomena, if they understand the underlying data distribution. Such data characteristics can be efficiently extracted through *spatial clustering*, which partitions the network into a set of spatial regions with similar observations. The goal of this paper is to perform such a spatial clustering, specifically δ*-clustering*, where the data dissimilarity between any two nodes inside a cluster is at most δ. We present an *in-network* clustering algorithm *ELink* that generates good δ-clusterings for both synchronous and asynchronous networks in $O(\sqrt{N}\log N)$ time and in $O(N)$ message complexity, where N denotes the network size. Experimental results on both real world and synthetic data sets show that ELink's clustering quality is comparable to that of a centralized algorithm, and is superior to other alternative distributed techniques. Furthermore, ELink performs 10 times better than the centralized algorithm, and 3-4 times better than the distributed alternatives in communication costs. We also develop a distributed index structure using the generated clusters that can be used for answering range queries and path queries. The query algorithms direct the spatial search to relevant clusters, leading to performance gains of up to a factor of 5 over competing techniques.

1 Motivation

Sensor networks are being deployed over large networks to monitor physical phenomenon: to collect, analyze, and respond to time-varying data. The analysis and querying of sensor data should be done in a distributed manner in order to remove the performance bottlenecks and to avoid the single point of failure of a centralized node. We address the problem of discovering spatial relationships in sensor data through the identification of clusters. This clustering is achieved through in-network distributed algorithms.

Sensing phenomena such as temperature [2] or contaminant flows [5] over large spatial regions helps scientists explain phenomena such as wind patterns, and varying disease rates in different regions. For example, Fig. 1 shows the varying sea surface temperature regions in the Tropical Pacific [2]. Given such a heat map, a geologist can explain that the wind currents shown in the figure arise due to the pressure variations among the underlying temperature zones.

The goal of this paper is to partition the network into such a set of zones or *clusters*, which have observed similar phenomena. For example, consider the time series of four sensors (from Fig. 1) that are shown in Fig. 2. Notice that the spatially proximate sensors follow similar trends. Since our objective is to cluster regions based on the underlying trend, spatial clustering would group the top pair of sensors (shown in Fig. 2) into one cluster and the bottom pair into another.

Spatial clustering also serves to prolong network lifetime. Instead of gathering data from every node in the cluster, only a set of cluster representatives need to be sampled based on their spatio-temporal correlations. This reduces data acquisition and transmission costs [9, 14] in a sensor network constrained by storage, communication and power

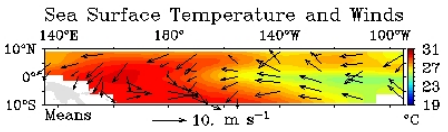


Fig. 1. Tao: Ocean monitoring with sensors

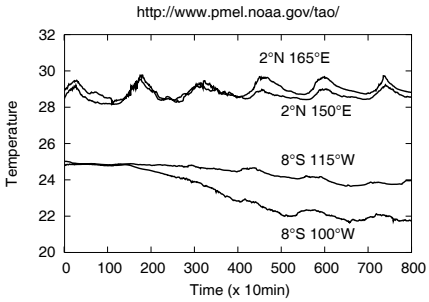


Fig. 2. Correlations in time series from four sensors

resources. Furthermore, there exists a need in the sensor network community to identify such clusters where *space-stationarity* holds. Eiman et al. [10] assume spatial stationarity of data in order to remove faults and outliers based on neighborhood and history. Guestrin et al. [14] perform in-network regression using kernel functions assuming rectangular *regions of support*. Our work addresses this important necessity to discover clusters that are both spatially stationary and are natural regions of support.

Clustering can be done off-line at the base station, if every node transmits its data to the central base station. But, this leads to huge communication costs. Besides, the power consumption for communication is up to three orders of magnitude higher than that for computation on a sensor node (such as a Crossbow Mica2 mote [3]). Therefore, for power efficiency, we propose *in-network* clustering. Furthermore, we regress time-series data at each node to build models. Clustering on model coefficients not only captures global spatio-temporal correlations, but also reduces transmission and memory costs. Overall, the contributions of the paper are:

1. We prove that δ-clustering is NP-complete and hard to approximate.
2. We present and design a distributed clustering algorithm called ELink that generates high quality clusterings in $O(\sqrt{N}\log N)$ time and in $O(N)$ message complexity, for both synchronous and asynchronous networks.
3. We present an efficient slack-parameterized update algorithm that trades quality for communication. Furthermore, we employ the spatial clusters to efficiently answer both range queries and path queries.
4. Our experimental results on both real world and synthetic data sets show that ELink's clustering quality is comparable to that of a centralized algorithm and is superior to other distributed alternative techniques. Furthermore, ELink performs 10 times better than the centralized algorithm, and 3-4 times better than other distributed alternative techniques in communication costs. For the query algorithms, the average communication gains were up to a factor of 5 over competing techniques.

2 Parameterized Clustering

We first define the clustering problem and discuss its complexity. Then, we briefly discuss the distance measure used for clustering.

2.1 Clustering: Definition and complexity

A good spatial clustering algorithm should group nodes in a sensor network based on data characteristics. In order to achieve this, data is regressed locally at each node to build models [26]. We adopt an auto-regression framework for defining the models (discussed in Section 2.2). The coefficients of this model are used as the *features* [15] at each node. We

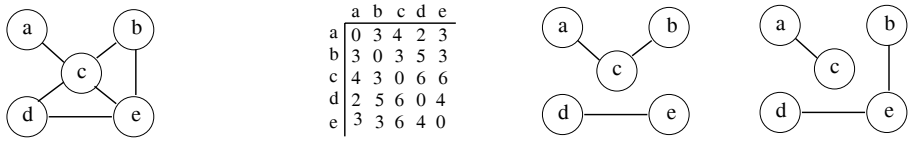
denote the feature at a sensor node i by F_i. The (dis)similarity between any two features F_i and F_j is captured by distance $d(F_i, F_j)$. We assume that the distance is a metric, i.e., it satisfies positivity, symmetry and triangle inequality. For example, consider the communication graph CG of a sensor network S as shown in Fig. 3a). Fig. 3b) shows an example of a distance metric $d()$ between the features of the nodes in S.
Using the dissimilarity threshold δ, a cluster is defined as follows:

Definition 1. *(δ-cluster) Given a set of sensors S, their communication graph CG, distance metric d, and a real number δ, a set of sensors C is called a* δ-cluster *if the following two conditions hold.*

1. *The communication subgraph induced by C on CG is* connected.
2. *For every pair of nodes i and j belonging to C, $d(F_i, F_j) \leq \delta$. We refer to this property as the δ-condition or δ-compactness.*

δ-Clustering is defined as the partition of the communication graph CG into a set of *disjoint* δ-clusters. Our goal is to find the optimal δ-Clustering, i.e., the clustering with the minimum number of δ-clusters.

Consider the network in Fig. 3a). If $\delta = 5$, then nodes c and e cannot belong to the same cluster since $d(F_c, F_e) = 6 > 5$, and for the same reason, nodes c and d cannot belong to the same cluster. Hence, the two possible minimal clusterings are as shown in Fig. 3c). Next, we present the complexity results for *optimal* clustering.



a) Communication graph (CG)

	a	b	c	d	e
a	0	3	4	2	3
b	3	0	3	5	3
c	4	3	0	6	6
d	2	5	6	0	4
e	3	3	6	4	0

b) Distance matrix

c) Two minimal clusterings

Fig. 3. For the communication graph in a) and distance matrix b), and $\delta = 5$, the minimal possible clusterings are shown in c).

Theorem 1. *Given a set of sensors S, a communication graph CG, a metric distance d, and a real number δ:*

1. *The decision problem* "Does there exist a partition of graph CG into m disjoint δ-clusters?" *is NP-complete.*
2. *It is not possible to approximate (in polynomial time) the optimal solution to δ-clustering within ratio n^{ϕ} where $\phi > 0$, unless P=NP.*

Proof. The proof is based on a reduction from the *clique cover* [19] problem. The clique-cover problem ($G = (V, E)$, c) states that the decision problem *"Does there exist a partition of graph G into c disjoint cliques?"* is NP-complete.

Given an instance I of the clique-cover ($G = (V, E)$, c), we map it into an instance I' of δ-clustering (CG, d, δ, m) as follows: Define CG as a clique of size $|V|$. Set δ equal to 1 and m equal to c. For every pair of vertices $i, j \in V$, define $d()$ as shown.

$$d(F_i, F_j) = \begin{cases} 1 & \text{if } e_{ij} \in E, \\ 2 & \text{otherwise.} \end{cases}$$

Note that $d()$ satisfies the metric properties. Output the solution of I' as the solution for I. From the above mapping, we can see that I has a solution iff I' has a solution. Also, there exists a one to one correspondence between the solutions of I and I'; so, the above reduction is approximation preserving. Since clique cover denies any approximation within ratio n^{ϕ}, therefore the same bound holds for δ-clustering as well. □

Since optimal δ-clustering is a hard problem, even in a centralized setting, we propose an efficient distributed algorithm that generates high quality clusterings. But first, we explain features and distance measures.

2.2 Feature model and distance

In order to discover the global spatio-temporal patterns in a sensor network, spatial clustering should be performed on the underlying trend rather than on the raw time-series data. Therefore, we construct a data model at each node to capture the structure in the data (e.g., the trends and cycle in the four sensors observed in Fig. 1). Each node models data using an Auto-Regression (AR) model. The general ARIMA model [26] captures the seasonal moving averages (MA) along with the daily up and down trends (AR). At a node i, the set of model coefficients represent the feature F_i.

In an AR(k) model, the time series of an attribute X at any node is modeled as $X_t = \alpha_1 X_{t-1} + \ldots + \alpha_k X_{t-k} + \epsilon_t$ where $\alpha_1, \ldots, \alpha_k$ are the auto-regression coefficients and ϵ_t is white noise with a zero mean and non-zero variance. Given m such data measurements at a single sensor node, the problem of finding auto-regressive coefficients can be stated in matrix notation as $\mathbf{Y} = \mathbf{X^T}\alpha + \mathbf{e}$ where $\mathbf{Y}$ is a $m \times 1$ column of known X_t values, $\mathbf{X}$ is $k \times m$ matrix of known explanatory variables $(X_{t-1}, \ldots, X_{t-k})$ and α is a $k \times 1$ column of unknown regression coefficients. Under basic assumptions of $\mathbf{e} = N(0, \sigma^2 I)$, the minimization of least squares errors leads to the solution $\hat{\alpha} = (\mathbf{X}\mathbf{X^T})^{-1}\mathbf{X}\mathbf{Y}$.

Next, we discuss the distance $d()$, between models. Consider the models at three nodes:

$$N_1 : x_t = 0.5x_{t-1} + 0.4x_{t-2} + \epsilon_t \quad (1)$$

$$N_2 : x_t = 0.5x_{t-1} + 0.3x_{t-2} + \epsilon_t \quad (2)$$

$$N_3 : x_t = 0.4x_{t-1} + 0.4x_{t-2} + \epsilon_t \quad (3)$$

Node N_1 is more correlated to N_2 than to N_3, because of the importance of higher order coefficients. Therefore, simple euclidean or Manhattan distance between the coefficients will not suffice. We need to consider a weighted euclidean distance on the model coefficients. Such distances are metrics. This motivates us to formulate the clustering problem in the context of metric spaces, rather than euclidean spaces.

3 Distributed Clustering Algorithm

In this section, we present and analyze a distributed algorithm, ELink, for *in-network* clustering. In the experimental section, we present three other alternative techniques: a centralized *spectral clustering* algorithm, and two other distributed clustering techniques, *Spanning forest* and *Hierarchical* for comparison. Section 9 discusses the drawbacks of extending the traditional clustering algorithms such as k-medoids-, hierarchical-, and EM- [8, 13, 23] based algorithms to this particular problem setting.

At the termination of the ELink algorithm, the communication graph *CG* is decomposed into disjoint δ-clusters. Each cluster is organized as a tree, referred to as a *cluster tree*, with the root as the designated leader. A node i inside a cluster C_i maintains a 3-tuple $\langle r_i, F_{r_i}, p \rangle$. The first is the root id, r_i; the second is the root feature, F_{r_i}; the third is the id of the parent, p, in the cluster tree.

3.1 ELink clustering

The key idea behind ELink is to grow clusters from a set of *sentinel* nodes to the maximal extent, i.e., until they are δ-compact, and then start growing another set of clusters from a different set of sentinel nodes, reiterating this process until every node is clustered. A definite order is imposed on the scheduling of the different sentinel sets; and moreover, a new sentinel set begins expanding only after the previous set has finished clustering. A node in the new sentinel set does not start expanding either until it is contacted by a node in the previous sentinel set (in an explicit signalling approach), or until its predefined timer expires (in an implicit signalling approach). Although both the techniques are guaranteed to run in $O(\sqrt{N}\log N)$ time and in $O(N)$ message complexity, the implicit signalling technique is designed for synchronous networks, whereas the explicit signalling technique is designed for asynchronous networks. We first give an overview of the general ELink algorithm, and then explain both the techniques in detail.

3.2 Algorithm

In order to understand the sentinel sets, we begin with a decomposition of the sensor network. To simplify the discussion, we assume a square grid of N nodes. Spatially, the entire topology is recursively broken down into cells at different resolutions (*levels*) in a quadtree like structure. The root cell is at level 0. Every cell elects a leader node [11, 16][1]. Sentinel set S_l comprises of all the cell leaders at a particular level l (as shown in Fig. 4), for $0 \leq l \leq \alpha$, such that $\sum_{l=0}^{\alpha} |S_l| = N$. Since $|S_l| = 4^l$, the depth of the hierarchy, α, evaluates[2] to $\log_4(3N+1)-1$. The parents of all the nodes in the sentinel set S_l comprise the set S_{l-1}. Initially, the single sentinel in S_0 begins expanding its cluster until it is δ-compact. Then, all the sentinels in S_1 are either explicitly or implicitly signalled to start expanding. This process is carried recursively at every level. The expansion of each sentinel is carried out only using the edges of the communication graph *CG*.

We use the term *root* (and *tree*) in two different contexts. The quadtree has a root, sentinel S_0. Every cluster C_i also has a root, cluster leader r_i, which is one of the sentinels belonging to $S_0, S_1, \ldots, S_\alpha$. The quadtree is used for the definition of the sentinels and their signalling. The cluster tree is used for defining the clusters.

The underlying idea behind the ELink algorithm is as follows. We first suppose that the whole network can be placed in a single δ-cluster; hence, we allot sufficient time for the cluster from S_0 to expand and include every node in the network. In that case, none of the lower level sentinels in $S_1, \ldots, S_\alpha$ start, and the single cluster remains intact. Otherwise, we suppose that the whole network can be partitioned into *at most* five clusters. So, cluster formation is initiated from each of the four sentinels in S_1. We allow nodes to switch cluster memberships a limited number of times. This handles the case when the number of clusters should be less than 5. Now, if the whole network is still not clustered after sentinel set S_1's expansion, we assume that the network can be decomposed into at most twenty one clusters (five from the previous levels), and start growing each of the sixteen sentinels in S_2. A sentinel set S_l's expansion begins *only* after sentinel set S_{l-1} terminates clustering. The implicit and explicit signalling techniques ensure that expansion happens strictly as above. Next, we explain how a sentinel node grows its δ-cluster.

[1] For routing purposes, the node closest to the cell centroid is elected as the leader.

[2] This is precise for a grid network. But, even under the general assumption of uniform network density [11, 18], α can be bounded by $\log_4(3N+1)+k$, for some small positive integer k, which is sufficient for all the subsequent theorems to hold.

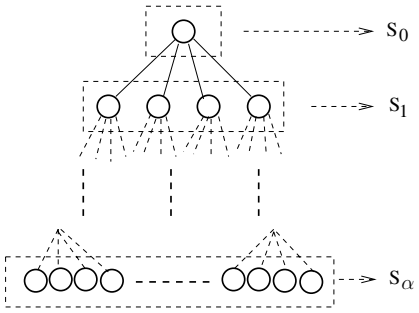


Fig. 4. Sentinel set S_0 comprises of the quadtree root. Sentinel set S_l comprises of all the leaders at level l.

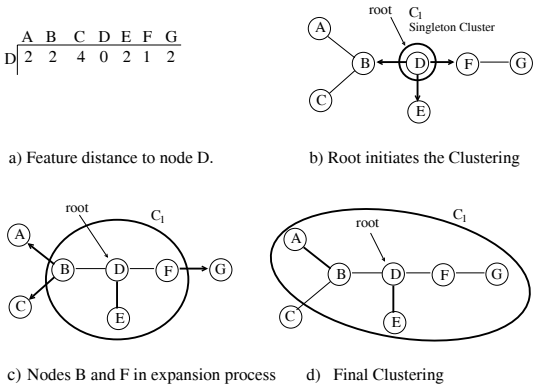


Fig. 5. For $\delta = 6$, the feature distances of every other node to sentinel node D are shown in a). The expansion of cluster C_1 starts as shown in b), continues as shown in c), and terminates as shown in d).

Once a sentinel node i at level l has been signalled, it examines if it is already clustered. If so, it does nothing. Else, it elects itself as the leader (root) of cluster C_i and sets $F_{r_i} = F_i$. Then, it attempts to include every neighbor j in its cluster, if $d(F_{r_i}, F_j) \leq \delta/2$. If node j is unclustered, it joins cluster C_i. If it is already clustered, we ensure that its reclustering does not destroy clusters grown from a lower level and that the gain in the clustering quality (measured by a distance in the metric space) is above a certain predefined threshold ϕ. Furthermore, we allow node j to switch clusters at most c times, where c is again a predefined constant. This is done in order to reduce the communication overhead. Constant c is application specific and is usually small, around 3–5.

If node j decides to be a member of cluster C_i, then j sets its root id to i, and stores the root feature F_{r_i}. It now attempts to expand cluster C_i. This process repeats until no new nodes can be added. Since the distance between the feature value of any node in the cluster and the root feature F_{r_i} is at most $\delta/2$, triangle inequality ensures that the distance between the feature values of any two nodes in the cluster is at most δ.

The following example (Fig. 5) illustrates the clustering by a sentinel node D, for $\delta = 6$. The metric distance of every node to sentinel D is shown in Fig. 5a). Initially, D sets itself as the root, as shown in Fig. 5b). Since $d(F_D, F_F) = 1 \leq 3 = \delta/2$, node D includes neighbor F in its cluster C_1, and transmits its root feature F_D to it. Similarly, neighbors B and E are included in cluster C_1, as shown in Fig. 5c). Nodes D and E cannot expand further, since all their neighbors are already clustered. Now, node F expands C_1 to include node G, since $d(F_D, F_G) = 2 \leq 3$. In a similar way, node B includes node A in C_1, but does not include node C, since $d(F_D, F_C) = 4 > 3$. After this step, none of the nodes in the cluster can expand, and so the clustering terminates. The final cluster C_1 is shown in Fig. 5d). The complete algorithm is outlined in Fig. 16 (Appendix B).

4 Implicit Signalling Technique

The implicit signalling technique is designed for synchronous networks. This algorithm ensures that the sentinel set S_l is granted sufficient time to complete expansion, before sentinel set S_{l+1} starts growing. Consider the bounding rectangle $[L \times L]$ of the entire N node network on the x-y plane. If ρ denotes the node density, then $\rho L^2 = N$. For the sake of simplicity, we assume $\rho = 1$ implying $L = \sqrt{N}$. We assume that the every

node has at most d neighbors. In sensor networks [12], d is assumed to be a constant and very small compared to N. Let stretch factor γ denote the ratio of the worst case increase in path length of "expand" messages (for cluster expansion) to the shortest path length between two nodes using multi-hop communication. Constant γ is usually small, around 0.2–0.4 [18]. For simplicity, assume that the worst-case delay over a hop is a single time unit. This blurs the distinction between path length and end-to-end communication delay between two nodes. Let κ denote the worst-case message cost or time (in a synchronous network) for the root sentinel S_0 to cluster with any other node in its level 0 cell (the whole network).

Note that $\kappa = (1+\gamma)\sqrt{\frac{N}{2}}$. Similarly, the worst-case time for a sentinel in S_m to cluster with any other node in its level m cell is $\kappa/2^m$. Extending this reasoning, a sentinel in S_l can cluster with any other node in the entire network in time $t_l = \kappa(1 + 1/2 + ..1/2^m + .. + 1/2^l)$. Therefore, every node in S_l is allotted a time interval t_l to finish expansion. Fig. 6 illustrates the interval t_1 of a sentinel node $A \in S_1$. Hence, scheduling of the sentinel set is done as follows. At time $T = 0$, sentinel set S_0 starts expanding, and every other sentinel set S_l starts its expansion at time $T = \sum_{i=0}^{l-1} t_i$. The algorithm is outlined in Fig. 17 (Appendix B).

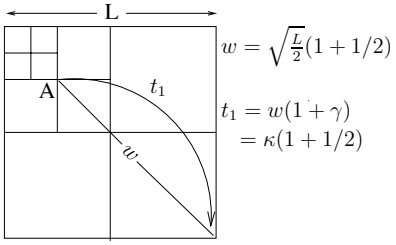


Fig. 6. The worst case path length from node $A \in$ sentinel set S_1 to cluster any node in the network is t_1.

Theorem 2. *The implicit signalled ELink algorithm runs in $O(\sqrt{N}\log N)$ time and requires $O(N)$ messages.*

Proof. First, we prove the bound on time. Since every sentinel set S_l terminates expansion before S_{l+1} starts growing, the total running time of the algorithm is bounded by the scheduling time for the sentinel set for the last level α, combined with its allotted expansion duration. Therefore, the total running time is $T = (\sum_{l=0}^{\alpha-1} t_l) + t_\alpha$. Since $t_1 < t_2 < t_3 \ldots < t_{\alpha-1} < t_\alpha$, and $t_\alpha < 2\kappa$, T is at most $2\kappa\alpha$. Hence, ELink runs in time $O(\sqrt{N}\log N)$.

Next, we consider the message complexity. A node sends out a message only in two cases; first, when it is scheduled as a sentinel, and second, when it has received a message from a neighbor, which is a member of a different cluster. In the first case, a sentinel node sends messages to all its neighbors, and since every node has at most d neighbors, the total cost over all nodes is at most dN. In the second case, a node sends a message only if it has switched clusters, or has just been clustered for the first time. As a node is restricted to switch cluster membership at most c times, it will send no more than $c+1$ messages to each of its neighbors. Hence, the N nodes will send at most $d(c+1)N$ messages. Therefore, ELink's message complexity is $O(N)$. □

5 Explicit Signalling Technique

The running time and message complexity of the implicit technique are guaranteed only for a synchronous network. In order to retain these complexities for an asynchronous network, we designed the explicit signalling technique, incorporating additional synchronization [4] into ELink. In this technique, a sentinel in S_{l+1} does not begin cluster expansion until it is explicitly contacted by its parent in S_l. This does not happen until every sentinel in S_l

completes expansion; thus, maintaining the order in sentinel set expansion: $S_0 \rightarrow S_1 \rightarrow \cdots \rightarrow S_\alpha$. Next, we give an overview of the synchronization that ensures this ordering.

The synchronization at every level l is divided into two phases. After realizing the completion of its cluster expansion, each sentinel in S_l begins the first phase, $phase$ 1, by contacting its quadtree parent. This parent, after being contacted by all its children, notifies its own parent. This is carried recursively up till the root sentinel, S_0. When the root is contacted by all its quadtree children, implying that all the nodes in S_l have finished expansion, it starts the second phase, $phase$ 2, by propagating messages recursively down the quadtree to notify all the nodes in S_l. After receiving such a $phase$ 2 message, each node in S_l instructs its children in S_{l+1} to start ELink.

The complete algorithm is shown in Fig. 18. During the ELink expansion of a sentinel in S_l, an intermediate node i along every path of cluster expansion, maintains a $children$ counter to denote the number of children it has in the cluster tree. Node i receives an $ack2$ message from a child j, when the cluster expansion of subtree rooted at j is complete. Then, node i decrements the $children$ counter by 1, and if the counter equals 0, then node i realizes the completion of cluster expansion of the tree rooted at i. Now, if node i is not the cluster root, it transmits an $ack2$ to its parent p. Else, it realizes that the entire cluster expansion has been completed, and contacts its quadtree parent by a $phase$ 1 message.

Theorem 3. *The explicit signalled ELink algorithm runs in $O(\sqrt{N}\log N)$ time and requires $O(N)$ messages.*

Proof. First, we prove the bound on time. A node i in the sentinel set S_l completes expansion in time interval t_l (defined in Section 4). Therefore, within $2 * t_l$ time, all $ack2$ messages denoting the completion of cluster expansion arrive at the sentinel. After this, there is the additional time of contacting the root sentinel S_0 with $phase$ 1 messages, and then, S_0 responding back with $phase$ 2 messages. The worst-case path length from a sentinel in S_m to its quadtree parent in S_{m-1} is bounded by $\kappa/2^m$. Thus, the total time for a sentinel in S_l, to contact S_0 in $phase$ 1 is at most $\kappa(1/2 + 1/2^2 + .. + 1/2^m + .. + 1/2^l)$, which is the same as $t_{l-1}/2$. Similarly, the time taken by $phase$ 2 can also be bounded by $t_{l-1}/2$. After the completion of two phases, sentinel i contacts the children in S_{l+1} via $start$ messages. This delay is bounded by $\kappa/2^{l+1}$. Hence, the total running time for all the sentinel sets is $T = \sum_{l=0}^{\alpha} 2 * t_l + \sum_{l=1}^{\alpha} t_{l-1} + \sum_{l=0}^{\alpha-1} \kappa/2^{l+1}$. In a manner similar to the implicit technique, the first and second summations evaluate to $O(\sqrt{N}\log N)$, whereas the third summation evaluates to $O(\sqrt{N})$. Hence, the time complexity is $O(\sqrt{N}\log N)$.

Now, consider the message complexity. In addition to the two types of clustering messages transmitted as in the implicit technique, nodes have to deal with four other types of messages— first, to inform the parents in the cluster tree ($ack1$) about their children; second, to inform the sentinel node about the completion of cluster expansion ($ack2$); third, the messages sent up the quadtree while notifying the root ($phase$ 1), and down the quadtree while receiving a reply ($phase$ 2); fourth, the messages sent to instruct the children ($start$) to invoke ELink. In the first and second cases, the total number of messages will be the same as those needed for cluster expansion as in the implicit technique, which is $O(N)$. We will now bound the total number of messages in $phase$ 1 and $phase$ 2, μ, over all levels. Let β_l denote the message cost of notifying the root by all the nodes in S_l in $phase$ 1. Then β_l can be recursively expressed as $\beta_l = \beta_{l-1} + |S_l| * \kappa/2^l$. Since $|S_l| = 4^l$, this recurrence yields the solution $\beta_l = (2^{l+1} - 1) * \kappa$. Hence summing over all sets S_l, the total cost of $phase$ 1 and $phase$ 2 messages, μ, can be expressed as shown below in equation (1).

$$\begin{aligned}\mu &= \sum_{l=1}^{\alpha} 2 * \beta_l = 2\sum_{l=1}^{\alpha}(2^{l+1}-1) * \kappa \\ &= 2\kappa\,(4(2^{\alpha}-1)-\alpha) \\ &= 2\kappa\,(2\sqrt{}(3N+1) - \log_4(3N+1) - 3) \end{aligned} \quad (4)$$

Since $\kappa = O(\sqrt{N})$, the total cost of the above term is $O(N)$. The total cost of $start$ messages is $\sum_{l=0}^{\alpha-1} |S_l| * \kappa / 2^{l+1}$. In a manner similar to Equation 4, this term also evaluates to $O(N)$. Hence, the explicit technique's message complexity is $O(N)$. □

Note that if optimizing the time complexity was our sole concern, then an unordered expansion of the sentinel suffices. We can expand all the sentinels simultaneously, subject to the constraint that a node can switch clusters at most c times. Since 2κ is the worst-case time for any node to reach any other node in the network, this algorithm will terminate in $O(\sqrt{N})$ time. The message complexity of this algorithm can be bounded by $O(N)$. However, this algorithm has poor clustering quality due to excessive contention across sentinel levels.

6 Dynamic Cluster Maintenance

After the clustering of distributed data sources has been carried out, the underlying data distribution may change[3] . This may lead to violations of the δ-compactness conditions within a cluster, necessitating an expensive re-clustering. In this section, we show that introducing a small slack [25] locally at each node avoids such global computations. Although this leads to a degradation in clustering quality, the resulting benefits in communication are huge.

Given a slack parameter Δ, the maximum divergence within a cluster, δ, is reduced to $(\delta - 2\Delta)$ during the initial clustering, and during any global cluster re-computation. Such a reduction gives a Δ slack for the feature update at node j for the δ-compactness condition.

Let F_i the feature at node i be updated to F_i' with the arrival of a new measurement. Similarly, let the feature at the root of node i, F_{r_i} be updated to F'_{r_i}. The root node verifies locally that $d(F_{r_i}, F'_{r_i}) \leq \Delta$. Node i verifies the conditions $\mathbf{A_1}$, $\mathbf{A_2}$ and $\mathbf{A_3}$.

If any of these conditions holds then it follows from triangle inequality that the δ-compactness property is not violated. If all the three conditions are violated, a re-clustering needs to occur. Node i propagates a message up the cluster tree to the root to obtain the updated root feature F'_{r_i}. After obtaining this feature, node i evaluates $d(F_i', F'_{r_i}) \leq \delta$. If the condition is violated, node i detaches from the cluster, and merges with the cluster of a neighbor k if $d(F_i', F_{r_k}) \leq \delta$. Else, it becomes a singleton cluster.

$$\begin{aligned} d(F_i, F_i') &\leq \Delta && (\mathbf{A_1}) \\ d(F_i', F_{r_i}) - d(F_i, F_{r_i}) &\leq \Delta && (\mathbf{A_2}) \\ d(F_i', F_{r_i}) &\leq \delta - \Delta && (\mathbf{A_3}) \end{aligned}$$

If the condition $d(F_{r_i}, F'_{r_i}) \leq \Delta$ is violated at the root, then the root propagates F'_{r_i} down to every node in the cluster tree. Every intermediate node computes its distance to this feature and decides if it should remain in the same cluster. The details of the update algorithm are deferred to the full paper [21].

7 Index Structure and Queries

In this section, we first discuss how a distributed index structure is built on the models, and then, describe how this index structure along with the δ-compactness property is employed to prune large portions of the sensor network for range and path queries.

[3] On the arrival of a new measurement, each node updates its AR model as shown in Appendix A.

7.1 Index structure

The ELink algorithm partitions the network into cluster trees that provide a natural way to build a hierarchical index structure. Our index structure is similar to a distributed *M-tree* [7] built on the feature space, but physically embedded on the communication graph. An index at node i maintains a *routing feature*, F_i^R, and a *covering radius*, R_i such that the feature of every node in the subtree rooted at i is within distance R_i from F_i^R. A leaf in the cluster tree propagates its routing feature $F_i^R = F_i$ and covering radius R_i (set to 0 for a leaf node) to its parent. The parent uses its own feature and the information from all its children to compute its own routing feature and covering radius. This process is carried on recursively up to the root of the cluster.

A range query q with radius r on an M-tree retrieves all the nodes whose feature values are within distance r from the query feature q. The range search starts from the root and recursively traverses all the paths leading to nodes which cannot be excluded from the answer set. A subtree rooted at node i can be safely pruned from search, if $d(q, F_i^R) > r + R_i$. The whole subtree satisfies the query, if $d(q, F_i^R) \leq r - R_i$. Since, each node stores the information of all its children, node i can prune a child-subtree rooted at j, if the condition $|d(q, F_i^R) - d(F_i^R, F_j^R)| > r + R_j$ holds; or it can include the subtree completely in the query if the condition $d(q, F_i^R) + d(F_i^R, F_j^R) \leq r - R_j$ holds. All the above follow from triangle inequality. Next, we discuss the range and path query algorithms.

7.2 Range querying in sensor networks

Geographic regions exhibiting abnormal behavior similar to that of the El Nino pattern [2] are of critical interest to scientists. These regions can be discovered by posing range queries of the form: "*Which are the regions behaving similar to node x?*" or "*Given a query model q, find all the regions whose behavior is within a specified distance r (measured in terms of model coefficients) of q*?".

A spanning tree connecting the leaders of different clusters (a *backbone* network) is built in order to efficiently route the query to every cluster. A range query can be initiated from any node in the network. The initiator routes the query to its cluster root, which forwards it to other cluster roots using the backbone tree. Every root first prunes using the δ-compactness property (explained next), and then employs the hierarchical index structure to selectively propagate the query to its children. Results are aggregated, first within the cluster tree, and then on the backbone network, and returned to the query initiator.

Pruning by the δ-compactness property is achieved as follows. No node in a cluster will satisfy the query q with radius r if query q's distance from the root $d(q, F_{r_i}) > r + \delta/2$. On the other hand, every node inside the cluster will satisfy the query if $d(q, F_{r_i}) \leq r - \delta/2$. (These follow from triangle inequality). If the query doesn't satisfy either of these conditions, the root employs the M-tree to prune the query (as mentioned in 7.1) in order to retrieve the answer set.

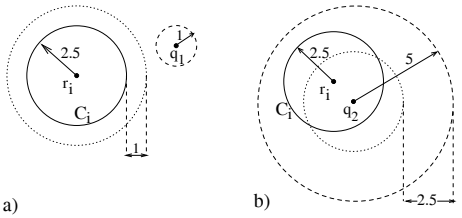


Fig. 7. Pruning conditions for $\delta = 5$: a) cluster C_i is *excluded* from the query $q_1(8, 9)$ with $r = 1$, b) cluster C_i is *included* in query $q_2(4, 6)$ with $r = 5$.

The above pruning is now illustrated using an example in which a model is represented by a tuple of two coefficients. For the sake of simplicity, assume euclidean distances. Let the root of cluster C_i (shown in Fig. 7) be F_{r_i}:$(3, 7)$ where $\delta = 5$. Fig. 7a) shows that no node inside cluster C_i can intersect the query q_1:$(8, 9)$ with radius $r = 1$ because $d(q_1, F_{r_i}) = 5.38 > 1 + 5/2$. Therefore, C_i can

be completely pruned. On the other hand, every node in the cluster C_i will satisfy the query q_2:$(4, 6)$ with radius $r = 5$ as shown in Fig. 7b), since $d(q, F_{r_i}) = 1.414 \leq 5 - 5/2$. Query q_3:$(3, 7)$ with $r = 1$ does not fall into either of the cases since $d(q_3, F_{r_i}) = 0 > 1 + 5/2$ is *false* and $d(q_3, F_{r_i}) = 0 \leq 1 - 5/2$ is also *false*. Therefore, it is injected into the M-tree root of cluster C_i. The M-tree pruning conditions are employed in this case.

7.3 Path querying in sensor networks

During pollutant leaks and fire hazards, rescue missions need to navigate a safe path from a source node to a destination node. Ensuring the safety of mission implies that the exposure to chemical along the path is at least a *safe* γ distance away from the danger (represented as a feature) F_D. Formally, a path query is posed as "*Return a path from the source node x to destination node y, such that for all the nodes j along the path, $d(F_j, F_D) \geq \gamma$*".

Path querying employs pruning similar to a range query. A cluster C_i with root r_i is *safe* for traversal if $d(F_{r_i}, F_D) > \gamma + \delta/2$; it is *unsafe* for traversal if $d(F_{r_i}, F_D) \leq \gamma - \delta/2$. Otherwise, the *safe* and *unsafe* regions inside a cluster are identified by drilling down the index structure, as safe and unsafe *sub-clusters*. Then, every set of spatially contiguous safe clusters (and sub-clusters) is connected using a *safe* backbone tree. Thus, the whole network is partitioned into disjoint set of safe trees.

The source node x forwards the query to the root of its cluster. If the cluster is evaluated to be *unsafe*, the root suppresses the query. Else, it dispatches a BFS query using the safe backbone tree to the root of the cluster (or sub-cluster), which contains the destination y. The whole path from the destination node y to source node x can be traced back. If node y does not belong to the same safe backbone tree as node x, then there does not exist a safe path between these two nodes.

8 Experimental Results

In this section, we first explain our real-world and synthetic datasets, then present interesting alternatives for clustering and querying, and finally evaluate a) the quality and the communication gains of ELink clustering, and b) the communication benefits achieved by the query pruning techniques over the presented alternatives. We only report the results for range queries. Results for path queries are deferred to the full paper [21].

8.1 Data sets

Tao (Spatially correlated dynamic data set): This data consists of daily sea surface temperature measurements of Tropical Pacific Ocean [2]. We obtain a 10-minute resolution data for a month (December 1998) from each sensor in a 6×9 grid. The sensors are moored to the buoys between $2S$–$2N$ latitudes and $140W$–$165E$ longitudes. The temperature range was $(19.57, 32.79)$ with $\mu = 25.61$ and $\sigma = 0.67$. The neighbors in the communication graph were defined by the grid. Each node is initialized with a model trained on the previous month's data. The temperatures within a day follow regular upward and downward trends, i.e, AR(1), whereas the daily variations in mean were observed to follow an AR(3). Hence, the temperature at every node is modelled as $x_t = \alpha_1 x_{t-1} + \beta_1 \mu_{T-1} + \beta_2 \mu_{T-2} + \beta_3 \mu_{T-3} + \epsilon_t$. The weight vector for distance computation is $(0.5, 0.3, 0.2, 0.1)$. Coefficient α_1 is updated for every measurement whereas β's are updated every day.

Death Valley data (Spatially correlated static data set): This data consists of geographic elevation of Death Valley [1]. Sensors are assumed to be scattered over the terrain and the elevation of the terrain at a sensor location is assigned as the sensor feature. The altitude range was $(175, 1996)$. Our performance results are averaged over 5 different random topologies, each consisting of 2500 samples.

Synthetic data (Spatially uncorrelated dynamic data set): Experiments were conducted on network sizes ranging from 100 nodes to 800. We used a random placement of nodes with a uniform probability distribution. Node densities were varied from 0.7 to 0.9. Each node has on the average 4 nodes within its radio range. Data at every node i is modeled as, $x_t = \alpha_i x_{t-1} + e_t$ where $e_t \sim U(0,1)$ and $\alpha_i \sim U(0.4, 0.8)$. $100,000$ readings were generated at each node. The range was $(10.00, 132.93)$ with $\mu = 69.27$ and $\sigma = 48.19$. Every node is initialized with $\alpha_1 = 1$. This model is updated for every measurement.

8.2 Performance metrics

Clustering: The clustering quality is measured by the number of clusters generated by each algorithm.
Communication: Communication overhead is measured by the total number of messages exchanged for each algorithm. A message can transmit a single coefficient or a data value. The query cost is the average number of messages required to route the query, and to aggregate the results back at the originator. The cost of building the inter-cluster leader backbone network is accounted in the ELink algorithm.

8.3 Alternative clustering and querying techniques

The performance of the ELink algorithm is compared against a centralized algorithm [22] and two other distributed algorithms that we propose. Our range query algorithm is compared against TAG [20] and our path query algorithm is compared against BFS.

Centralized clustering: There are two kinds of centralized algorithms. In the first, every update to the raw data is sent to the centralized base station (for baseline comparison in communication). In the second, an AR model is built at each node, and the model coefficients are sent to the centralized base station if the coefficient changes by more than a certain threshold [25]. We explain how this threshold is set in Section 8.5. At the base station, a *spectral decomposition algorithm* [22] is used for clustering. It computes the Laplacian L of the affinity matrix and then partitions the network into k clusters using the k largest eigenvectors of L. If x denotes the distance $d(F_i, F_j)$ between any two nodes i and j, then we define the affinity matrix $a()$ as follows:

$$a(i,j) = \begin{cases} x & \text{if } e_{ij} \in CG, \\ 0 & \text{otherwise.} \end{cases}$$

The algorithm is repeated with different values of k and the smallest k is chosen such that each cluster satisfies the δ-condition.

Spanning forest based clustering: This algorithm generates sub-optimal clusters in a greedy manner, but incurs a low communication cost. It consists of two phases. In the first phase, the algorithm decomposes the network into a spanning forest of trees, and in the second phase, it partitions each tree greedily into subtrees which satisfy the δ-compactness property.

In the first phase, each node selects the neighbor with the smallest feature distance, and an id smaller than its own id as its parent (to ensure a partial order). By iterative expansion, this phase decomposes the network into a forest of trees. In the second phase of the algorithm, these trees are checked for δ-compactness. Variable *height* at a node stores an upper bound on the feature distance between the node and any leaf belonging to the cluster subtree of the node. Every node initializes its *height* to 0. Beginning with the leaves, each leaf i sends its *height* (0) and its feature F_i to its parent p. Each parent node p maintains its own *height*, its local feature F_p, and the identifier of the child with the maximal height in a variable *highest_child*. When it receives a new $height$ from one of its children i, it uses a temporary variable h to store the value $height + d(F_i, F_p)$, and then examines if the sum of h and the local $height$ variable at node p exceeds δ. If it does, then node p instructs the

child whose height is the largest (*highest_child*) to detach. Otherwise, node p updates its *height* and *highest_child* variables. After receiving the heights of all its children, p sends its own height and feature F_p to its parent. Every detached subtree forms a new cluster with the *highest_child* as the root. The time and message complexity of this algorithm is $O(N)$.

Hierarchical clustering: The second distributed algorithm, Hierarchical clustering, grows the clusters in a hierarchical fashion using a notion of optimality absent in the spanning forest algorithm. Every cluster maintains a feature diameter and spatially neighboring clusters whose merger increases the diameter the least are merged in a bottom-up hierarchical fashion [23].

For a given cluster, every neighboring cluster is a candidate for merger if it does not violate the δ-condition. A *fitness* value is defined for all the candidates. The candidate with the minimum *fitness* is called the *best_candidate*. A pair of clusters merge if they are the *best_candidates* with respect to each other. This merger continues recursively until there is a single cluster in the whole network or no further mergers are possible.

Assuming that k clusters (trees) have been generated, we now explain how two neighboring trees C_i and C_j merge. Every cluster C_i maintains its diameter m_i. Clusters C_i and C_j verify the condition: $m_i + d(F_{r_i}, F_{r_j}) + m_j \leq \delta$. If they violate the condition, then C_i and C_j rule each other out as candidates for merger; else, they evaluate the *fitness* of the possible merger. The fitness of the merger is determined by m_{ij}, the diameter of the merged cluster. If $m_i \geq m_j$ then m_{ij} is set to $\max(m_i, m_j + d(F_{r_i}, F_{r_j}))$, else it is set to $\max(m_j, m_i + d(F_{r_i}, F_{r_j}))$. Cluster C_i chooses the optimal candidate based on these values: $best\_candidate_i = \text{argmin}_j\ m_{ij}$. Clusters C_i and C_j merge if $best\_candidate_i = j$ and $best\_candidate_j = i$. The time and message complexity of this algorithm is $O(N^2)$.

TAG querying: TAG [20] is a tiny aggregation scheme, distributed as a part of the TinyDB (Tiny Database) package that runs on motes [3]. TAG uses a SQL like declarative query interface to retrieve data from the network. It consists of two phases. In the *distribution* phase, the query is pushed down into the network using an overlay tree network, and in the *collection* phase, the results are aggregated continually up from the children to parents and reported to the base station. We evaluate the pruning benefits achieved by our range query algorithm by comparing it to TAG.

8.4 Clustering quality

Figs. 8 & 9 compare the clustering quality of ELink with the competing schemes, for varying δ. The threshold decrease required to switch a cluster, ϕ, was set to 0.1δ, and the maximum number of switches allowed, c, was set to 4. The Implicit and Explicit signalled ELink algorithms output the same clusters, except that the Explicit ELink has a higher communication cost than the Implicit one due to additional synchronization. The clustering quality of these algorithms is almost as good as the centralized scheme for all the data sets. Notice that ELink generates better clustering quality than the Hierarchical and Spanning forest algorithms. The Hierarchical algorithm performs better than Spanning forest, as it employs the *fitness* function to optimize the diameter. Results for the synthetic data set [21] were similar, except that there was an increase in the number of clusters due to little data correlations among spatial neighbors.

8.5 Communication costs

In this section, we evaluate the cluster update handling algorithm and the scalability of our algorithms in terms of their communication costs. Computational costs are negligible compared to communication costs in a sensor network. For brevity, we only report the representative results.

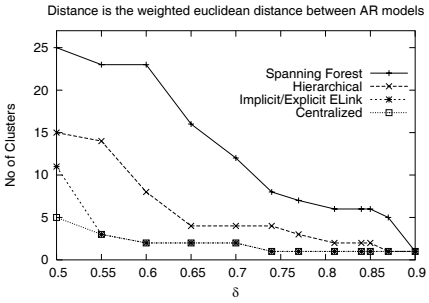


Fig. 8. Clustering quality for Tao data

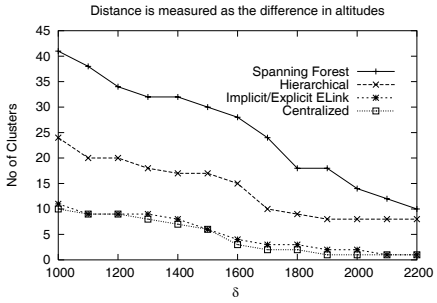


Fig. 9. Clustering quality for Death Valley data

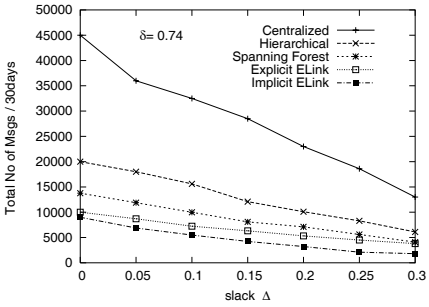


Fig. 10. Update costs with varying slack

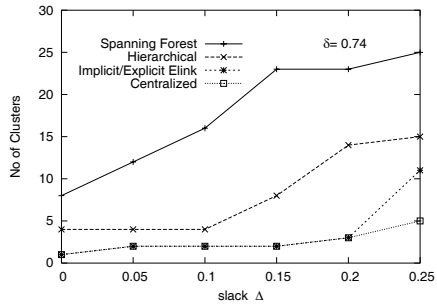


Fig. 11. Clustering quality with varying slack

Update handling: Every node in the centralized algorithm has to update the base station if the local model violates the slack Δ; hence, the algorithm incurs a huge communication overhead. Even if the slack condition ($\mathbf{A_1}$) is violated locally, the ELink update algorithm (in section 6) does not generate any messages if the conditions $\mathbf{A_2}$ and $\mathbf{A_3}$ are satisfied. Since these conditions require the cluster root feature F_{r_i}, and a node in the centralized algorithm does not maintain F_{r_i} locally, the centralized algorithm cannot prune by conditions $\mathbf{A_2}$ and $\mathbf{A_3}$. Due to these reasons, the communication cost of ELink algorithms is 10 times lower than the centralized algorithm as seen in Fig 10. As the slack is increased (effectively reducing the δ parameter), the quality of clustering decreases for all the algorithms as shown in Fig 11.

Scalability: Fig. 12 & 13 depict the scalability of the algorithms with time, and with the size of network. Fig. 12 shows the *log scale plot* of the scalability of algorithms on the Tao data set. We have included an extra plot for the centralized algorithm, in which every update to a raw value at a node is sent to the centralized base station. This figure illustrates that communication benefits obtained by modeling alone are an order of magnitude compared to raw data updates, whereas modeling combined by *in-network* clustering brings the cost down by another order of magnitude. Fig. 13 shows the scalability of algorithms with network size. We see the superior scalability of ELink based algorithms. This is because all the distributed techniques confine the updates locally, whereas the centralized scheme incurs a huge overhead of transmitting the model coefficients to the base station. Furthermore, Hierarchical clustering also incurs a huge cost since every merger decision has to be propagated to the cluster leader in order to evaluate the $best\_candidate$. Since Explicit ELink algorithm has additional synchronization costs, it incurs a larger overhead than the Implicit ELink algorithm.

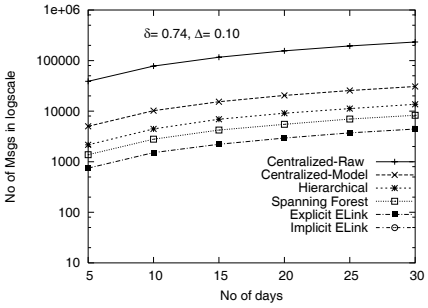


Fig. 12. Scalability with time on Tao data (shown in logscale plot)

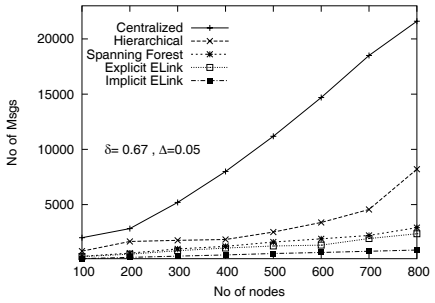


Fig. 13. Scalability with network size on synthetic data

8.6 Range querying

Figs. 14 & 15 show the average per-query cost when the range query algorithm is run separately on each clustering algorithm on both the data sets. The query point was sampled uniformly from the nodes. The query radius r was varied from $(0.7\delta, 0.9\delta)$ for the real data and $(0.3\delta, 0.7\delta)$ for the synthetic data. TAG is shown for comparison. Since TAG builds an overlay tree and aggregates the results back, the average number of messages per query is fixed and is equal to twice the number of edges in the spanning tree. In the real data set in Fig. 14, the clustering was compact, and hence ELink and Hierarchical pruned many clusters using the δ-compactness property, thus decreasing the query cost 5 times. But, as the query radius increased, the benefits of pruning by the δ-compactness property decreased, the pruning was now primarily due to the distributed index structure. Fig. 15 shows that there were less communication benefits for the synthetic data set. This is because the data was not spatially correlated.

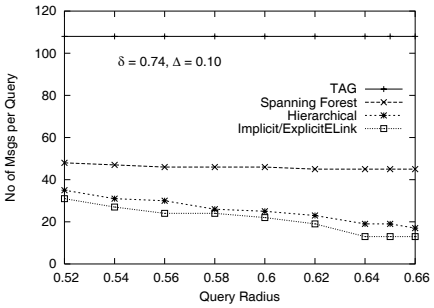


Fig. 14. Range query cost on Tao data

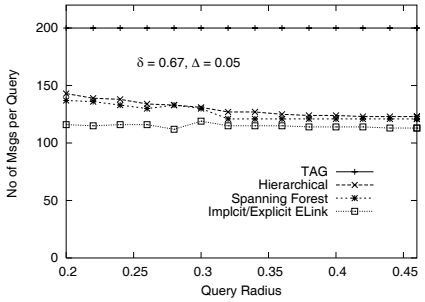


Fig. 15. Range query cost on synthetic data

9 Related Work

The general problem that the paper addresses—*clustering of data distributions* —has been extensively studied in statistics, machine learning and pattern recognition literature. There are three basic types of clustering algorithms: partitioning, hierarchical and mixture of Gaussian models. Partitioning algorithms such as k-means for euclidean spaces, or k-medoids for metric spaces (e.g. PAM, CLARANS [23]) represent each of the k clusters by

a centroid or an object. For our problem, distributed k-medoids would be communication intensive because in every iteration, all the medoids would have to be broadcast throughout the network so that every node computes its closest medoid. In hierarchical clustering (e.g. CURE [24]), the most similar pair of clusters are merged in each round, finally resulting in a single cluster. Our distributed hierarchical clustering technique is based on the same idea. But, this incurred a huge communication cost because of the exchange of data in in every round of merger.

Spatial data mining discovers interesting patterns in spatial databases. STING [28], a spatial clustering technique, captures the statistics associated with spatial cells at different resolutions, in order to answer range-queries efficiently. But it generates *isothetic* clusters whose boundaries are aligned to horizontal or vertical axis. WaveCluster [27] finds the densely populated regions in the euclidean space using the multi-resolution property of wavelets. It is a centralized scheme. In sensor networks distributed clustering has been studied for effcient routing purposes rather than for discovering data correlations [30]. Chintalapudi et al. [6] detect the edges of clusters (or phenomenon) in the specific setting where a sensor node emits only binary values. Instead of clustering, Kotidis [17] aims to determine the representatives among groups of sensors with similar observations.

10 Conclusions

We considered the problem of spatial clustering in sensor networks, and showed that is both NP-complete and hard to approximate. We presented a distributed algorithm, ELink, based on a quadtree decomposition and a level by level expansion using sentinel sets. Our algorithm generated good quality clustering, comparable to those achieved by centralized algorithms. Our algorithm is also efficient: it takes $O(N)$ messages and $O(\sqrt{N}\log N)$ time for both synchronous and asynchronous networks. Our experiments showed that ELink outperforms a centralized algorithm (10 times) and competing distributed techniques (3-4 times) in communication costs of clustering. We also answered range queries and path queries efficiently based on the δ-compactness property and by using a hierarchical index, resulting in communication gains of up to a factor of 5 over competing techniques.

Acknowledgements: This work was supported by the Army Research Organization grant DAAD19-03-D-004 through the Institute of Collaborative Biotechnologies.

References

1. The EROS data center for geological survey. *http://edc.usgs.gov/geodata/*.
2. Tropical atmosphere ocean project. *www.pmel.noaa.gov/tao/*.
3. *Crossbow, Inc. Wireless sensor networks*, http://www.xbow.com/.
4. B. Awerbuch. Complexity of network synchronization. *JACM*, 32(4):804–823, 1985.
5. J. Berry, L. Fleischer, W. E. Hart, and C. A. Phillips. Sensor placement in municipal water networks. *World Water and Environmental Resources Congress*, 2003.
6. K. K. Chintalapudi and R. Govindan. Localized edge detection in a sensor field. *SNPA*, 2003.
7. P. Ciaccia, M. Patella, and P. Zevula. M-tree: An efficient access method for similarity search in metric spaces. *VLDB*, 1997.
8. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the expectation-maximization algorithm. *Journal of Royal Statistical Society*, 9(1):1–38, 1999.
9. A. Deshpande, C. Guestrin, W.Hong, and S. Madden. Exploiting correlated attributes in acquisitonal query processing. *ICDE*, 2005.
10. E. Elnahrawy and B. Nath. Context-aware sensors. *EWSN*, 2004.
11. D. Estrin, R. Govindan, and J. Heidemann. Next century challenges: Scalable coordination in sensor networks. *MOBICOM*, 1999.
12. D. Ganesan, D. Estrin, and J. Heidemann. Dimensions: Why do we need a new data handling architecture for sensor networks? *SIGCOMM*, 2003.

13. V. Ghanti, R. Ramakrishnan, and J.Gehrke. Clustering large datasets in arbitrary metric spaces. *ICDE*, 1999.
14. C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden. Distributed regression: An efficient framework for modeling sensor network data. *IPSN*, 2004.
15. J. Han and M. Kamber. Data mining: Concepts and techniques. *Morgan Kaufmann*, 2001.
16. B. Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. *MOBICOM*, 2003.
17. Y. Kotidis. Snapshot queries: Towards data-centric sensor networks. *ICDE*, 2005.
18. Q. Li, M. DeRosa, and D. Rus. Distributed algorithms for guiding navigation across a sensor network. *MOBICOM*, 2003.
19. C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *JACM*, 41(5):960–981, 1997.
20. S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. *SIGMOD*, 2003.
21. A. Meka and A. K. Singh. Distributed algorithms for discovering and mining spatial clusters in sensor networks. *UCSB TechReport*, 2005.
22. A. Y. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *NIPS*, 2002.
23. R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. *VLDB*, 1994.
24. R. T. Ng and J.Han. Efficient clustering methods for spatial data mining. *VLDB*, 1997.
25. C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. *SIGMOD*, 2001.
26. M. Pourahmadi. Foundations of time series analysis and prediction theory. *Wiley*, 2001.
27. G. Sheikholeslami, S. Chatterjee, and A.Zhang. WaveCluster: A multi-resolution clustering approach for very large spatial databases. *VLDB*, 1998.
28. W. Wang, J. Yang, and R. R. Muntz. STING: A statistical information grid approach to spatial data mining. *VLDB*, 1997.
29. B. K. Yi, N. D. Sidiropoulos, T. Johnson, H. V. Jagadish, and C. Faloutsos. Online data mining for co-evolving time sequences. *ICDE*, 2000.
30. O. Younis and S. Fahmy. HEED: A hybrid energy-efficient distributed clustering for adhoc sensor networks. *INFOCOM*, 2004.

Appendix A: Online Updates to a Model

In this section we show how our model coefficients are updated incrementally. Let us assume that X is an $k \times m$ matrix of m measurements (one set of k input variables per column), α is the $k \times 1$ vector of regression coefficients and y the $m \times 1$ vector of outputs. The Least Squares solution to the over-determined system $X^T\alpha = y$ is the solution of $XX^T\alpha = Xy$. Let P denote XX^T and b denote XY. We can compute $\hat{\alpha} = P^{-1}b$, where

$$P_k = [\sum_{i=1}^{k} x_i x_i^T]^{-1} \text{ and } b_k = \sum_{i=1}^{k} x_i y_i \tag{5}$$

The operations above will be performed once. When a new vector x_{m+1} and output y_{m+1} arrive, the recursive equations for online model computation can be derived from [29] as:

$$b_k = b_{k-1} + x_k y_k \tag{6}$$

$$P_k = P_{k-1} - P_{k-1}x_k[1 + x_k^T P_{k-1} x_k]^{-1} x_k^T P_{k-1} \tag{7}$$

$$\hat{\alpha}_k = \hat{\alpha}_{k-1} - P_k(x_k x_k^T \alpha_{k-1} - x_k y_k) \tag{8}$$

Appendix B: ELink Algorithms

▷ **ELink clustering at node i** $(0 \leq i < N)$

l : level of node i in the quadtree.
$clustered$: boolean variable. Initially false.
m : level of the sentinel it is clustered by.
p : parent of i in the cluster tree. Initially set to i.
r_i : root of cluster to which i belongs.
F_i : feature value at node i.
F_{r_i} : feature value at root r_i.
$counter$: number of times a node can switch clusters. Initialized to c.
ϕ : threshold for switching clusters.

// Procedure executed upon receiving a signal
ELink (i) ::
if $(\neg clustered)$ **then**
$r_i := i$;
$clustered := true$;
$F_{r_i} := F_i$;
$m := l$;
send <"expand", $F_{r_i}, r_i, m>$ to all neighbors.

receive <"expand", $F_{r_j}, r_j, n>$ from a neighbor j ::
if $(d(F_{r_j}, F_i) \leq \delta/2$ & $(\neg clustered$ || $(n = m$ & $d(F_{r_j}, F_i) < d(F_{r_i}, F_i) + \phi$ & $counter \geq 0))$ **then**
$p := j$;
$r_i := r_j$;
$F_{r_i} := F_{r_j}$;
$m := n$;
if (*clustered*) **then**
$counter := counter - 1$;
$clustered := true$;
send <"expand", $F_{r_i}, r_i, m>$ to all neighbors;
// Explicit Signalling: send <"ack1"> to p.

Fig. 16. ELink clustering algorithm.

▷ **Implicit signalling at node i** $(0 \leq i < N)$

S_l : Sentinel set to which i belongs.
$\kappa := (1+\gamma)\sqrt{\frac{N}{2}}$ // Worst-case time for the quadtree root to cluster with any node in the network.
$t_l := \kappa(1+1/2+...+1/2^l)$ // Duration for i to expand.
$Timer := \sum_{j=0}^{l-1} t_j$ // Time after which i starts Elink.

$TimerExpires$ ::
ELink (i).

Fig. 17. Implicit signalling technique.

▷ **Explicit signalling at node i** $(0 \leq i < N)$

$children$: number of i's children in the cluster tree. Initially 0.
$quad\_children$: number of i's children in the quadtree. Initially 4.
$quad\_parent$: i's parent in the quadtree.
// The rest of the variables are from ELink algorithm.

// A node after sending the $expand$ messages (Fig. 16), determines that it is a leaf in the cluster tree if it does not receive any $ack1$ messages from its neighbors within a conservative time-out period.

if (i is a leaf)
send <"ack2"> to p.

$MessageHandler$ ::

// This message is received during
// ELink's cluster expansion.
receive <"ack1"> from j ::
$children := children + 1$.

receive <"ack2"> from j ::
$children := children - 1$;
if $(children = 0)$
if $(i = r_i)$ // i is the cluster leader.
send <"phase 1", $l>$ to $quad\_parent$;
else
send <"ack2", $l>$ to p.

receive <"phase 1", $n>$ from j ::
$quad\_children := quad\_children - 1$;
if $(quad\_children = 0)$
if $(i \in S_0)$ // the quadtree root
send <"phase 2", $n>$ to all quadtree children;
else
send <"phase 1", $n>$ to $quad\_parent$;
$quad\_children := 4$. // Reset for the next round.
receive <"phase 2", $n>$ from j ::
if $(l = n)$
send <"start"> to all quadtree children;
else
send <"phase 2", $n>$ to all quadtree children.

receive <"start"> from j ::
$quad\_parent := j$;
ELink (i).

Fig. 18. Explicit signalling technique.