Clustering Distributed Time Series in Sensor Networks

Jie Yin
Information Engineering Laboratory
CSIRO ICT Centre, Australia
jie.yin@csiro.au

Mohamed Medhat Gaber
Faculty of Information Technology
Monash University, Australia
mohamed.gaber@infotech.monash.edu.au

Abstract

Event detection is a critical task in many important applications of wireless sensor networks, especially for environmental monitoring. Traditional solutions to event detection are based on analyzing one shot data points, which might incur a high false alarm rate because sensor data is inherently unreliable and noisy. To address this issue, we propose a novel Distributed Single-pass Incremental Clustering (DSIC) technique to cluster time series obtained at sensor nodes based on their underlying trends. In order to achieve scalability and energy-efficiency, our DSIC technique uses a hierarchical structure of sensor networks as the underlying infrastructure. The algorithm first compress the time series produced at sensor nodes into a compact representation using Haar wavelet transform, and then incrementally groups the approximate time series into data clusters based on dynamic time warping distances. Experimental results on both real data and synthetic data have demonstrated that our DSIC algorithm is accurate, energy-efficient and robust with respect to network topology changes.

1 Introduction

The advent of wireless sensor networks has fostered growing interest in many important applications for environmental monitoring. Sensor networks facilitate the process to monitor the physical environments and make real-time decisions about events in the environment. In such monitoring applications, automatic event detection is an essential task, which aims at identifying emergent physical phenomena of particular concern to the users. When abnormal events are detected, the system will sound an alarm for immediate attention. Traditional solutions to event detection can be classified into threshold-based approaches [1] and pattern-based approaches [12, 22, 25]. Threshold-based approaches consider an event to occur when sensor readings exceed a pre-defined threshold value. Pattern-based approaches, on the other hand, represent events as spatio-temporal patterns in sensor readings and detect events using efficient pattern matching techniques. These approaches focus on analyzing one shot data points to detect emergent events. However, they may suffer from a high false alarm rate because sensor data is inherently unreliable and noisy.

In this work, we address the problem of clustering distributed time series in sensor networks. We aim at identifying homogeneous regions of sensor readings based on the underlying trends of time series obtained at sensor nodes. Each homogeneous region corresponds to a cluster of sensor nodes generating time series of similar trends. Obtaining these clusters can help to have a good understanding of sensor group behaviour in many environmental monitoring applications. For instance, in a coal mine surveillance application, when a gas leakage event occurs, the gas density readings measured at the sensor nodes near the source would follow a gradual decreasing trend. Therefore, identification of such an area in real time would help rescuers to evacuate workers in the mine safely.

Clustering distributed time series in sensor networks is inherently more complex than traditional clustering tasks [17]. First, sensor networks typically consist of small, battery-powered nodes with limited communication and computational capability. We therefore need to design an energy-efficient technique to cluster the time series obtained at sensor nodes. Second, sensor networks are usually deployed in a wide area, the clustering algorithm must be designed to operate in a distributed setting. Third, the communication links among sensor nodes are highly unreliable subject to constrained energy, making the network topology change over time. This requires our clustering technique to be robust to the changes of network topology. Therefore, it is highly desirable to design an *energy-efficient*, *distributed* and *accurate* approach to clustering distributed time series.

In this paper, we propose a novel Distributed Single-pass Incremental Clustering (DSIC) technique to deal with these problems. In order to achieve scalability and energy-efficiency, our DSIC technique uses a hierarchical structure of sensor networks as the underlying infrastructure, where

the sensor nodes are self-organized into *physical clusters* with one node selected as a cluster head for each physical cluster, and the cluster heads form a routing tree back to the gateway. Our DSIC technique works in two phases. In the first phase, the time series produced at sensor nodes are first transformed to a compact representation using Haar wavelet transform [13] and the selected wavelet coefficients are sent to a cluster head. Upon receiving new data from its children, a cluster head first reconstructs the time series and then incrementally construct a local clustering model based on Dynamic Time Warping (DTW) distances. In the second phase, the data clusters are merged across different physical clusters along the routing tree until the gateway obtains a global clustering model. To offset the effect of the data ordering on incremental clustering, we also devise a new heuristic strategy to ensure the quality of the clustering model. Experimental results on both real data and synthetic data have demonstrated that our DSIC algorithm is accurate, energy-efficient and robust with respect to network topology changes.

The remainder of this paper is organized as follows. Section 2 reviews previous work related to our problem. Section 3 discusses the sensor network infrastructure used in our algorithm. Section 4 describes our proposed DSIC algorithm in detail. Section 5 presents the results of experiments using simulation data. Section 6 concludes the paper and discusses directions for future work.

2 Related Work

There is extensive literature on data clustering in the data mining community [9]. In general, clustering algorithms can be classified into two major categories: *partitional algorithms* and *hierarchical algorithms*. k-means and its variants represent the category of partitional clustering algorithms that create a flat, non-hierarchical structure of k clusters. Hierarchical clustering is subdivided into *agglomerative methods*, which proceed by a series of merging operations that group the data into clusters, and *divisive methods*, which start to put all the data in one cluster and separate the cluster successively into small pieces. In the following, we discuss two data clustering techniques that are closely related to our problem: data stream clustering and distributed clustering.

2.0.1 Data Stream Clustering

Data stream clustering has attracted much attention in the past decade. Traditional algorithms cannot be directly applied to cluster data streams, because typically the data has an infinite volume and arrives at a high speed. Therefore, the ability to process the data in a single pass, while using little memory, is crucial for data stream clustering [14]. Dai et al. [5, 6] proposed a Clustering on Demand (COD) framework to dynamically cluster multiple evolving data streams. The COD framework produces a summary hierarchy of data statistics in the online phase, whereas the clustering is performed in the offline phase. Beringer and Hullermeier [2] proposed an online algorithm for clustering parallel data streams, which summarizes the data streams using the Discrete Fourier Transform (DFT) technique, and applies a k-means algorithm to cluster summarized data streams based on a weighted Euclidean distance measure. Likewise, Rodrigues et al. [18] proposed an Online Divisive-Agglomerative Clustering (ODAC) algorithm to incrementally construct a tree-like hierarchy of clusters using a top-down strategy. Data stream clustering techniques focus on summarizing the data by computing sufficient statistics in an incremental way, and then partitioning the summarized data into clusters. However, these techniques usually assume that all the data streams are gathered at a centralized site before they are processed. These centralized approaches cannot scale up in large sensor networks because data transmission typically consumes a lot of energy.

Recently, a distributed algorithm called Elink has been proposed to perform spatial clustering in sensor networks [16]. Elink uses Auto-Regression (AR) to model the time series obtained at individual nodes, and then, based on a communication graph, local clustering starts from a set of nominated root nodes and expands to include other nodes if the Euclidean distances between their model coefficients are less than a pre-defined threshold. However, the performance of Elink is limited because each cluster is coarsely represented by the feature of cluster root. Our proposed algorithm can outperform Elink to achieve a higher clustering accuracy as shown in the experiments.

2.0.2 Distributed Clustering

Distributed clustering assumes that the data to be clustered resides on different sites. Previous work on distributed clustering are usually based on the popular k-means algorithms. Instead of transmitting all the data to a central site, the clustering is performed on two different levels, i.e., the local level and the global level. At the local level, all sites carry out a local clustering independently from each other. At the global level, the central site is responsible for building a global clustering model based on the local models.

For example, Forman and Zhang [7] proposed a k-Harmonic Means algorithm to cluster homogeneously distributed data. During each iteration, each site computes and update the current k centroids based on its own data and broadcast their centroids to other sites. Once a site has received all the centroids from other sites, it can form its global centroids by taking a weighted average over the

entire data set. Kargupta et al. [10] developed a collective principal components analysis (PCA)-based clustering technique for heterogeneously distributed data. Each local site performs PCA, projects the local data along the principal components and applies a standard clustering algorithm. Having obtained the local clusters, each site sends a small set of representative data points to a central site. The central site then carries out PCA on the collected data and send the global principal components back to each local site.

However, most of the distributed clustering techniques mainly focus on one shot mining with respect to individual data points. Thus, they are not suitable for continuously clustering distributed data streams in wireless sensor networks, where the network topology is dynamic and the sensor nodes have limited communication range.

3 Sensor Network Infrastructure

To deal with the scalability and energy-efficiency of our distributed clustering algorithm, we adopt a hierarchical organization of the sensor network as the underlying infrastructure. The basic idea is to organize the network into different levels of granularity, ranging from small local areas at the lowest level to the entire network area at the highest level. More specifically, the sensor nodes are self-organized into a set of *physical clusters* based on available energy resources. Each physical cluster consists of a *cluster head* (CH) and several *cluster members*. The cluster head performs most of the computational tasks in each physical cluster. All the cluster heads form a multi-hop routing tree to the gateway.

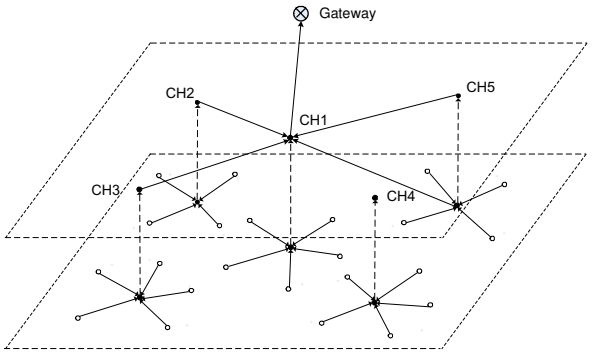

Figure 1. Sensor network infrastructure

As shown in Figure 1, at the lowest level of the routing tree, each cluster member sends its sensor data to the corresponding cluster head. Moving up the hierarchy, the cluster heads at the higher level are responsible for collecting data from the cluster heads at the lower level. For example, in Figure 1, node CH1 represents a cluster head in the second level of the hierarchy, and it receives the data from nodes CH2, CH3, CH4, and CH5. After receiving all the information from the lower levels, node CH1 sends the data back to the gateway at the highest node.

To obtain such a hierarchical structure, existing techniques, such as LEACH [23] and HEED [27], can be applied to select the cluster heads and determine the cluster members for each physical cluster. These techniques can ensure that the role of cluster heads rotates among all the nodes in the network, and that the self-organization of physical clusters is done in an energy-efficient way.

4 Distributed Incremental Clustering Algorithm

Our proposed clustering algorithm takes a hierarchical organization of sensor networks described in Section 3 as the underlying infrastructure. In general, it works in two phases: In the first phase, the data clusters are constructed locally within the members of each individual physical cluster. In the second phase, the data clusters are merged across different physical clusters to produce a global clustering model. We detail the two phases in the following.

4.1 Phase I: Clustering within a Physical Cluster

In the first phase, the main task is to construct a local clustering model for each physical cluster at the lowest level of the routing tree.

4.1.1 Data Compression Using Wavelets

In order to reduce the data transmission across the network, we propose to apply Haar Wavelet Transform to compress the time series obtained at individual sensor nodes. Haar wavelets have been found to be effective in providing good approximation of time series [4, 8, 3]. The Haar wavelet is chosen because it has a multi-resolution representation of time series and it can be computed quickly and easily, requiring linear time in the length of the sequence. Haar wavelet transform can be seen as a series of averaging and differencing operations on a discrete time function. Below, we give an example to illustrate the procedure to perform the Haar transform on a time series $f(t) = (9, 7, 6, 6)$.

The full resolution of the time series $f(t)$ is 4. In resolution 2, $(8, 6)$ is obtained by taking the average of $(9, 7)$ and $(6, 6)$ respectively, at resolution 4. $(1, 0)$ are the differences of $(9, 7)$ and $(6, 6)$ divided by 2, respectively. This process is recursively applied until resolution 1 is reached. Finally, the Haar transform of the original time series $H(f(t)) = (7, 1, 1, 0)$, which are called wavelet coefficients. The time series can also be reconstructed at different resolutions by

Table 1. An example of the Haar wavelet transform

Resolution	Averages	Differences
4	(9,7,6,6)	
2	$(8, 6)$	$(\mathbf{1}, \mathbf{0})$
1	$(\mathbf{7})$	$(\mathbf{1})$

adding differences back to or subtract differences from averages. For example, $(8, 6) = (7 + 1, 7 - 1)$, where 7 and 1 are the first and second coefficients, respectively. The motivation behind Haar transform is that elements of little variation in the original data manifest themselves as small or zero values in the transformed data. Therefore, we can approximate each original time series by selecting the k largest Haar coefficients so that the optimal amount of energy can be preserved per time series [24].

4.1.2 Incremental Clustering Process

Let CH_i be a sensor node elected as a cluster head of a physical cluster, and let CM_i be a set of cluster members belonging to the physical cluster. Each cluster member $v_j \in CM_i$ compresses the time series $f_{j,W}$ that it observes in a sliding window of size W using Haar wavelet transform, and transmits the compressed data, including the k largest coefficients along with their positions, to its corresponding cluster head CH_i. Upon receiving the compressed data, the cluster head reconstructs the time series by applying an inverse Haar transform to Haar coefficients, and incrementally groups the compressed data into a set of data clusters.

In order to discover meaningful clusters, we propose to use Dynamic Time Warping (DTW) [3, 26] to measure the distance of two time series. The reason for using DTW is that: First, sensor nodes are loosely synchronized across the network, and some nodes may suffer from package lost unexpectedly, the time series are not aligned exactly in the time axis; Second, there are varying time delays for different sensor nodes to detect an environmental event. Figure 2 shows three time series received by sensor nodes in the same time interval. The sensor network is used to measure light strength in our office area. The time series were collected when the lighting conditions changed. We can clearly observe that the three time series are of similar trends, although with different time shifts. Therefore, DTW is a good measure for similarity matching of sensing time series.

The time warping distance is computed using dynamic programming [26]. We consider two time series $x = (x_1, \ldots, x_N)$ and $y = (y_1, \ldots, y_N)$. Let $M(i, j)$ stores the shortest cumulative time warping distance from $(x_1, \ldots, x_i)$ to $(y_1, \ldots, y_i)$. $M(i, j)$ is computed as the

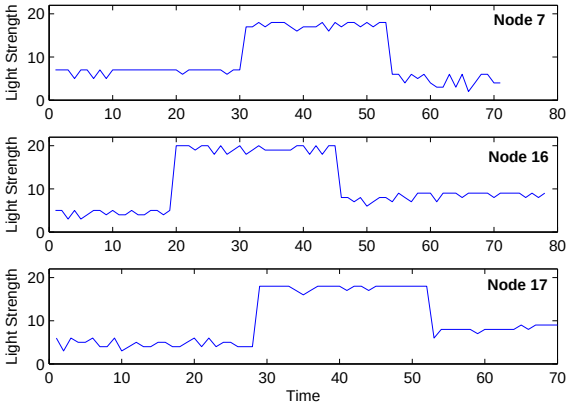

Figure 2. An example of three time series with different time shifts received by sensor nodes

Euclidean distance $D(x_i, y_i)$ and the minimum of the cumulative distances of adjacent elements:

$$M(i, j) = D(x_i, y_i) + \min \begin{pmatrix} M(i-1, j) \\ M(i, j-1) \\ M(i-1, j-1) \end{pmatrix}. \quad (1)$$

The general principle of the algorithm is to find the shortest cumulative distance for each pair of values between time series x and y, starting from the first pair (x_0, y_0), till the last pair (x_N, y_N). The final result is the shortest cumulative distance $M(x_N, y_N)$. Therefore, the complexity of the classic algorithm is $O(N^2)$ for two time series of length N. This quadratic algorithm is too much for sensor nodes with limited resources. Therefore, we adopt an accurate approximation algorithm FastDTW [21] to scale up the computation, which can run in linear time and space.

Based on the distance measure, each physical cluster head incrementally builds a local clustering model upon receiving new compressed data from its members. Since cluster members may send their compressed time series data to the cluster head in an unsynchronized manner, the data ordering might largely affect the overall quality of the distance-based clustering model. To address this issue, we propose a two-step heuristic strategy to offset the effect of data ordering on the clustering quality. In our method, each cluster is represented using four features:

- Cluster center C: is a vector in which each element represents the average of all the corresponding elements of the time series belonging to the cluster $\mathcal{C}$.
- Cluster upper bound C_u: is a vector in which each element is the maximum value among all the corresponding elements of the time series belonging to the cluster.

- Cluster lower bound C_l: is a vector in which each element is the minimum value among all the corresponding elements of the time series belonging to the cluster.
- Cluster size $|c|$: is defined as the number of time series belonging to the cluster.

Now we describe the process of incremental clustering performed for each physical cluster. The cluster head CH_i starts by initializing a data cluster for its own time series $f_{i,W}$. As wavelet coefficients are received from its cluster members CM_i, the cluster head incrementally update the data clusters as follows: The time series $\hat{f}_j$ is reconstructed from Haar wavelets coefficients and associated positions. We calculate the distances $D(\hat{f}_j, C_i)$ between the reconstructed time series and the cluster center C_i of existing clusters c_i. If there is no cluster within the distance threshold δ, a new data cluster c_{i+1} is created. If there are M clusters within the threshold δ, we further test the candidates by *simulating* the addition of the new time series to all of them and calculating the variances between the old and new cluster center. If all the variances exceed a pre-defined threshold ζ, that is, the addition of this time series would change the characteristics of all existing clusters dramatically, a new data cluster c_{i+1} is therefore created. By doing this, the time series $\hat{f}_j$ might have other chances to be merged with other clusters or new time series that comes later. Otherwise, for the cluster candidates whose center variances satisfy the threshold ζ, the time series $\hat{f}_j$ is assigned to the cluster c_k with the minimum variance of cluster centers. After determining the data cluster c_j to be merged with c_k, we update the cluster representatives for the cluster c_k using the following equations:

$$C_k^{new} = \frac{C_k \times |c_k| + \hat{f}_{j,W}}{|c_k| + 1}, \quad (2)$$

$$C_{u,k}^{new} = \max(\hat{f}_j, C_{u,k}), \quad (3)$$

$$C_{l,k}^{new} = \min(\hat{f}_j, C_{l,k}), \quad (4)$$

$$|c_k|^{new} = |c_k| + 1. \quad (5)$$

We summarize the detailed algorithm procedure in Algorithm 1. The *AtClusterMember* procedure (lines 3–6) and the *AtClusterHead* procedure (lines 7–28) present the clustering operations conducted at a cluster member and a cluster head, respectively. A cluster member $v_j \in CH_i$ is only responsible for compressing its time series data $f_{j,W}$ and sending the compressed data to its corresponding cluster head CH_i. For each cluster head CH_i, the *AtClusterHead* procedure starts by initializing a data cluster for its own time series $f_{i,W}$ and then incrementally construct a local clustering model for each physical cluster. The output of this algorithm is a list of data clusters constructed for each local physical cluster.

Algorithm 1 DSIC Algorithm: Phase I

```
1:  let CH_i be a sensor node elected as the cluster head in a physical cluster;
2:  let v_j ∈ CM_i be a cluster member belonging to the same physical cluster;
3:  procedure AtClusterMember(v_j)
4:    compress the time series f_{j,W} at each node v_j;
5:    send wavelets coefficients to cluster head CH_i;
6:    return;
7:  procedure AtClusterHead(CH_i)
8:    create a data cluster c_1 for its own time series f_{i,W};
9:    if a new time series is received from node v_j
10:     reconstruct time series f̂_{j,W};
11:     for each data cluster c_i do
12:       compute the distance D(f̂_{j,W}, C_i);
13:       calculate M = {c_i : D(f̂_{j,W}, C_i) ≤ δ};
14:       if M = 0
15:         create a new data cluster c_{i+1};
16:       else
17:         simulate adding f̂_{j,W} to c_i;
18:         compute ΔC_i = C_i^{new} − C_i^{old};
19:         if ΔC_i ≤ ζ
20:           assign f̂_{j,W} to c_k s.t. ΔC_k is minimized;
21:           update the cluster c_k using Eqn. (2)-(5);
22:         else
23:           create a new cluster c_{i+1};
24:         end if
25:       end if
26:     end for
27:   end if
28:   return;
```

4.2 Phase II: Clustering across Different Physical Clusters

After a local clustering model is constructed for each physical cluster, the next task is to merge the data clusters across different physical clusters along the routing tree. At intermediate levels of the routing tree, each parent cluster head collects the cluster representatives from the physical clusters at the lower level and merges their associated data clusters. The merged data clusters are then communicated to the parent cluster head in an upper level. This process continues until the gateway has a global clustering model.

Considering a data cluster c_j built at a cluster head CH_j at the lower level L_{m+1}, we calculate its distance to every data cluster c_i built at a cluster head CH_i at an upper level L_m. If there is only one data cluster within the distance threshold δ, we merge the two data clusters together and update the cluster representatives. If there are more than one data clusters c_i within the threshold δ, we calculate a

cluster merge criterion as follows:

$$link(c_j, c_k) = \frac{D(C_j, C_k)}{d(C_j)d(C_k)}, \quad (6)$$

where the terms $d(C_i)$ and $d(C_j)$ represent intra-cluster dissimilarity of clusters c_i and c_j, which can be calculated as $d(C_i) = D(C_{u,i}, D_{l,i})/|c_i|$ and $d(C_j) = D(C_{u,j}, D_{l,j})/|c_j|$, respectively. The principle is to merge a pair of clusters that have small self-similarity. In this way, clusters with a larger similarity are retained and the final clusters have larger self-similarity which can better represent actual events of interest. Finally, the data cluster c_j is merged with the cluster c_k such that $link(c_j, c_k)$ is minimized. Accordingly, the cluster representatives for cluster c_k is updated as follows:

$$C_k^{new} = \frac{C_k \times |c_k| + C_j \times |c_j|}{|c_k| + |c_j|}, \quad (7)$$

$$C_{u,k}^{new} = \max(C_{u,k}, C_{u,j}), \quad (8)$$

$$C_{l,k}^{new} = \min(C_{l,k}, C_{l,j}), \quad (9)$$

$$|c_k|^{new} = |c_k| + |c_j|, \quad (10)$$

where C_k and C_j are the centers of the two clusters to be merged, and $C_{u,k}$, $C_{u,j}$ and $C_{l,k}$, $C_{l,j}$ are the cluster upper bounds and lower bounds of the two clusters to be merged, respectively.

We now summarize the algorithm procedure in Algorithm 2. The *MergeClusters* procedure (lines 10–26) merges the data clusters $DataC_i$ built at a cluster head CH_i with the data clusters $DataC_j$ built at one of its child cluster heads CH_j. The merged data clusters are maintained using $DataC_i$ at the cluster head CH_i. The *MergeAtClusterHead* procedure (lines 4–9) recursively merges the data clusters $DataC_i$ built at the cluster head CH_i at the level L_m with the data clusters from all of its child cluster heads until the gateway obtains a global clustering model.

Algorithm 2 DSIC Algorithm: Phase II

1: let CH_i be a physical cluster head at the level L_m;
2: let $CH_j \in Children(CH_i)$ be a child cluster head at the level L_{m+1};
3: let $DataC_i$ and $DataC_j$ be the set of data clusters built at the cluster head CH_i and CH_j;
4: procedure **MergeAtClusterHead**(CH_i)
5: **for** each child node $CH_j \in Children(CH_i)$ **do**
6: initiate **MergeAtClusterHead**(CH_j);
7: initiate **MergeClusters**(CH_i, CH_j);
8: **end for**
9: return;
10: procedure **MergeClusters**(CH_i, CH_j)
11: receive the data clusters $DataC_j$ from CH_j;
12: **for** each data cluster $c_j \in DataC_j$ **do**
13: calculate $D(C_i, C_j)$ for any cluster $c_i \in DataC_i$;
14: compute $M = \{c_i : D(C_i, C_j) \leq \delta\}$;
15: **if** $M = 0$
16: add the cluster c_j to $DataC_i$;
17: **else if** $M = 1$
18: merge the two clusters c_i and c_j;
19: update the cluster c_i using Eqn. (7)-(10);
20: **else**
21: compute $link(c_j, c_k)$ using Eqn. (6);
22: merge c_j with c_k s.t. $link(c_j, c_k)$ is minimized;
23: update the cluster c_k using Eqn. (7)-(10);
24: **end if**
25: **end for**
26: return;

5 Experimental Evaluation

In order to evaluate the performance of our proposed algorithm, we performed extensive experiments on both synthetic data and real data. For comparison, three different clustering approaches were used as the baselines. The first approach applies group-average hierarchical clustering after all the raw time series are sent back to the gateway, which is referred to as Centralized. The second approach is called AR-Elink [16]. AR-Elink builds an AR model at each sensor node, and based on a communication graph, clustering starts from a set of nominated root nodes and expands to include other nodes if the Euclidean distances between their model coefficients are less than a pre-defined threshold. AR-Elink only uses the feature of root nodes to represent each cluster. The third approach differs from our proposed algorithm in that, after the time series is compressed using Haar wavelets, a standard clustering algorithm is applied for cluster forming and merging, solely based on the Euclidean distance to cluster centers. We call this approach Haar-Centre. Our proposed algorithm is referred to as Haar-DSIC in the experiments.

5.1 Evaluation Criteria

The performance of the clustering algorithms is assessed using the following two evaluation criteria.

- **Clustering quality:** The quality of clustering is measured using the well-known Silhouette score [11, 19]. It is a metric-independent measure designed to describe the ratio between cluster coherence and separation. Specifically, for each cluster C_k, we first compute a quality measure sil_{ik} for its member i as

$$sil_{ik} = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \quad (11)$$

where $a(i)$ is the average dissimilarity of member i to all other members of its cluster, and $b(i)$ is the average dissimilarity of member i to all members of the

closest cluster. The value of sil_{ik} varies from -1 to 1. The closer to one sil_{ik} is, the better the stream i is clustered. Accordingly, a silhouette score for a cluster C_k who owns m members is defined as $sil_{C_k} = \frac{1}{m}\sum_{i \in C_k} sil_{ik}$. Finally, the overall silhouette score is defined as

$$sil = \frac{1}{p}\sum_{j=1}^{p} sil_{C_k}, \tag{12}$$

where p is the total number of detected clusters.

- **Communication cost:** In order to evaluate the communication cost, we employ a simple energy model in our experiments, in which the radio dissipates $E_{elec} = 50nJ/bit$ to run the transmitter or receiver circuitry, and $\epsilon_{amp} = 100pJ/bit/m^2$ for the transmission to achieve an acceptable value of signal-to-noise ratio. To build the sensor network infrastructure as described in Section 3, we implemented the LEACH protocol [23] to generate physical clusters, in which each sensor node has an equal probability to become a cluster head at each iteration. After that, we built up a routing tree based on the distances between the gateway and cluster heads.

5.2 Experiments on Real Data

Experiments were first carried out on real data collected from a sensor network. We deployed 30 off-the-shelf motes (TMote Sky) with the light sensors (Hamamatsu S1087 PAR) to measure the light strength in our office area. All the motes were programmed to collect two samples of light strength every minute. The samples were packaged to a sink at 2405MHz radio frequency and then sent to a computer via a USB serial port. We chose light as the sensing modality for our experiments because it is relatively easy to control the light intensity in an indoor setting, and introduce temporal events by covering light sensors with paper cups. In our experiments, we set the length of the sliding window to be 64 for all the algorithms.

In our first experiment, we kept the light level constant in our office and stimulated a temporal event by covering a group of sensor nodes with colorful paper cups. By doing this, the time series generated by this group of nodes have different trends compared with the data obtained at other nodes. However, the time series obtained at other nodes might also correspond to several clusters because of different lighting conditions and random noises caused by people moving in our office. We applied the four clustering algorithms to this data set. For AR-Elink, the order of the AR model is set to be 3 because the best clustering result can be obtained. For Haar-Center and Haar-DSIC, the number of wavelet coefficients used to reconstruct the time series is set be 4. The clustering results are summarized in Table 2. We can see from the table that the clustering accuracy achieved by Haar-DSIC is very close to the performance of the Raw-Centralized algorithm. We can also see that both Haar-DSIC and Haar-Center outperform the AR-Elink algorithm to a large extent. This indicates that, compared with the AR model, Haar wavelet transform can capture more significant information contained in the original time series, which makes subsequent clustering more effective. In addition, our Haar-DSIC algorithm can achieve higher clustering accuracy that Haar-Center. This shows that, by employing the DTW-based distance measure, Haar-DSIC can better capture inherent similarity between time series with different local trends.

Table 2. Comparison of Clustering Accuracy

Clustering algorithm	Silhouette score
Raw-Centralized	**0.7410**
AR-Elink ($n = 3$)	0.4719
Haar-Center ($k = 4$)	0.6176
Haar-DSIC ($k = 4$)	**0.7356**

We also performed experiments to investigate the effect of data ordering on the clustering quality for Haar-Center and Haar-DSIC. Given a specific set of physical clusters, we randomly generated 50 different orderings of the data to be sent from the cluster members to each cluster head. Figure 3 shows the clustering accuracy with respect to different numbers of Haar wavelets coefficients. We can see that, when too few coefficients are used, the restored time series is too coarse to capture significant information from the original time series, and thus, the subsequence clustering becomes inaccurate. On the other hand, when too many wavelet coefficients are used, the restored time series has a fine granularity, which might also retain random noises involved in the original time series. As a result, the performance of the clustering algorithms might degrades as well. We can observe that, Haar-Center and Haar-DSIC can achieve the highest clustering accuracy when the number of wavelet coefficients is four. In the following experiments, we use four wavelet coefficients for Haar-Center and Haar-DSIC.

In addition, as the number of wavelet coefficients increases, our Haar-DSIC algorithm can be seen to consistently outperform Haar-Centre, with small variances in clustering accuracy. This is because, for Haar-Centre, the data clusters are formed solely depending on the Euclidean distance to the cluster centers, which makes the clustering accuracy quite sensitive to the data ordering. In contrast, Haar-DSIC improves the performance of Haar-Centres by considering the evolution of cluster centers in the cluster forming process.

Figure 4 shows the clustering result obtained by the Haar-DSIC algorithm. In total, Haar-DSIC generates three

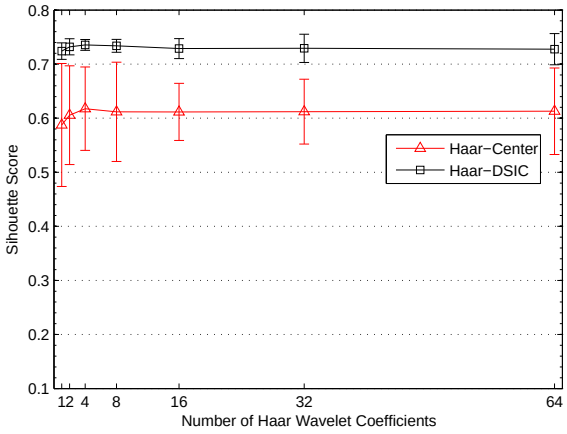

Figure 3. Clustering quality vs. Number of Haar Wavelet Coefficients

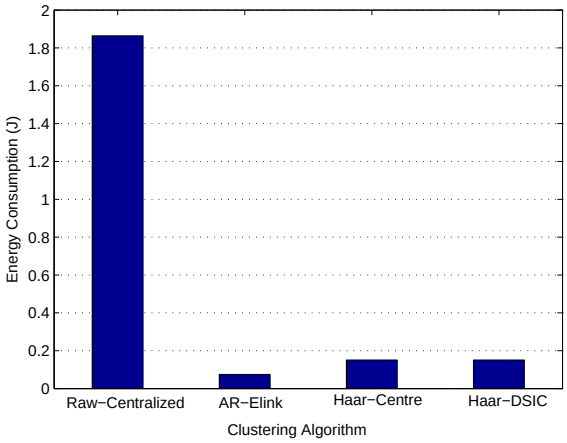

Figure 5. Comparison of energy consumption

clusters, as marked in the figure. Among them, "Cluster 1" corresponds to the temporal event simulated in our experiment, and the other two clusters reflect different lighting conditions in our office.

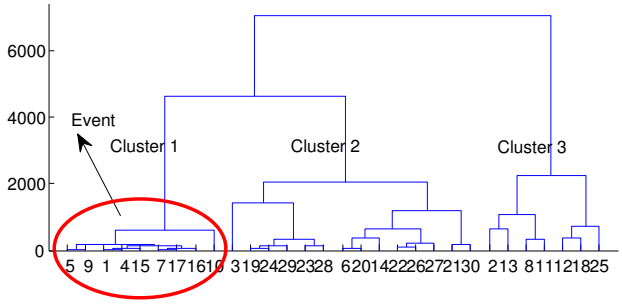

Figure 4. The clustering result obtained by the Haar-DSIC algorithm

Figure 5 compares the amount of energy consumed by the four clustering algorithms. We can see from the figure, AR-Elink, Haar-Center and Haar-DSIC can remarkably improves the energy-efficiency of Raw-Centralized by performing in-network clustering. The three algorithms perform local clustering and only cluster representatives are needed to be sent across the network. In contrast, Raw-Centralized needs to transmit the raw time series data back to the gateway and perform global clustering. We can also see that, Haar-DSIC consumes slightly more energy than AR-Elink because more data statistics are kept for each cluster as the cluster representatives, however, Haar-DSIC can achieve much better clustering quality than AR-Elink.

Experiments were also carried out to compare the robustness of the three distributed algorithms (AR-Elink, Haar-Centre and Haar-DSIC) with respect to network topology changes. Specifically, we used the LEACH protocol to randomly generate 20 different sets of physical clusters by initializing the number of physical clusters to be five. The three algorithms were then applied on the same data set. Table 3 shows the mean and the standard deviation of Silhouette score. We can see from the table that, based on different configurations of physical clusters, Haar-DSIC can achieve better clustering accuracy with less variance than Haar-Centre and AR-Elink. This indicates that Haar-DSIC is more robust than Haar-Centre with respect to network topology changes over time.

Table 3. Clustering quality vs. Network topology changes

Clustering algorithm	Silhouette score (variance)
AR-Elink	0.4319 (0.0658)
Haar-Center	0.6014 (0.0581)
Haar-DSIC	0.6896 (0.0346)

5.3 Experiments on Synthetic data

To study the scalability of our proposed algorithm, we simulated a sensor network at a square field of 100×100 meters. The default number of sensor nodes was set at 100, and the locations of sensor nodes were randomly generated in the field. We simulated the measurements generated by individual sensor nodes based on the Cylinder-bell-funnel data set. The generation of this data set was proposed in

[15, 20]. The data set contains three distinct classes and the time series data has similar local trends in each class. For each class, we generated 500 time series at a length of 128. The data generated by an individual sensor node was randomly selected from the total of 1500 time series.

We first performed experiments to compare the clustering quality of the four algorithms. The clustering results are summarized in Table 4. Each value of Silhouette score is the average of 20 trials. In this experiment, for AR-Elink, the order of the AR model is set to be three, and the number of wavelet coefficients is set to be eight for Haar-Center and Haar-DSIC. Again, we can see that, Haar-DSIC outperforms the other two distributed algorithms (AR-Elink and Haar-Center) to a large margin, and its accuracy is quite close to that of Raw-Centralized.

Table 4. Comparison of Clustering Accuracy

Clustering algorithm	Silhouette score
Raw-Centralized	**0.7937**
AR-Elink ($n = 3$)	0.1827
Haar-Center ($k = 8$)	0.3203
Haar-DSIC ($k = 8$)	**0.7657**

We then carried out experiments to compare the communication cost of the four algorithms. We varied the density of sensor nodes from 0.01 $sensors/sq.m$ to 0.1 $sensors/sq.m$ to study the scalability of the algorithms with respect to energy consumption. For this setting, we kept the number of physical clusters as eight. Figure 6 shows the amount of energy consumed by the four algorithms with respect to different values for the density of sensor nodes. We can see from the figure that, as the network density increases, AR-Elink, Haar-DSIC and Haar-Centre become more energy-efficient than Raw-Centralized. This is because, the three algorithms perform local clustering within the network, and thereafter only the cluster representatives are needed to be transmitted across the network. In contrast, Raw-Centralized need to transmit the raw time series data to the gateway before global clustering can be performed. Therefore, the three algorithms remarkably improve the energy-efficiency, especially for large-scale sensor networks. Although Haar-Centre and Haar-DSIC consumes more energy than AR-Elink to preserve more data statistics for each cluster, they can be seen to scale well with the increase in the density of sensor nodes.

6 Conclusions and Future Work

In this paper, we proposed a novel DSIC algorithm to cluster distributed time series in sensor networks. our DSIC technique uses a hierarchical structure of sensor networks as the underlying infrastructure, and incrementally construct a global clustering model along the routing hierarchy. Experimental results on both real data and synthetic data have demonstrated that our DSIC algorithm is accurate, energy-efficient and robust with respect to network topology changes.

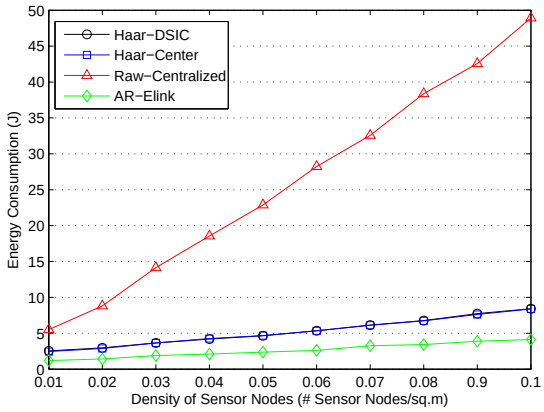

Figure 6. Energy consumption vs. Density of sensor nodes

Our work can be extended in several directions. First, we will implement our DSIC technique in a real event driven sensor network for monitoring soil moisture and evaluate its effectiveness in detecting the wetting front to minimize the use of irrigation water. Second, we will explore how to discover the boundaries of homogeneous regions based on our clustering results. Third, we will extend our technique to track the movements of homogeneous regions when the sensor readings change over time.

References

[1] D. Abadi, S. Madden, and W. Lindner. REED: Robust, efficient filtering and event detection in sensor networks. In *Proceedings of the 31st International Conference in Very Large Database (VLDB)*, pages 769–780, Trondheim, Norway, August–September 2005.

[2] J. Beringer and E. Hullermeier. Online clustering of parallel data streams. *Data and Knowledge Engineering*, 58(2):180–204, 2005.

[3] F. K.-P. Chan, A. W.-C. Fu, and C. Yu. Haar wavelets for efficient similarity search of time-series: With and without time warping. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):686–705, 2003.

[4] K.-P. Chan and A. W.-C. Fu. Efficient time series matching by wavelets. In *Proceedings of the 15th International Conference on Data Engineering (ICDE)*, pages 126–133, Sydney, NSW, Australia, March 1999.

[5] B.-R. Dai, J.-W. Huang, M.-Y. Yeh, and M.-S. Chen. Clustering on demand for multiple data streams. In *Proceed-*

ings of the 4th International Conference on Data Mining (ICDM), pages 367–370, Brighton, UK, November 2004.

[6] B.-R. Dai, J.-W. Huang, M.-Y. yeh, and M.-S. Chen. Adaptive clustering for multiple evolving streams. *IEEE Transaction on Knowledge and Data Engineering (TKDE)*, 18(9):1166–1180, 2006.

[7] G. Forman and B. Zhang. Distributed data clustering can be efficient and exact. *SIGKDD Explorations*, 2(2):115–122, 2000.

[8] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. One-pass wavelet decomposition of data streams. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):541–554, 2003.

[9] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2006.

[10] H. Kargupta, W. Huang, K. Sivakumar, and E. Johnson. Distributed clustering using collective principal component analysis. *Knowledge Information Systems*, 3:422–448, 2001.

[11] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An introduction to Cluster Analysis*. Wiler Interscience, 1990.

[12] M. Li, Y. Liu, and L. Chen. Non-threshold based event detection for 3d environment monitoring in sensor networks. In *Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS)*, pages 9–16, Toronto, Ontario, Canada, June 2007.

[13] T. Li, Q. Li, S. Zhu, and M. Ogihara. A survey on wavelet applications in data mining. *SIGKDD Explorations*, 4(2):49–68, 2002.

[14] J. G. M. Garofalakis and R. Rastogi. Querying and mining data streams: you only get one look. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 635–635, Madison, WS, USA, June 2002.

[15] S. Manganaris. *Supervised classification with temporal data*. PhD thesis, Department of Computer Science, Vanderbilt Universisty, 1997.

[16] A. Meka and A. Singh. Distributed spatial clustering in sensor networks. In *The International Conference on Extending Database Technology (EDBT)*, pages 980–1000, Munich, Germany, March 2006.

[17] P. Rodrigues, J. Gama, and L. Lopes. Requirements for clustering streaming sensors. In *First International Workshop on Knowledge Discovery from Sensor Data*, San Jose, CA, USA, August 2007.

[18] P. P. Rodrigues, J. Gama, and J. P. Pedroso. ODAC: Hierarchical clustering of time series data streams. In *Proceedings of the Sixth SIAM International Conference on Data Mining (SDM)*, pages 499–503, Bethesda, Maryland, USA, April 2006.

[19] P. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[20] N. Saito. *Local feature extraction and its applications using a library of bases*. PhD thesis, Department of Mathematics, Yale University, 1994.

[21] S. Salvador and P. Chan. FastDTW: Toward accurate dynamic time warping in linear time and space. In *The KDD Workshop on Mining Temporal and Sequential Data*, pages 70–80, Seattle, WA, USA, August 2004.

[22] I. Solis and K. Obraczka. Effcient continuous mapping in sensor networks using isolines. In *Proceedings of the 2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous)*, pages 325–332, San Diego, CA, USA, July 2005.

[23] A. C. W. R. Heinzelman and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences (HICSS)*, Maui, HI, USA, January 2000.

[24] Y.-L. Wu, D. Agrawal, and A. Abbadi. A comparison of DFT and DWT based similarity search in time-series databases. In *The Ninth International Conference on Information and Knowledge Management (CKIM)*, pages 488–495, McLean, VA, USA, November 2000.

[25] W. Xue, Q. Luo, L. Chen, and Y. Liu. Contour map matching for event detection in sensor networks. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 145–156, Chicago, IL, USA, June 2006.

[26] B.-K. Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *The 24th IEEE International Conference on Data Enginnering (ICDE)*, pages 201–208, Orlando, FA, USA, February 1998.

[27] O. Younis and S. Fahmy. HEED: A hybrid, energy-efficient, distributed clustering approach for ad-hoc sensor networks. *IEEE Transactions on Mobile Computing*, 3(4):366–379, 2004.