Distributed Weighted Clustering of Evolving Sensor Data Streams with Noise

Marwan Hassani, Thomas Seidl
Data Management and Data Exploration Group
RWTH Aachen University
Germany
{hassani, Seidl}@cs.rwth-aachen.de

Journal of Digital Information Management

ABSTRACT: *Collecting data from sensor nodes is the ultimate goal of Wireless Sensor Networks. This is performed by transmitting the sensed measurements to some data collecting station. In sensor nodes, radio communication is the dominating consumer of the energy resources which are usually limited. Summarizing the sensed data internally on sensor nodes and sending only the summaries will considerably save energy. Clustering is an established data mining technique for grouping objects based on similarity. For sensor networks, k-center clustering aims at grouping sensor measurements in groups, each contains similar measurements.*

In this paper we propose a novel resource-aware -center clustering algorithm called: SenClu. Our algorithm immediately detects new trends in the drifting sensor data stream and follows them. SenClu powerfully uses a light-weighted decaying technique that gives lower influence to old data. As sensor data are usually noisy, our algorithm is also outlier-aware. In thorough experiments on drifting synthetic and real world data sets, we show that SenClu outperforms two state-of-the-art algorithms by producing higher clustering quality and following trends in the stream, while consuming nearly the same amount of energy.

Categories and Subject Descriptors: I.3.7 [Computer Graphics]: Three- Dimensional Graphics and Realism—Animation; **I.3.5 [Computer Graphics]:** Computational Geometry and Object Modeling

General Terms:
Sensor Data, Data Algorithm

Keywords: Data Clustering, Data Processing, Algorithm, Wireless Sensor Nodes

Received: 25 July 2012, Revised 29 September 2012, Accepted 3 October 2012

1. Introduction

Nowadays, sensor networks are deployed in tens of applications from everyday scenarios. Particularly in Wireless Sensor Networks (WSNs), these applications start from home scenarios like the smart homes to environmental applications and monitoring tasks in the health sector [12] , but do not end with military applications. In all of these applications, monitoring is the dominating task of WSNs. Collecting useful data from remote sensor nodes is the ultimate goal of researchers and domain people who want to monitor some parameters using the WSN. The collection of all of the sensed data from all of the nodes within the network directly when they are sensed results usually in a perfect information gain about the monitored phenomena. However, not all of the sensed data are always interesting “enough” to be sent directly to the gathering station. The resulted excessive energy consumption when sending all sensed data to the gathering station encourages us to consider the previous sentence.

Wireless sensor nodes are spread in many scenarios over mountains or deserts or under the sea, where a continuous energy supply is impossible. In such cases, nodes are powered by batteries with a limited capacity. Moreover, the cost of changing the battery is in most of the cases bigger than getting a completely new sensor node deployed again. Sensor nodes consume energy while sensing, performing internal computations and during the communication of data with other nodes or with the central station. The radio part is the dominating energy consumer. Thus, minimizing the communication times of sensor nodes is mainly targeted when optimizing the energy consumption of a sensor network. By summarizing the sensed data internally on each sensor node, and then sending only the summaries of these data, one can

considerably reduce the updating frequency between the sensor node and the data collecting station. Apparently, this will compromise the information quality of the collected data. To obtain the maximum out of this trade off, some aggregation techniques should be carefully used that both the information gain and the resources usage time are maximized. In data mining, clustering is a task for summarizing data such that similar objects are grouped together while dissimilar ones are separated. In the special case of sensor networks, clustering of the streaming data aims at the summarization of similar sensor measurements. By detecting k representative measurements in k-center clustering for instance, one ensures good clustering quality if each representative is assigned to only very similar measurements. In addition to the clustering quality which constitutes the final information gain, one aims at an efficient cluster computation.

1.1 Challenges and Contribution

When designing an energy aware in-sensor-network clustering algorithm that considers drifting data with outliers, the following challenges appear:

• **Up-to-date Incremental Clustering:** The algorithm must incrementally cluster the stream data points to detect evolving clusters over the time, while forgetting outdated data.

• **Single Passing, Storage awareness:** Due to the limited processing and storage resources in the sensor node, the clustering algorithm must perform only a single pass over the incoming data stream and storage must be independent on n the size of input stream.

• **Minimal Communication:** As the energy consumption of transceiving data between the nodes is usually too big compared to the computation cost inside the node, the size of data being sent from the sensor nodes to the base station must be minimized.

• **High Clustering Quality:** The algorithm must show a good approximation to the optimal clustering by reducing the clustering radius as much as possible.

• **Outlier Awareness:** The algorithm should not be sensitive to outliers, nevertheless, it must be able to detect trends in the input stream.

Apparently, not only the two parts of the last aspect are contradicting each other. The third aspect is met when additional in-the-node computations are done to reduce the size of the data to be sent to other node or to the base station, which opposes the second aspect. On the other hand the fulfillment of the third and fourth aspect contradicts again achieving the second one, high-quality incremental clustering needs to store summaries of old points and needs additional processing. When considering the k-center clustering (will be explained later in Section 2.1), many attempts in the literature tried to fulfill the last two aspects, only limited work [4] [6] [9] has considered the first and the third ones. The EDISKCO algorithm [10] was the first algorithm designed to consider the last four aspects together. In spite of its powerful performance and efficient usage of energy, EDISKCO [10] did not mainly address the first aspect. In this paper, we propose a novel energy efficient -center clustering approach: *SenClu* that incrementally groups the data locally on each sensor node and computes its representatives. The algorithm gives more importance to newly received data for a faster adaptation to the evolving trends in the stream and more information gain of recent data on the collecting station. *SenClu* improves the clustering quality by using a novel light-weighted decaying technique to give less importance to old sensed values. It uses also a smart merging technique for the decayed clusters. The clusters with the least weight represent decaying data are smartly treated in *SenClu* as outliers. This gives a space for grouping the new emerging clusters, and thus following the current trend in the input streams. We sustain the powerful features of EDISKCO [10] but show that *SenClu* produces considerably better clustering results than EDISKCO [10] and another state-of the art competing algorithm, while consuming nearly the same amount of energy.

The remainder of this paper is organized as follows: Section 2 reviews previous work related to our clustering and communicating problem. In Section 3 we pass through some preliminaries. Section 4 describes our proposed *SenClu* algorithm in detail. Section 6 presents the experimental results. And finally we conclude the paper in Section 9.

2. Related Work

We will list some previous work done in two strongly related areas: approaches for sensor data clustering (particularly k-center clustering) and energy and data aware routing approaches for sensor networks.

2.1 Stream and k-Center Clustering Algorithms

DenStream [1] was the first stream clustering algorithm which used the weighing method to give recent data more influence by weighing the objects down according to their age. This work was followed by many other algorithms which used similar techniques (e.g. [5]). Although some of them were storage aware, these techniques were designed to perform the decaying method using enough processing and energy resources. Our algorithm in contrast applies a novel decaying and merging techniques that is light weighted enough to be implemented on the limited resources of sensor nodes. In the k-center clustering problem of a group of points P, we are asked to find k points from P and the smallest radius R such that if disks with a radius of R were placed on those centers then every point from P is covered [9] . The quality of the k-center clustering algorithms is measured using the approximation to the optimal clustering. Many clustering solutions have been presented for the k-center problem.

Offline Approaches assume that all of the n input points are stored in the memory. The "*Furthest Point*" Algorithm [8] and the "*Parametric Pruning*" [14] gave a 2-approximation to the optimal clustering by making $O(kn)$ distance computations. It is NP-hard to find a $(2-\epsilon)$-approximation to the optimal clustering of the k-center problem for any $\epsilon > 0$ [7] .

Online Approaches were developed to cope up with streaming input, the "*Doubling Algorithm*" [2] is a single pass streaming algorithm which guarantees an 8-factor approximation to the optimal clustering and uses $O(k)$ space. Cormode et. al [6] has formulated the "*Parallel Guessing Algorithm*" resulting with a $(2+\epsilon)$-approximation to the optimal clustering. This algorithm uses the first points in the input stream to make Δ guesses of the clustering radius R as $(1+\frac{\epsilon}{2}), (1+\frac{\epsilon}{2})^2, (1+\frac{\epsilon}{2})^3$,... This will end up by storing $O(\frac{k}{\epsilon}\log\Delta)$ points. The algorithm is very sensitive to the first k centers selected from received points and some of them might even be outliers, which reduces the clustering quality by using a big guess. The storage is dependent on Δ which can be in reality a big value for the limited storage of sensor nodes. In addition, the parallel nature of the algorithm does suite the limited processing ability of sensor nodes even for small values of Δ.

Distributed Approaches where distributed sites are considered, each is maintaining a k-center clustering algorithm on his local input stream. The idea was originally raised in [6] Here we have m remote sites applying the parallel guessing algorithm or an *online* furthest point algorithm on its local data. The site sends its k-centers to a central site called *coordinator* which in turn applies another k-center clustering algorithm on the centers. They proved that if the $k \times m$-center clustering algorithm on the site side gives an α -approximation and the one on the coordinator site gives a β-approximation then the resulting k-center clustering algorithm offers an $(\alpha+\beta)$ -approximation to the optimal clustering of the whole input data. A similar idea for a light weighted processing was used in [10] where sensor nodes are considered.

***k*-Center Clustering with Outliers** was first presented by Charikar et. al [3] in an offline algorithm with a approximation which drops z outliers. McCutchen et. al [15] presented an algorithm which gives a $(4+\epsilon)$-approximation using $O(\frac{kz}{\epsilon})$ memory space. EDISKCO [10] online z drops far and non-dense outliers by achieving $(2+\epsilon)$-approximation. In contrast, our algorithm: *SenClu* considers aged clusters with the least weights as outliers.

2.2 Energy-and-data-aware Routing Approaches in Wireless Sensor Networks

After collecting their measurements from the physical environment and processing them, sensor nodes have to send these data to one or more base stations. By having the base station(s) within the radio range of each sensor node, the naïve single-hop communication between each node and the base station is possible but not energy-efficient and not reliable because of possible resulting interferences. Figure 1 (Left) illustrates one solution to this problem. Low Energy Adaptive Clustering (LEACH) protocol [13] dynamically groups sensor nodes in a small number of clusters. The randomly chosen representatives (cluster heads) locally fuse data from their neighboring and transmit it to the base station, which results of a factor of 8 improvement compared to direct transmissions. In the Hybrid Energy-Efficient Distributed HEED Clustering Approach ý[16] the cluster head selection is mainly based on residual energy and the neighbor proximity of each node. In ECLUN [11] , a smarter representative selection process is performed not only by considering spatial similarities, but also similarities over some dimensions of measured data. This considerably increases the stability of the clusters and the representatives within the network, which results in an additional saving of the energy (cf. Figure 1 (Right)). Similar to EDISKCO, our algorithm uses a networking protocol that extends the lifetime of the wireless sensor network. The protocol efficiently groups local sensor nodes that locally send their data to one of them called coordinator which in turn aggregates these data and sends it to the far base station. The coordinator is iteratively changed depending upon the residual energy which is accurately estimated by our algorithm.

3. Preliminaries

3.1 The *k*-center Problem

Given a set P of n points $P=\{p_1,...,p_n\}$, a distance function $d(p_a,p_b)\geq 0$ satisfying the triangle inequality and an integer $k<n$, the k-center set is C contained in P such that $|C|=k$. The -center problem is to find a k-center set that minimizes the maximum distance of each point $p \in P$ to its nearest center in C; i.e., the set which minimizes the quantity $max_{p\in P} min_{c\in C} d(p,c)$. The well-known -median clustering problem is the minsum variant of this problem where we seek to minimize $\Sigma_{p\in P} min_{c\in C} d(p,c)$ [9].

3.2 Incremental *k*-Center Clustering Problem

Let $S=\{c_1,c_2,...,c_k,R\}$, be a *current* solution of a k-center clustering algorithm A applied on n input points that are arriving to the algorithm one by one in a sequence of updates, where: c_i; $i=1$ ...k are the centers and R is the radius of all of the clusters. A is an incremental clustering algorithm if it can always maintain a valid solution over the low of stream. In other words, whenever a new point arrives to the algorithm it should either be assigned to one of the clusters indicating the validity of current clustering, or it does not fit in any of the current S clusters then the current must be changed into another solution S' such that this new point is assigned to some cluster in the new solution S'. S' can differ from S by the centers, radius or by both of them.

3.3 Weighted *k*-Center Clustering

A weighted k-center clustering algorithm A uses the following structure to save information about a clustering

C:{c_1, w_1, t_{u1}, c_2, w_2, t_{u2},...,c_k, w_k, t_{uk}, R}. Where w_i is the weight of cluster i, c_i is its center and t_{ui} is the last time when the cluster was updated by a point from the stream input. Let t_{now} be the current time, the weight of cluster i is calculated as follows:

• [$w_i = 2^{-\lambda(t_{now}-t_{ui})}$] if the cluster i was not updated by the current stream input point (at time t_{now}).

• [$w_i = 2^{-\lambda(t_{now}-t_{ui})} + 1$] if the cluster [$i$ was updated by the current stream input point (at time t_{now}).

Where $0 \leq \lambda \leq$ represents the decaying factor. Larger values of λ result in a faster decaying of old members of the current cluster, while smaller values represent more contribution of old members in the calculation of the weight of the current cluster.

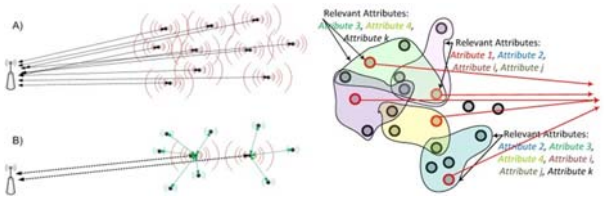

Figure 1. Left: An example of a routing protocol: A) All sensor nodes use a big sending power to send their data to the base station. B) Sensor nodes use lower sending power to send data locally to a local node which aggregates and sends them to the base station. Right: An example how ECLUN [11] performs the grouping of nodes and the selection of representatives additionally according to similarity w.r.t. some attributes

3.4 The Distributed Clustering

In distributed clustering we track each sensor node data locally, process it then combine the results in a central node or a coordinator. The target is to minimize the communication and share the resources. We define the ***distributed clustering*** problem. Let 1, 2,...,d be distributed sites, each site i applies a clustering algorithm A_i on its stream input of data X_i and produces a solution S_i. It is required to perform a global clustering of all X_i; $i = 1, 2,...,d$ input streams distributed over the sites. One efficient solution to do that is to have a central site which collects the union $\mathrm{U}_{i=1}^{d} S_i$ and answers the querying or monitoring requests of the whole input streams. It is also possible that a further clustering algorithm B at the coordinator to be applied on $\mathrm{U}_{i=1}^{d} S_i$. In the distributed k-center clustering problem, we will consider in this paper, the solution of A_i at each site i is $S_i = \{c_{i1}, c_{i2},..., c_{ik}, R\}$ and on the coordinator side, we perform another k-center clustering algorithm over the whole k centers coming from the whole d sites, i.e.,

$\mathrm{U}_{i=1}^{d}\{c_{i1}, c_{i2},..., c_{ik}\}$. When applying incremental clustering algorithms, continuous updates with the new solutions must be sent from the sites to the coordinator. Figure 2 illustrates an example where the coordinator applies another k-center clustering over the centers sent from the sites.

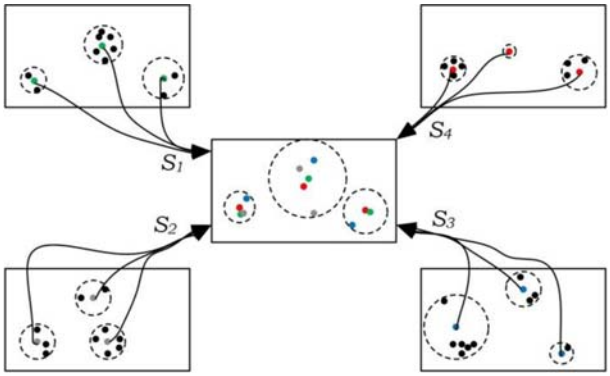

Figure 2. An example of a distributed k-center clustering for k = 3 clusters of data coming from d = 4 sites, the coordinator applies another k-center clustering over the k _ d centers sent from the sites.

3.5 The Problem of Weighted *k*-center Clustering with Outliers

Sensor data are usually noisy. If a clustering algorithm is not outlier-aware the results will become extremely sensitive to the noise. This sensitivity has two effects: **(*a*)** on the clustering quality which appears in Figure 2 for example in site 3 where the cluster on the left has a bigger clustering radius (worse clustering quality) because of not considering one point too far from the center as an outlier. And **(*b*)** on the energy consumption in the distributed model since outliers cause current solutions to be invalid and result in additional updates with the coordinator. Formally in the weighted k-center clustering problem with outliers we group m out of the n input points into the k clusters by dropping maximally $z = m - n$ points. The decision of labeling those z or less points as outliers is done according to their small weights compared to other clusters. Noisy data form less dense clusters that receive updates less frequently, the weight of such clusters will soon decrease.

4. The *SenClu* Algorithm

In this section we present *SenClu*. Each local sensor node receives its input stream through its sensors, produces a weighted k-center clustering solution to it by considering the existence of outliers and sends this solution to the coordinator. The coordinator performs another clustering algorithm to the solutions coming from the sites. A server part of the algorithm manages iteratively assigning coordinators and receiving data from them. Therefore, similar to EDISKCO [10] , the input streams are processed at each node locally, and a global clustering of all sensors data is performed globally in the coordinator. SenClu uses a heap structure h for storing maximally $k + z$ weighted clusters. Each member c_j; $j = 1, 2,..., (k + z)$ in this heap represents a cluster, where: $c_j.\ center$ represents the center of this cluster, $c_j.\ weight$: the weight, and $c_j.\ up\_time$: the last time when c_j was updated. The members in this heap are arranged in a descending order according to $c_j.\ weight$. The top k members represent the clusters, while the rest which could be maximally z represent the outlier clusters.

Algorithm 1 *insert(h,p)*

In *A heap h of current clusters and an input point p from the stream*

Out 0:*if OK, j: if p is a new cluster in position j or*
err:if there is no place to add new cluster p

```
1. for j = 1 to size(h) do
2.   if d(p, c_j.center) ≤ R then
3.     //decay all clusters then and increase the weight of j by one using maintain(h)
4.     maintain(h);
5.     return 0;
6.   end if
7. end for
8. if size(h) < (k + z) then /* there is still a place in the heap to insert p */
9.   c_j.center = p;
10.  c_j.weight = 1;
11.  maintain(h);
12.  return j
13. if c_{k+z}.weight ≤ w_min then /* there is an old cluster replace it */
14.  delete(h, k + z);
15.  //maintain the weights using maintain(h)
16.  maintain(h);
17.  return (the insertion position)
18. return err /* we have to recluster */
```

Algorithm 1.

Arranging the clusters according to the weights needs to be done only once after each reclustering. Once the members are arranged, only the updated cluster needs to be rearranged such that it is in the correct place of the list. All non-updated clusters decay with the same factor together. We define the following functions on h:

(a) *maintain* (h): applied after each reclustering or birth of a new cluster such that for all $1 \le j, q \le k+z$ we have $q > j$ only if $c_q.weight \ge c_j.weight$. Whenever a point is inserted in a cluster, $maintain(h)$ simply performs a decaying step for all clusters weight: $c_i.weight = 2^{-\lambda} \times c_i.weight$ for all $1 \le i \le k+z$, and then in the next step increases only the weight of the cluster where the input point was inserted by 1. The decaying step will leave the order of the heap correct, after the increasing step, one scan step is needed to insert the updated cluster in its correct place in the arranged list. The previous step is performed to avoid the complicated mathematics associated with calculating: $w_i = 2^{-\lambda(t_{now}-t_{ui})} + 1$ in general, which most sensor node processors cannot afford.

(b) *size* (h): returns the number of the members in h which can be any value between 0 and $k+z$.

(c) *get* (h, j): returns the member j from the heap.

(d) *delete* (h, j): deletes the member j from the heap and directly maintains the heap.

(e) *insert* (h, p): inserts an input point j from the stream in j, (see Algorithm 1). It scans the members of h beginning with the high-weighted ones. When a cluster is found where p is not further than R from its center, all the cluster are aged by $2^{-\lambda}$, and only the found cluster's weight is incremented by 1, and p is forgotten. If p was further than R from all available cluster centers and there were less than $k+z$ members in h, then a new cluster is established with p is its center. Otherwise check if the least weighted cluster $k+z$ has less weight than w_{win} (the minimum weight), if yes, delete it and insert the new point in a new cluster and return its position. Otherwise, an error is returned for not having a place to add p.

4.1 On The Node Local Side

Each node i receives an input stream $X(i)$ and runs the *SenCluNode* algorithm (see Algorithm 2) and sends updates to the coordinator with k center outlier-and- weight-aware clustering representation of $X(i)$ in addition to the corresponding radius R_i. Please mind that during the initialization phase (not shown in Algorithm 2 for readability), the node increases the radius without sending updates to the coordinator until n input points are received. Then the running phase of *SenCluNode* starts. *SenCluNode* is explained in details in Algorithm 2.

4.2 On The Coordinator Side

The coordinator side algorithm is explained in SenCluCoordinator (cf. Algorithm 3). Lines 15-20 explain the communication messages between the server and the coordinator for managing the selection of the next coordinator according to the residual energy that each node still possesses. The coordinator keeps a special space for saving summary about the energy consumption of each node i. The total number of centers that were

Algorithm 2 SenCluNode$(X_i, k, z, \lambda, w_{min}, \epsilon)$

In *An input stream* X_i*, num of clusters* k*, num of outlier clusters* z*, decaying factor* λ*, minimum weight* w_{min} *and step size* ϵ

Out *update the coordinator with the new opened clusters' centers and radii*

1. $X_i(t) \rightarrow p, c_1.center = p, c_1.weight = 1, R = R_{min}$
2. **while** there is input stream $X_i(t)$ **do**
3. $X_i(t) \rightarrow p$
4. $fit = insert(p, h)$
5. **if** *next_coordinator* signal from server **then** *//current node becomes coordinator*
6. Save current local centers in C_{local} *// in order not to lose its local solutions*
7. Switch to *SenCluCoordinator*
8. **end if**
9. **if** $fit \neq err$ **and** $0 < fit \leq k$ **then**
10. Send the new center $get(h, fit)$ with a *new_center* signal to the coordinator.
11. **end if**
12. **if** $fit == err$ **then** *// we need to recluster by increasing the radius*
13. Send *radius_increase* request to coordinator
14. **while** there is no reply from coordinator **do**
15. Buffer incoming input points from X_i
16. **end while**
17. $R \leftarrow max\{R(1 + \frac{\epsilon}{2}), R_{global}\}$ *// get the biggest possible next R*
18. **while** $j < size(h)$ **do** *// search for overlapping clusters and merge them*
19. $q = j + 1$;
20. **while** $q < size(h)$ **do**
21. search for the closest j and q to each other;
22. increase the weight of the higher-weighted of them by 1;
23. delete the lower-weighted one;
24. $q++$;
25. **end while**
26. $j++$;
27. **end while**
28. $maintain(h)$;
29. Keep only the the clusters whose weight is at least w_{min} in h;
30. Run this algorithm on buffered points;
31. $C_i = \{c_1.center, c_2.center, \ldots, c_{\hat{k}}.center\}$; *// get the heaviest* $\hat{k} \leq k$
32. Send C_i, R_i as an update to the coordinator;
33. **end if** *// end of the reclustering phase*
34. **end while**

Algorithm 2.

sent from a node *i* and the total number of radius increase requests sent from a node during this phase are saved under $num\_Centers_i$ and $num\_Requests_i$ respectively. These are important for the server to calculate the total energy consumption of each node including the coordinator during this phase. The server sends from time to time consumption update requests to the coordinator which in turn replies to them. The server uses an energy model similar to the one in [10] According to the residual energy of each node and the coordinator, the server makes a decision of changing the current coordinator by sending *chg_coordinator* message to it, with the *id* of the next coordinator.

5. Experiments

We have evaluated the performance of *SenClu* using extensive experiments on both synthetic and real data. As competitors, we have chosen two state-of-the-art algorithms: EDISKCO [10] and Global-PG [6] (we refer to it as PG in the experiments). Both competitors represent single-pass distributed k-center clustering algorithm on the node sides which also applies the furthest point algorithm on the coordinator side. In order to have fair results, we have implemented our suggested node-coordinator-server model also on the Global-PG. We have implemented simulations of the three algorithms in Java.

Algorithm 3 SenCluCoordinator($C_i, R_i, c_{ij}.center$)

In *solutions C_i, R_i, new opened cluster centers, cluster_increase requests from node i*

Out *send ack and R_{global} to nodes, maintain C_{global}, R_{global} and send them to next coordinator*

1. **if** this is **not** the first coordinator **then**
2. Receive C_{global}, R_{global} from last coordinator; // *undertake the solutions*
3. **end if**
4. Broadcast *I_am_coordinator* signal to all nodes;
5. $C_{global} \leftarrow FurthestPoint(C_{global}, C_{local})$; // *as in [8]*
6. **if** $radius\_increase_i$ **do** // *received: radius_increase request from node i*
7. Send to node i an *ack* with R_{global};
8. Receive C_i, R_i from node i; // *new solution from i w.r.t. the new value of R*
9. $C_{global} \leftarrow FurthestPoint(C_{global}, C_i)$; // *update C_{global} with the new solution*
10. $R_{global} \leftarrow max\{R_{global}, R_i\}$; // *update R_{global} with the new solution*
11. **end if**
12. **if** new_center_i **do**// *coordinator received: new centers on node i*
13. $C_{global} \leftarrow FurthestPoint(C_{global}, \{c_{ij}.center\})$
14. **end if**
15. **if** *consumption_update* received from server **then**
16. Send $\{numCenters_i, numRequests_i\}$ for all $i = 1 \ldots m$ nodes during this phase to server
17. **end if**
18. **if** *chg_coordinator* received from server **then**
19. Send C_{global}, R_{global} to next coordinator
20. Switch to *SenCluNode* using R_{global}
21. **end if**

Algorithm 3.

We have chosen one synthetic dataset and two real world datasets; we give a small description of each:

5.1 Synthetic data set: RandomWalk (RW)[1]

A synthetic data set based on the random walk model. The increments between two consecutive values are independent and identically distributed. Each increment: t_{i+1} is produced by randomly adding or subtracting from t_i a uniformly random value from the interval $[1, 10]$. We generated 19 diffierent data sets each for one node, each contains $42{,}000$ measures. Subsequently, to produce a natural outliers effect, we replaced randomly selected values (4.5% of the dataset size) with noise values, uniformly at random in the interval $[min, max]$ of the dataset.

5.2 Real Dataset: I9 Sensor Dataset[1]

We have collected a real data from a sensor network. We deployed TelosB motes in our department area. All motes were programmed to collect temperature samples each seconds and send them directly to a sink connected to a computer. The data was collected for more than days between the 10th and the 23rd of April 2009 and forming measures of each node. The minimum difference between raw measures is . The nodes were not always able to communicate perfectly with the sink due to collisions or loss of signal, this appeared in of the total data. Instead of each measure that did not reach the sink, we introduced a noise data. In a different way of adding outliers to that of **RW**, a uniformly at random value from the interval was selected and then uniformly at a random either added to or deducted from , the resulting value was inserted instead of the lost measure.

Real Dataset: Physiological Sensor Dataset This data was presented in ICML 2004 as a challenge for information extraction from streaming sensor data. The training data set consists of approximately $10{,}000$ hours of this data containing: userID, gender, sessionID, sessionTime, annotation, characteristic 1, characteristic and sensor $[1...9]$ We have extracted the firrst $24{,}000$ readings of sensor2 by userID. We have chosen the data of 12 different userIDs with the same gender, each representing a node. We did not add outliers to this dataset as they naturally exist in such datasets.

We have used the following four criteria to evaluate *SenClu* w.r.t. EDISKCO and the PG algorithm:

(a) Silhouette Coefficient: We use this measure to evaluate the clustering quality on the nodes side. It reflects how appropriate the mapping of data objects is to clusters. It subtracts the average distance of objects to their representative from the average distance of objects to their second closest cluster and then divides the results over the bigger average. When calculating the average of these values for all objects in all clusters, the final value will range from -1 to $+1$. Where -1 will reflect the worst clustering and $+1$ a perfect one. For the streaming case,

[1]dataset is available under http://dme.rwth-aachen.de/SenClu

we have used a sliding window over the stream input and then performed the calculation of the Silhouette coefficient at the end of each window for all the objects within it.

(b) Global Clustering Radius: Another measure to reflect the clustering quality, this time on the coordinator side. In k-center clustering, better clustering uses smaller radius to cover all of the input points.

(c) Maximum Clustering Radius: A third measure that shows also the worst case of the clustering accuracy. It measures the maximum radius of the clusters over all nodes; this value will decide later the global clustering radius (when a reclustering signal is sent to the coordinator).

(d) Energy Consumption: Evaluated through the average energy consumption of the one sensor node in the network in Joule based on the detailed cost model suggested in [10] and the datasheets of TelosB mote, TI MSP430 microcontroller and the CC2420 radio chip in addition to the TinyOS 2.0.2 operating system installed on the motes.

5.1 Experimental Setup and Results

For all experiments, we selected the parameters for *SenClu* and EDISKCO for all datasets as: $k = 15, z = \frac{k}{4} = 4$, $\in = 0.5$. For *SenClu* only we selected: $w_{min} = 0.5$, and λ as: (0.005 in RW dataset) and (0.018 in I9 dataset) and (0.01in Physiological Dataset). For EDISKCO only: the number of the most dense clusters to be sent to the coordinator after each cluster increase: $l = \frac{k}{4} = 4$. The maximum allowed number of input points after which the node can send a solution to the coordinator: $n = 100$, and the maximum allowed number of outliers in total: $o = 10\% n$. For the Global-PG we have selected the number of points collected at the beginning to perform the parallel guessing equals to our $n = 400$ and also $\in = 0.5$. We performed the evaluations on a 3.00 Ghz core Duo 4 GB RAM machine. For all algorithms we set the initial radius as $R_{min}(P) = min_{p, q \in p \neq q} d(p, q)$ for each dataset P separately. Figure 3(a) shows that *SenClu* achieves considerably higher Silhouette coefficient values than both EDISKCO and PG over almost the entire data stream of the RW dataset. This high clustering quality of *SenClu* is due to its novel weighing technique that allow new emerging trends to influence the clustering result, and thus grouped in the correct cluster. Also on the clustering performed on the coordinator side, Figure 3(b) shows that *SenClu* has always smaller global radius than EDISKCO which constitutes a better clustering. PG has considerably worse performance than *SenClu* and EDISKCO on the coordinator side (mind the logarithmic scaling in Figure 3(a)). Also in Figure 4(a) we can see that SenClu has a better clustering performance on the node side than both competitors over the whole data measurements of the I9 Sensor Dataset. Figure 4(b) shows again that the decaying nature and the smart

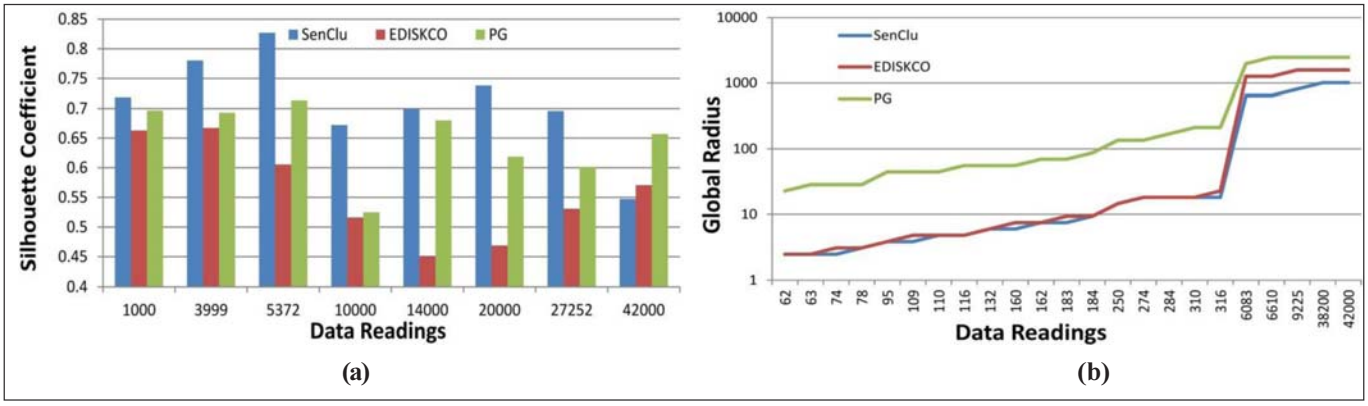

Figure 3. The clustering quality using the Random Walk Synthetic Dataset over different parts of the input stream data (a) the Silhouette Coefficient, (b) The global radius R_{global}

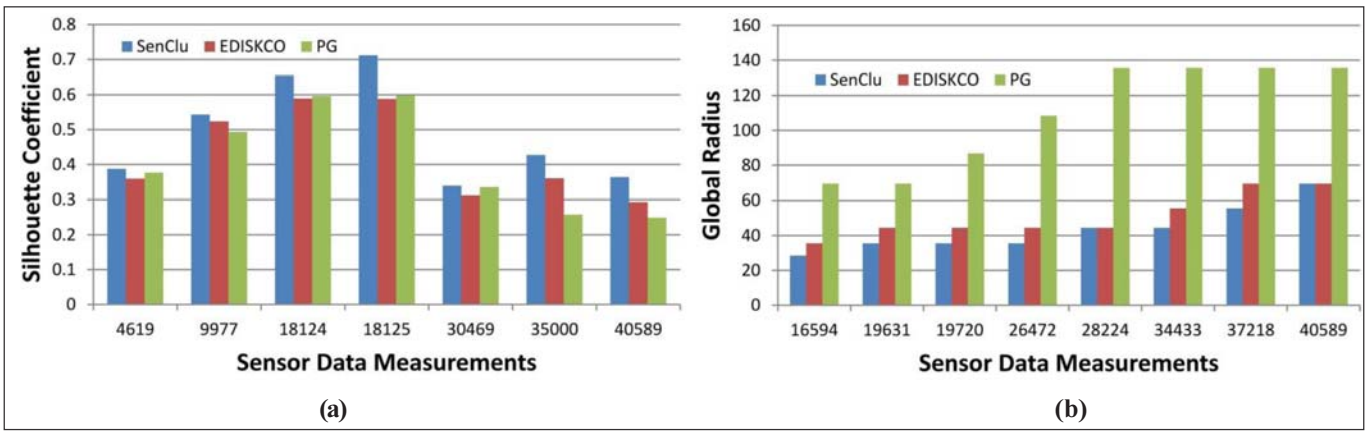

Figure 4 The clustering quality using the I9 Real Sensor Dataset over different parts of the input measurements (a) Silhouette Coefficients, (b) The global radius R_{global}

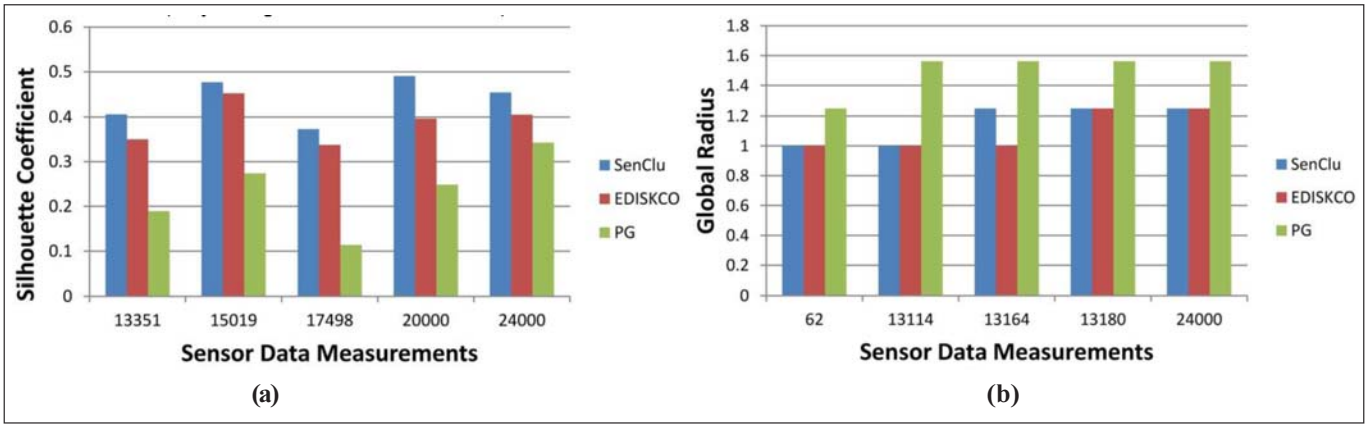

Figure 5. The clustering quality using the Real Physiological Sensor Dataset over different parts of the input stream data (a) Silhouette Coefficients, (b) The global radius R_{global}

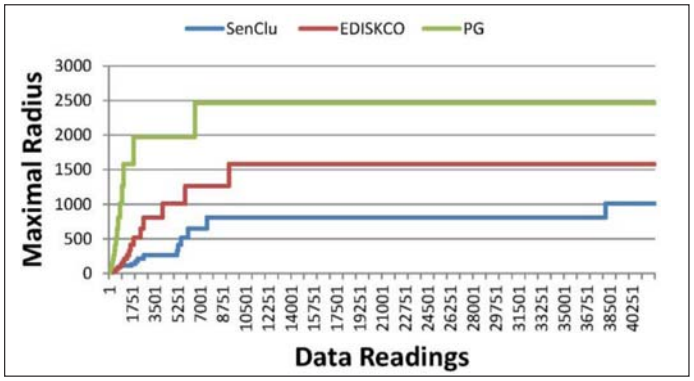

Figure 6. The clustering quality using the Random Walk Synthetic Dataset over different parts of the input stream data illustrated with the maximal radius

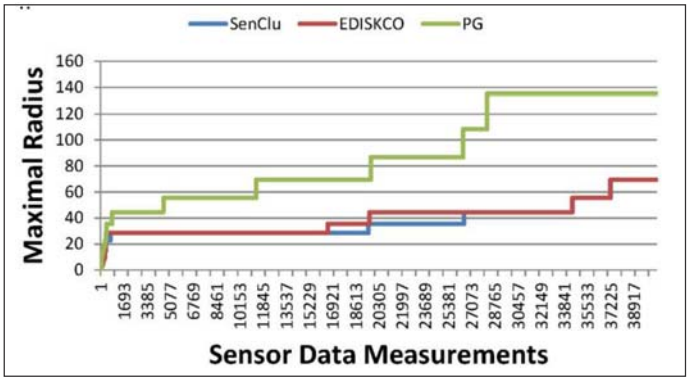

Figure 7. The clustering quality using the I9 Real Sensor Dataset over different parts of the input stream data illustrated with the maximal radius

merging technique that *SenClu* has, result in a smaller radius on the coordinator side and thus better overall clustering results than both PG and EDISKCO. Figure 5(a) shows on another real dataset (Physiological Sensor Dataset) that *SenClu* always has a better clustering quality than both competitors on the node side. Because PG is more sensitive to noise than *SenClu* and EDISKCO, it is performing considerably worse than others on this relatively noisy dataset. Figure 5(b) is showing that on the node side, *SenClu* is having most of the time the same global radius as EDISKCO. Only for a short time, *SenClu* is having a bigger radius than EDISKCO. shows that *SenClu* consumes a bit more energy on average than both EDISKCO and PG when applied on RW dataset. This can be explained by the random nature of the Random Walk dataset that results in different new trends in the data, which *SenClu* tries to follow. This results in multiple updates to the coordinator of new created clusters. This is not the case for EDISKCO and PG where a very lazy update of newly emerging clusters saves some energy while extremely affects the clustering quality (cf. Figure 3(a) and 3(b)). This effect does not appear when using natural real datasets. We can see from Table 1 that on I9 Sensor Dataset, *SenClu* consumes less than twoJoules more than EDISKCO, and absorbs considerably less energy than PG. When using the Physiological Sensor Dataset, *SenClu* consumes less energy than both competitors.

Figure 6, Figure 7 and Figure 8 show the clustering quality

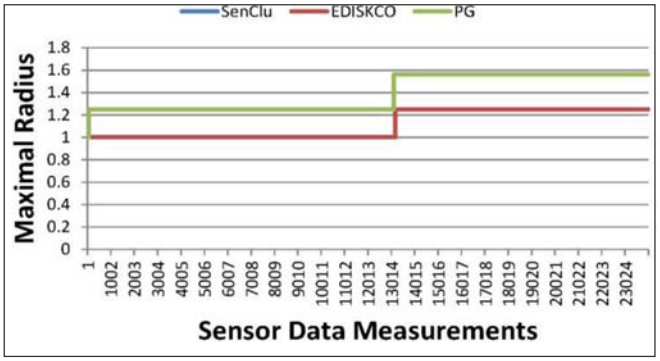

Figure 8. The clustering quality using the Real Physiological Sensor Dataset over different parts of the input stream data illustrated with the maximal radius

of the three algorithms using the maximal radius measure for the RW, I9 and the Physiological datasets respectively. The maximal radius reflects the biggest clustering radius over all nodes. Similar to the global radius; the bigger the value of the maximal radius, the worse the quality of the clustering algorithm. From Figures 6, 7 and 8, one can see that the PG clustering algorithm has considerably worse results than both EDISKCO and *SenClu*. It can be noticed also that for all of datasets *SenClu* has always a better or at least equal clustering quality as EDISKCO.

Dataset	Size	Nodes	SenClu	EDISKCO	PG
RW	4200	19	29805.7	**29776.4**	29792.11
I9-Sensor	40589	19	28770.2	**28768.38**	28792.5
Physio	24000	12	**17074.3**	17074.4	17078.9

Table 1. Average energy consumption in Joule of a single node in the network by the end of each dataset when using SenClu, EDISKCO and PG

6. Conclusions and Future Work

In this work we present our novel energy efficient weighted *k*-center clustering solution. We presented our algorithm: *SenClu* as a single-pass algorithm that immediately detects new trends in the drifting sensor data stream and follows them. The light-weighted decaying technique which we used to enhance the clustering quality, gives lower influence to old data. As sensor data are usually noisy, *SenClu* is also outlier-aware. In thorough experiments on drifting synthetic and real world data sets, we showed that *SenClu* outperforms two state-of-the-art algorithms by producing higher clustering quality and following trends in the stream, while consuming nearly the same amount of energy. In future work, we aim at a further improvement of the clustering quality. One possible way for that is the usage of a new data structure and a dedicated distance function for the weighted -center clustering in the distributed case. Additionally, we would like to discuss the possibility of tracking clusters which are available in some subspace of the data. Some existing techniques already try to achieve that on static data. But it will be definitely interesting to check the possibility of including the streaming case.

7. Acknowledgments

This research was funded in part by the cluster of excellence on Ultra-high speed Mobile Information and Communication (UMIC) of the DFG (German Research Foundation grant EXC 89).

References

[1] Cao, F., Ester, M., Qian W.,and Zhou, A. (2006). Density-based clustering over an evolving data stream with noise. *In SDM'06*, p. 326-337.

[2] Charikar, M., Chekuri, C., Feder, T., Motwani, R. (1997). Incremental clustering and dynamic information retrieval. *In*: Proc. ACM STOC'07, p. 626-635.

[3] Charikar, M., Khuller, S., Mount, D. M., Narasimhan, G. (2001). Algorithms for facility location problems with outliers. *In*: Proc. SODA'01, p. 642-651.

[4] Charikar, M., O'Callaghan, L., Panigrahy, R. (2003). Better streaming algorithms for clustering problems. *In*: Proc. ACM STOC'03, p. 30-39.

[5] Chen, Y., Tu, L. (2007) Density-based clustering for real-time stream data. *In*: Proc. KDD'07, p. 133-142.

[6] Cormode, G., Muthukrishnan, S., Zhuang, W. (2007). Conquering the divide: Continuous clustering of distributed data streams. *In*: Proc. IEEE ICDE'07, p. 1036-1045.

[7] Feder, T., Greene, D. (1988). Optimal algorithms for approximate clustering. *In*: Proc. ACM STOC'88, p. 434-444.

[8] Gonzalez, T. F. (1985). Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38 (2-3) 293-306.

[9] Guha, S. (2009). Tight results for clustering and summarizing data streams. *In*: Proc. ICDT'09, p. 268-275.

[10] Hassani, M., Müller, E., Seidl, T. (2009). EDISKCO: Energy efficient Distributed In-sensor-network *k*-center Clustering with Outliers. *In*: Proc. SensorKDD'09, p. 39-48.

[11] Hassani, M., Müller, E., Spaus, P., Faqolli, A., Palpanas, T., Seidl, T. (2010). Self-organizing energy aware clustering of nodes in sensor networks using relevant attributes. *In*: Proc. SensorKDD'10, p. 87-96.

[12] Hassani, M., Seidl, T. (2011). Towards a mobile health context prediction: Sequential pattern mining in multiple streams. *In*: Proc. of IEEE MDM '11, 2, 55-57.

[13] Heinzelman, W. R., Chandrakasan, A., Balakrishnan, H. (2000). Energy-efficient communication protocol for wireless microsensor networks. *In*: Proc. HICSS'00.

[14] Hochbaum, D., Shmoys, D. (1985). A best possible approximation algorithm for the *k*-centre problem. *Math. of Operations Research*,10, p. 180-184.

[15] Matthew Mccutchen, R., Khuller, S. (2008). Streaming algorithms for *k*-center clustering with outliers and with anonymity. *In*: *Proc. Workshop APPROX / RANDOM'08*, p. 165-178.

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

Author biographies

Marwan Hassani received an equivalence Master degree in Computer Science from RWTH Aachen University in Germany in December 2008. Marwan has received his bachelor degree from Aleppo University, Syria in July 2004. Currently he is a PhD student at RWTH Aachen University. His research interests include data exploration techniques, such as stream data mining, in particular energy-efficient sensor clustering algorithms and advances stream clustering algorithms.

Thomas Seidl is a professor for computer science and head of the data management and data exploration group at RWTH Aachen University, Germany. His research interests include data mining and database technology for multimedia and spatio-temporal databases in engineering, communication and life science applications. Prof. Seidl received his Diplom (MSc) in 1992 from TU Muenchen and his PhD (1997) and venia legendi (2001) from LMU Muenchen.