Clustering distributed sensor data streams using local processing and reduced communication

João Gama[1,4] Pedro Pereira Rodrigues[1,2,3]
Luís Lopes[2,5]

[1] LIAAD - INESC Porto L.A.
Rua de Ceuta, 118 - 6 andar, 4050-190 Porto, Portugal
[2] DCC-FCUP - Faculty of Sciences of the University of Porto
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
[3] SBIM-FMUP - Faculty of Medicine of the University of Porto
Alameda Prof Hernâni Monteiro, 4200-319 Porto, Portugal
[4] FEP - Faculty of Economics of the University of Porto
Rua Dr. Roberto Frias, 4200-464 Porto, Portugal
[5] CRACS - INESC Porto L.A.
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
`jgama@fep.up.pt pprodrigues@fc.up.pt lblopes@dcc.fc.up.pt`

Abstract

Nowadays applications produce infinite streams of data distributed across wide sensor networks. In this work we study the problem of continuously maintain a cluster structure over the data points generated by the entire network. Usual techniques operate by forwarding and concentrating the entire data in a central server, processing it as a multivariate stream. In this paper, we propose *DGClust*, a new distributed algorithm which reduces both the dimensionality and the communication burdens, by allowing each local sensor to keep an online discretization of its data stream, which operates with constant update time and (almost) fixed space. Each new data point triggers a cell in this univariate grid, reflecting the current state of the data stream at the local site. Whenever a local site changes its state, it notifies the central server about the new state it is in. This way, at each point in time, the central site has the global multivariate state of the entire network. To avoid monitoring all possible states, which is exponential in the number of sensors, the central site keeps a small list of counters of the most frequent global states. Finally, a simple adaptive partitional clustering algorithm is applied to the frequent states central points in order to provide an anytime definition of the clusters centers. The approach is evaluated in the context of distributed sensor networks, focusing on three outcomes: loss to real centroids, communication prevention, and processing reduction. The experimental work on synthetic data supports our proposal, presenting robustness to a high number of sensors, and the application to real data from physiological sensors exposes the aforementioned advantages of the system.

Keywords: online adaptive clustering, distributed data streams, sensor networks, incremental discretization, frequent items monitoring.

1 Introduction

Data gathering and analysis have become ubiquitous, in the sense that our world is evolving into a setting where all devices, as small as they may be, will be able to include sensing and processing ability. Nowadays applications produce infinite streams of data distributed across wide

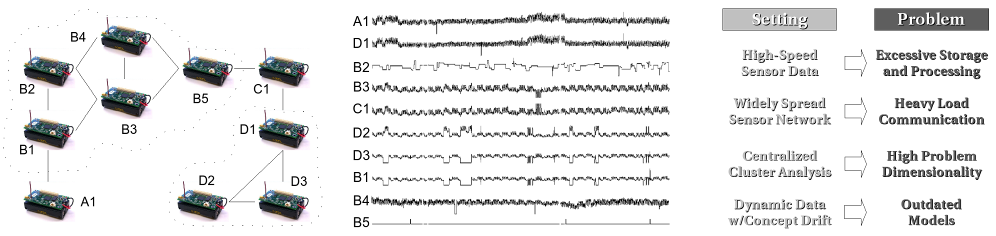

B4
B2
B1
A1
B3
B5
C1
D1
D2
D3
A1
D1
B2
B3
C1
D2
D3
B1
B4
B5
Setting
Problem
High-Speed Sensor Data
Excessive Storage and Processing
Widely Spread Sensor Network
Heavy Load Communication
Centralized Cluster Analysis
High Problem Dimensionality
Dynamic Data w/Concept Drift
Outdated Models

Figure 1: Example of a mote sensor network and a plot of the data each univariate sensor is producing over time. The aim is to find dense regions of the data space.

sensor networks (see example on figure 1). The aim of the analysis addressed in this work is to continuously maintain a cluster structure over the data points generated by the entire network, as clustering of sensor data gives information about dense regions of the sensor data space, yielding informative support to everyday decisions [27].

Sensors are usually small, low-cost devices capable of sensing some attribute and of communicating with other sensors. On top of this, the networks created by today's setting can easily include thousands of sensors, each one being capable of measuring, analysing and transmitting data series, which also implies resource restrictions which narrow the possibilities for high-load computation, while operating under a limited bandwidth. Stream data has tendentiously infinite length, while ubiquitous data has tendentiously infinite width. Therefore, data should also be processed by sensor networks in a distributed way. Usual techniques operate by forwarding and concentrating the entire data in a central server, processing it as a multivariate stream.

In this work we study the problem of continuously maintaining a cluster structure over the data points generated by the entire network, where each sensor produces a univariate stream of data. The main issues of the addressed setting are summarised in figure 1. The main problem in applying clustering to data streams is that systems should consider data evolution, being able to compress old information and adapt to new concepts [25]. In this paper, we propose a new distributed algorithm which reduces both the dimensionality and the communication burdens.

In the next section our method is explained, with relevant analysis of the overall processing. Section 3 focus on the advantages presented by our proposal in terms of memory and communication resources, especially important in distributed sensor networks. Validation of the system and experimental results on real-world scenarios are presented in section 4. Section 5 presents a quick discussion on the advantages extracted from the analysis of empirical results. In section 6, a review of previous related work is presented, whereas the final section concludes the paper, including foreseen future work.

2 *DGClust* – Distributed Grid Clustering

This work addresses the problem of continuously monitoring clustering structures of distributed sensor data. The main intuition supporting our research is to reduce dimensionality - by monitoring and clustering only frequent states - and communication - by applying online sensor data discretization and controlled transmission to a central server.

In this paper we propose *DGClust*, a distributed grid clustering system for sensor data streams, which operates with the following procedure. Each local sensor receives data from a given source, producing a univariate data stream, which is potentially infinite. Therefore, each sensor's data is processed locally, being incrementally discretized into a univariate adaptive grid. Each new data point triggers a cell in this grid, reflecting the current state of the data stream at the local site. Whenever a local site changes its state, that is, the triggered cell changes, the new state is communicated to a central site. Furthermore, the central site keeps the global state of the entire network where each local site's state is the cell number of each local site's grid. Nowadays, sensor networks may include thousands of sensors. This scenario yields an exponential number of cell combinations to be monitored by the central site. However, it is expected that only a

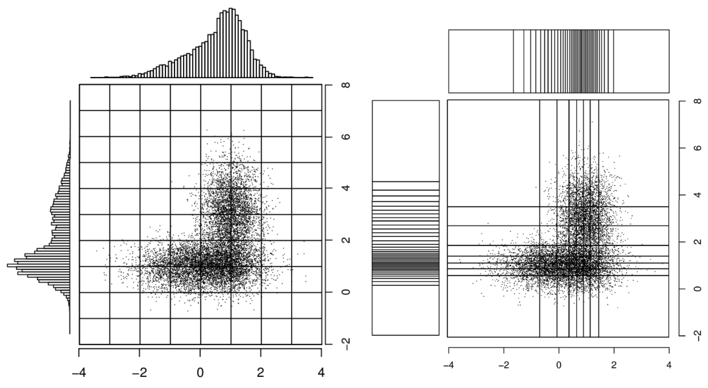
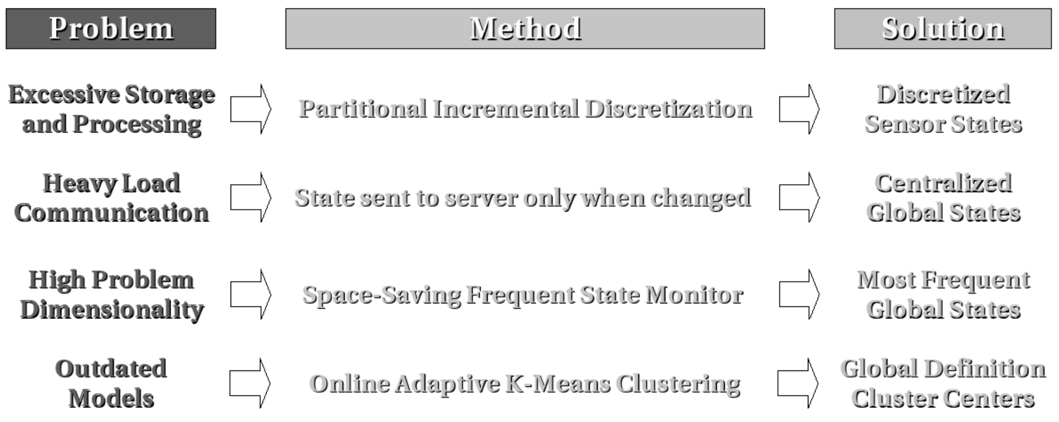


Figure 2: Example of a 2-sensor network, with data distribution being sketched for each sensor: although the number of cells to monitor increases exponentially with the number of sensors (dimensions), and unless data is uniformly distributed in all dimensions (extremely unlikely in usual data) the cells where most of the data lie are much less. Left plot presents an equal-width discretization, while right plot presents an equal-frequency discretization.



Problem
Method
Solution
Excessive Storage and Processing
Partitional Incremental Discretization
Discretized Sensor States
Heavy Load Communication
State sent to server only when changed
Centralized Global States
High Problem Dimensionality
Space-Saving Frequent State Monitor
Most Frequent Global States
Outdated Models
Online Adaptive K-Means Clustering
Global Definition Cluster Centers

Figure 3: DGClust: Compact representation of streams produced in distributed sensors using online discretization; centralized monitoring of the global state of the network, with lightweight maintenance of frequent states; and fast and adaptable clustering of frequent states to estimate actual clusters centers.

small number of this combinations are frequently triggered by the whole network, as observed in the example sketched in figure 2. Thus, parallel to the aggregation, the central site keeps a small list of counters of the most frequent global states. Finally, the current clustering definition is defined and maintained by a simple adaptive partitional clustering algorithm applied on the frequent states central points. Figure 3 summarises the main approach and the outcome of each step of the algorithm.

2.1 Notation and Formal Setup

The goal is to find k cluster centers of the data produced by a network of d local sensors. Let $X = \{X_1, X_2, ..., X_d\}$ be the set of d univariate data streams, each of which is produced by one sensor in one local site. Each local site i keeps a two-layered discretization of the univariate stream, with p_i intervals in the first layer and w_i intervals in the second layer, where $k < w_i << p_i$ but $w_i \in O(k)$. At each time t, each local sensor produces a value $X_i(t)$ and defines its local discretized state $s_i(t)$, drawn from the set of possible states S_i, the unit cells in the univariate grid ($|S_i| = w_i$).

If no value is read, or $s_i(t) = s_i(t-1)$, no information is sent to the central site.

The central site monitors the global state of the network at time t, by combining each local discretized state $s(t) = \langle s_1(t), s_2(t), ..., s_d(t) \rangle$. Each $s(t)$ is drawn from a finite set of cell combinations $E = \{e_1, e_2, ..., e_{|E|}\}$, with $|E| = \prod_{i=1}^{d} w_i$. Given the exponential size of E, the central site monitors only a subset F of the top-m most frequent elements of E, with $k < |F| << |E|$. Relevant focus is given to size requirements, as $|E| \in O(k^d)$ but $|F| \in O(dk^{\beta})$, with small β. Finally, the top-m frequent states central points are used in an online adaptive partitional clustering algorithm, which defines the current k cluster centers, being afterwards continuously adapted.

2.2 Local Adaptive Grid

Discretization of continuous attributes is an important task for certain types of machine learning algorithms. Although discretization is a well-known topic in data analysis and machine learning, most of the works refer to a batch discretization where all the examples are available for discretization. Few works refer to incremental discretization. However, it is commonly seen as an essential tool for high-speed processing of data streams with limited memory resources [15]. For example, grid clustering algorithms operate on discrete cells to define dense regions of points [30]. In our approach, the main reason to perform discretization of each univariate data stream in sensor networks is to considerably reduce the communication with the central server, which tries to capture the most frequent regions of the entire input space.

2.2.1 Partitional Incremental Discretization

In our approach, we apply incremental discretization at each sensor univariate stream X_i using the *Partition Incremental Discretization* (*PiD*) algorithm [14], which consists of two layers. The first layer simplifies and summarizes the data, while the second layer constructs the final grid. Within the scope of this work, we consider only equal-width discretization. Although equal-frequency discretization seems better, it implies heavier computation and communication, and its benefits fade out with the frequent state monitoring step.

The first layer is initialized based on the number of intervals p_i (that should be much larger than the desired final number of intervals w_i) and the range of the variable. The range of the variable is only indicative, as it is used to define the intervals in the first layer using a equal-width strategy, but it is not rigid as layer-one bins can be split if necessary, monitoring the actual range of the values. Each time a new value $X_i(t)$ is read by the sensor i, we increment the counter in the corresponding interval.

After a given number of examples, the second layer can be defined. We consider a split operator to be triggered in a given layer-one bin as soon as the number of hits in it is above a user-defined threshold α (a percentage of the total number of points considered so far). This split operator generates new intervals in layer one, adapting the initial intervals with the actual distribution of values. This is especially important to correct badly initialized range values and to adapt to slight changes in the distribution of values. The second layer merges the set of intervals defined by the first layer, and defines the final univariate discretization grid $\langle b_1, b_2, ..., b_{w_i} \rangle$, with size $w_i > k$ linear to the number of final clusters to find.

The process of updating the first layer works online, doing a single scan over the data stream, hence being able to process infinite sequences of data, processing each example in constant time and (almost) constant space. The update process of the second layer works online along with the first layer. For each new example $X_i(t)$, the system increments the counter in the second-layer cell where the triggered first-layer cell is included, defining the discretized state $s_i(t)$. The grid this way defined represents an approximated histogram of the variable produced at the local site. Algorithm 1 presents the adaptive discretization procedure executed at each local site.

Algorithm 1 LocalAdaptiveGrid(X_i, α)

Input: univariate stream X_i; and threshold value α;
Output: *void* (sends discretized states s_i to central site);

```
1: let $n \leftarrow 0$
2: while $X_i(t) \leftarrow read\_sensor(i)$ do
3:     $n \leftarrow n + 1$
4:     let $c$ be the layer-one cell triggered by $X_i(t)$
5:     $count_c \leftarrow count_c + 1$
6:     if $count_c/n > \alpha$ then
7:         split cell $c$, dividing $count_c$ evenly by the two cells
8:         update second-layer intervals $\langle b_1, b_2, ..., b_{w_i}\rangle$
9:         send $\langle b_1, b_2, ..., b_{w_i}\rangle$ to central site
10:    end if
11:    let $b$ be the layer-two cell triggered by $c$
12:    $count_b \leftarrow count_b + 1$
13:    send $s_i \leftarrow \langle t, b\rangle$ to the central site
14: end while
```

2.3 Centralized Frequent State Monitoring

In this work we consider synchronous processing of sensor data. The global state is updated at each time stamp as a combination of each local site's state, where each value is the cell number of each local site's grid, $s(t) = \langle s_1(t), s_2(t), ..., s_d(t)\rangle$. If in that period no information arrives from a given local site i, the central site assumes that site i stays in the previous local state ($s_i(t) \leftarrow s_i(t-1)$). Given that common sensor networks usually imply asynchronous communication, this problem could be easily coped with using a time frame where the central server could wait for data from the nodes. However, older values that could arrive after this time frame could only be considered for the next time frame, and only if this is globally acceptable in the domain being studied.

A major issue with our setting is that the number $|E|$ of cell combinations to be monitored by the central site is exponential to the number of sensors, $|E| = O(w^d)$. However, only a small number of this combinations represent states which are frequently visited by the whole network. This way the central site keeps a small list, F, of counters of the most frequent global states, whose central points will afterwards be used in the final clustering algorithm, with $|F| = O(dk^\beta)$, for small β.

2.3.1 Space-Saving Top-m Elements Monitoring

Each seen global state $e \in E$ is a frequent element f_r whose counter $count_r$ currently estimates that it is the r^{th} most frequent state. The system applies the *Space-Saving* algorithm [21], to monitor only the top-m elements. Basically, if we observe a state $s(t)$ that is monitored in rank r ($f_r = s(t)$), we just increment its counter, $count_r$. If $s(t)$ is not monitored, we replace f_m, the element that currently has the least estimated hits, $count_m$, with $s(t)$, and increment $count_m$. For each monitored element f_i, we keep track of its over-estimation, ε_i, resulting from the initialization of its counter when it was inserted into the list. That is, when starting to monitor f_i, set ε_i to the value of the evicted counter. An element f_i is *guaranteed* to be among the top-m elements if its guaranteed number of hits, $count_i - \varepsilon_i$, exceeds $count_{m+1}$ (in the following referred as $count_e$ as it is implemented by storing the count of the last evicted state). Algorithm 2 presents the necessary procedure to keep the list of most frequent states.

The authors in [21] report that, even if it is not possible to guarantee top-m elements, the algorithm can guarantee top-m' elements, with $m' \approx m$. Hence, suitable values for m should be considered. Furthermore, due to errors in estimating the frequencies of the elements, the order of the elements in the data structure might not reflect their exact ranks. Thus, when performing clustering on the top-m elements, we should be careful not to directly weight each point by its rank.

Algorithm 2 SpaceSavingUpdate(F, m, $s(t)$) – adapted from [21]

```
Input: set $F$ of frequent states; maximum number $m$ of states to monitor; current global state
    $s(t) \leftarrow \langle s_1(t), s_2(t), ..., s_d(t) \rangle$;
Output: updated set $F$ of most frequent global states; updated number of states $l$; updated count
    $count_r$ and overestimation $varepsilon_r$ for current state; count of the last evicted state $count_e$
    for guarantee assessment;
 1: $l \leftarrow |F|$ (the current number of monitored states)
 2: $r \leftarrow i : f_i == s(t), 1 \leq i \leq l$
 3: if $\neg \exists r$ then
 4:    if $l < m$ then
 5:       $f_{l+1} \leftarrow s(t)$
 6:       $\varepsilon_{l+1} \leftarrow count_{l+1} \leftarrow 0$
 7:       $l \leftarrow r \leftarrow l + 1$
 8:    else
 9:       $count_e \leftarrow \varepsilon_m \leftarrow count_m$
10:       $f_m \leftarrow s(t)$
11:       $r \leftarrow m$
12:    end if
13: end if
14: $count_r \leftarrow count_r + 1$
15: update ranks, moving up $\langle f_r, count_r, \varepsilon_r \rangle$ while $count_r > count_{r-1}$
16: return $\langle F, l, count_r, \varepsilon_r, count_e \rangle$
```

The goal of this strategy is to monitor top-m states, using only the *guaranteed* top-m elements as points for the final clustering algorithm.

One important characteristic of this algorithm is that it tends to give more importance to recent examples, enhancing the adaptation of the system to data evolution. This is achieved by assigning to a new state entering the top-m list one plus the count of hits of the evicted state. Hence, even if this is the first time this state has been seen in the data, it will be at least as important to the system as the one being discarded.

2.4 Centralized Online Clustering

The goal of *DGClust* is to find and continuously keep a cluster definition, reporting the k cluster centers. Each frequent state f_i represents a multivariate point, defined by the central points of the corresponding unit cells s_i for each local site X_i. As soon as the central site has a top-m set of states, with $m > k$, a simple partitional algorithm can start, applied to the most frequent states.

2.4.1 Initial Centers

In the general task of finding k centers given m points, there are two major objectives: minimize the *radius* (maximum distance between a point and its closest cluster center) or minimize the *diameter* (maximum distance between two points assigned to the same cluster) [8]. The *Furthest Point* algorithm [17] gives a guaranteed 2-approximation for both the *radius* and *diameter* measures. It begins by picking an arbitrary point as the first center, c_1, then finding the remaining centers c_i iteratively as the point that maximizes its distance from the previously chosen centers $\{c_1, ..., c_{i-1}\}$. After k iterations, one can show that the chosen centers $\{c_1, c_2, ..., c_k\}$ represent a factor 2 approximation to the optimal clustering [17]. See [8] for a proof. This strategy gives a good initialization of the cluster centers, computed by finding the center k_i of each cluster after attracting remaining points to the closest center c_i. This algorithm is applied as soon as the system finds a set of $m' > k$ guaranteed top-m states.

2.4.2 Continuous Adaptive Clustering

It is known that a single iteration is not enough to converge to the actual centers in simple *k-means* strategies. Hence we consider two different states on the overall system operation: *converged* and *non-converged*. At every new state $s(t)$ that is gathered by the central site, if the system has not yet converged, it adapts the clusters centers using the m' guaranteed top-m states.

If the system has already converged, two different scenarios may occur. If the current state is being monitored as one of the m' top-m states, then the set of points actually used in the final clustering is the same, so the clustering centers remain the same. No update is performed. However, if the current state has just become *guaranteed* top-m, then the clusters may have change so we move into a non-converged state of the system, updating the cluster centers. Another scenario where the clusters centers require adaptation is when one or more local sites transmit their new grid intervals, which are used to define the central points of each state. In this case we also update and move to non-converged state.

A different scenario is created when a new state enters the top-m, replacing the least frequent one. In this case, some of the previously *guaranteed* top-m may loose their *guarantee*. However, if the list of frequent items is small (imposed by resources restrictions) this will happen very frequently so we disregard this scenarios to prevent excessive computation when cluster centers have already converged. Future work will focus on these scenarios for concept drift and cluster evolution purposes.

In scenarios where clusters centers adaptation is needed, our system updates the clustering definition by applying a single iteration of point-to-cluster assignment and cluster centers computation. This process (sketched in figure 4) assures a smooth evolution of the cluster centers, while it nevertheless adapts them to the most recent data, as old data points tend to be less frequent.

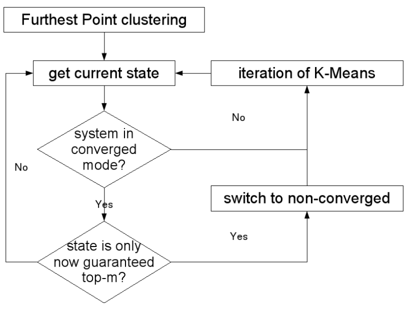

Furthest Point clustering
get current state
iteration of K-Means
No
system in converged mode?
No
Yes
switch to non-converged
Yes
state is only now guaranteed top-m?

Figure 4: Online partitional clustering of most frequent states.

Figure 5 presents an example of a final grid, frequent cells and cluster centers for a specific case with $d = 2$, $k = 5$, for different values of w and m. The flexibility of the system is exposed, as different parameter values yield different levels of results. Moreover, the continuous update keeps track of the most frequent cells keeping the gathered centers within acceptable bounds. A good characteristic of the system is this ability to adapt to resource restricted environments: system granularity can be defined given the resources available in the network's processing sites. Algorithm 3 presents the central adaptive procedure executed at the server site. The algorithm for ClusterCentersUpdate(K, F) is omitted for simplicity and space saving.

3 Algorithm Analysis

Although working altogether in the distributed stream paradigm, there are three different levels of process that we should inspect in our proposal, as they may introduce both complexity and error. First, each univariate data stream is discretized, with only the discretized state being forwarded to the central site. At this point, the granularity of each sensor's grid will directly influence the error in that dimension. Since the construction of the second layer is directly restricted to the intervals

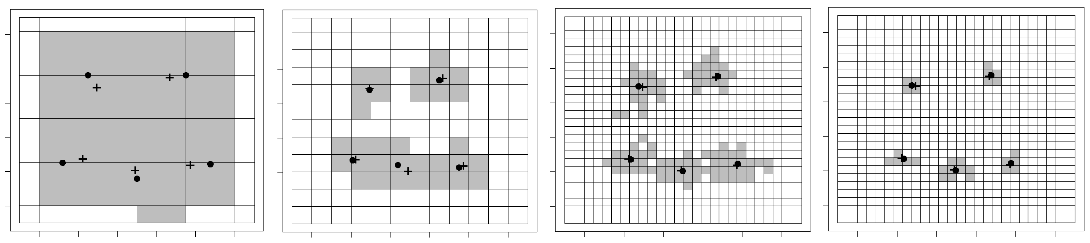


Figure 5: Example of final definition for 2 sensors data, with 5 clusters. Each coordinate shows the actual grid for each sensor, with top-m frequent states (shaded cells), gathered (circles) and real (crosses) centers, run with: $(w = 6, m = 20)$, $(w = 12, m = 60)$, $(w = 24, m = 180)$, and $(w = 24, m = 180)$ presenting only *guaranteed* top-m.

Algorithm 3 CentralAdaptiveProcessing(L, k, m)

```
Input: list of local sites $L = \{l_1, l_2, ..., l_d\}$; number of clusters $k$; and frequent states $m$;
Output: set $K$ of $k$ cluster centers;
 1: $F \leftarrow \{\}$ (set of frequent global states)
 2: $K \leftarrow \{\}$ (set of $k$ cluster centers)
 3: $conv \leftarrow false$ (are the centers stable?)
 4: for each timestamp $t$ do
 5:   for each site $l_i \in L$ do
 6:     if $s_i(t)$ has not been received then $s_i(t) \leftarrow s_i(t-1)$
 7:   end for
 8:   $s(t) \leftarrow \langle s_1(t), s_2(t), ..., s_d(t) \rangle$
 9:   $\left\langle F, l, count_{s(t)}, \varepsilon_{s(t)}, count_e \right\rangle \leftarrow$ SpaceSavingUpdate($F$, $m$, $s(t)$) (Algorithm 2)
10:   if $l > k$ and $K = \{\}$ then
11:     $K \leftarrow$ FurthestPoint ($F$, $k$) (as in [17])
12:   else
13:     if not $conv$ or $count_{s(t)} - \varepsilon_{s(t)} = count_e + 1$ then
14:       $\langle K, conv \rangle \leftarrow$ ClusterCentersUpdate($K$, $F$)
15:     end if
16:   end if
17: end for
18: return K
```

defined in layer one, the final histograms will also be an approximation of the exact histograms that would be defined directly if all data was considered. Nevertheless, with this two-layer strategy the update of the final grid is straightforward. The layer-two intervals just need to be recomputed when the split operator in layer one is triggered. Moreover, the number of intervals in the second layer can be adjusted individually for each sensor, in order to address different needs of data granularity and resources requirements, usually present in current real-world applications [28].

In this proposal we address univariate sensor readings. The data stream model we consider in sensor networks assumes that a sensor value represents its state in a given moment in time. If the readings of a local sensor fall consecutively in the same layer-two interval, no sound information would be given to the central site. Thus, local sites only centralize information when a new value triggers an interval different from the previously sent to the central server. The central site only monitors the top-m most frequent global states, disregarding infrequent states which could influence the final clusters. Finally, the system performs partitional clustering over the m' guaranteed top-m frequent states which is a sample of the actual states, being biased to dense cells. Moreover, although the furthest point algorithm may give guarantees on the initial centers for the clustering of the frequent states, the adaptive update is biased towards small changes in the concept generating the data streams.

3.1 Time and Space

Each sensor X_i produces a univariate adaptive grid. This process uses the *PiD* algorithm which, after the initial definition of the two layers based on n_i examples, in $O(n_i \log p_i)$ time and $O(p_i)$ space, is continuously updated in constant $O(\log p_i)$ time and (almost) constant space. Since this is done parallel across the network, the time complexity of the discretization of one example in the entire network is $O(\log p)$ where $p = \max(p_i)$, $\forall i \in \{1, 2, ..., d\}$. The central site aggregates the state of each of the d local sites. The focus is on monitoring the top-m frequent global states, which are kept in $O(md)$ space (the actual m frequent states) and continuously updated in $O(m)$ time (linear search for the current state). The initial clustering of frequent states, and its subsequent adaptation is made in $O(km)$ time.

3.2 Communication

Data communication occurs only in one direction, between the local sites and the central site. All queries are answered by the central server. Also, this communication does not include sending the original data, rather informing the central server of the current discrete state of the univariate stream of each local site. This feature of only communicating the state when and if it has changed reduces the network's communication requirements. The main reason of this is that, in usual sensor data, sensor readings tend to be highly correlated with previously read value [26], hence tending to stay in the same discretized state. In the worst case scenario, where all d local sites need to communicate their state to the central site, the system processes d messages of one discrete value. However, every time the local site X_i changes its univariate grid, the central site must be informed on the change so that it can correctly compute the points used to find the cluster centers, which imply sending w_i values. In the worst case scenario, the central site may have to receive $O(wd)$ data, where $w = \max(w_i)$.

4 Experimental Evaluation

Evaluation of streaming algorithms is a *hot-topic* in research as no standard exists for evaluating models over streams of data [16]. Hence the difficulty to define exact evaluation processes. Nevertheless, we conducted a series of experiments on synthetic scenarios, to assess the quality of our proposal. All synthetic scenarios were generated by applying the data generator used in [12], considering each dimension separated across sensors, in univariate streams.

The global view of the network scenarios is created by mixtures of k spherical Gaussians, with means μ_i in the unit hypercube. Each scenario was generated according to three parameters: dimensionality d of the network, the number of mixture components k, and the standard deviation of each sensor stream in each component σ. Each scenario (d, k, σ) is created with 100000 examples. Figure 5 shows an example of the final grid obtained for $d = 2$ and different parameter values. Given the scope of this validation, the system's quality is measured by assigning each of the found cluster centers to the closest real cluster center, using a greedy strategy. The *loss* of the chosen assignment is given by

$$L_K = \sum_{i=1}^{k} \sum_{j=1}^{d} (\hat{c}_{ij} - c_{ij})^2 \qquad (1)$$

where $\hat{c}_{ij}$ and c_{ij} are the gathered and real values, respectively, for center i in dimension j. Besides loss, evaluation in the following sections is also done with respect to two other main outcomes: number of values communicated to the server, to assess benefits in communication ratios; and number of clustering updates performed by the central server, to assess the benefits of the system in terms of computation reduction. Studied parameters are the granularity of the univariate adaptive grid, $w_i, \forall i \in \{1, 2, ..., d\}$, and the number of frequent states to monitor, m. We fixed $p_i = 1000, \forall i \in \{1, 2, ..., d\}$. Table 1 presents the studied parameters and the domain in which each of them was studied. In this section, we present results from first sensitivity analysis experiments, inter-dependence analysis of parameters, and application to real data from physiological sensors.

Table 1: Parameter description and corresponding values considered in each section of the experimental evaluation: experiments with synthetic data for evaluation and sensitivity analysis; study on possible parameter dependencies; and real data from physiological sensors.

Parameter		Synthetic Data: Sensitivity Analysis	Synthetic Data: Parameter Dependencies	PDMC Data
Scenario				
d	Number of dimensions/sensors	$d \in \{2, 4, 8, 16, 32, 64, 128\}$	$d \in \{2, 3, 4, 5\}$	–
σ	Standard deviation of data distribution	$\sigma = .05$	$\sigma = .1$	–
k	Number of clusters	$k = 3$	$k \in \{3, 4, 5\}$	$k \in \{2, 3, 4, 5\}$
Local Grid Granularity				
p	Number of bins in each sensor's first layer	$p = 1000$	$p = 1000$	$p = 1000$
α	Threshold for splitting each sensor's first layer	$\alpha = .05$	$\alpha = .05$	$\alpha = .05$
w	Number of bins in each sensor's second layer	$w \in \{5, 7, 9, 11, 13, 15, 17, 19, 21\}$	$w = \omega k + 1$	$w = \omega k + 1$
ω	How much should w be influenced by k?	–	$\omega \in \{1, 2, 3, 4, 5\}$	$\omega \in \{2, 4, 6, 8, 10\}$
Frequent States Monitoring				
m	Number of frequent states to monitor	$m \in \{6, 9, 12, 15$ $18, 21, 24, 27, 30, 33, 36, 39, 42, 45\}$	$m = \phi w d \mid m = \frac{dw^2}{\gamma}$	$m = \frac{dw^2}{\gamma}$
ϕ	How much should m be influenced by d and w	–	$\phi \in \{1, 2, 3, 4, 5\}$	–
γ	How much should m be influenced by d and w^2	–	$\gamma \in \{10, 8, 6\}$	$\gamma \in \{10, 8, 6, 4, 2\}$

4.1 Evaluation on Synthetic Data and Parameter Sensitivity

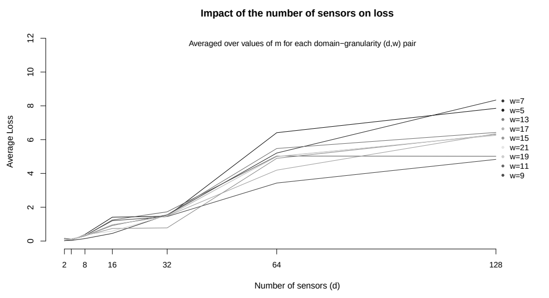
For a first evaluation, we have created a set of scenarios based on $k = 3$ clusters, ranging dimensionality in $d \in \{2, 4, 8, 16, 32, 64, 128\}$ in order to inspect the impact of the number of sensors in the quality of the results. For each scenario, 10 different datasets were created, using $\sigma = .05$, and the system's sensitivity to parameters was evaluated: the univariate grid granularity ranging in $w \in \{5, 7, 9, 11, 13, 15, 17, 19, 21\}$ and the number of states to monitor raging in $m \in \{6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45\}$. The main rationale behind the choice of values for these parameters is that we are addressing a scenario with $k = 3$ clusters. This way, it seems empirically right to have an odd number of cells in each sensor's grid, and a multiple of 3 states to monitor (increasing the average number of states monitored per cluster). The complete set of plots presenting the results for all the combination of parameters is included in Appendix A. There, the reader can find, for each scenario and combination of parameters, an estimate of the mean value for each outcome, with the corresponding 95% confidence interval. Here, we shall present an overview discussion of those results, and focus on some precise results which are relevant to the presentation, such as inspecting the quality of the system with respect to loss (to real centroids), communication (from sensors to the server) and processing (of clustering updates).

4.1.1 Loss to Real Centroids

The impact of the granularity on loss seems evident. If you increase w in low values (slightly above k) there is a reduction in loss. However, this benefit fades for higher values of w, indicating that there is a point where increasing w no longer improves the final result. The same observation can be made for the number of states to monitor (m) as increasing them above a given value revealed small (or no) improvement in the resulting structure. To evaluate the sensitivity of the system to the number of sensors, we analysed the average result for a given value of granularity (w), averaged over all values of m (as loss seemed to be only lightly dependent on this factor). In Figure 6 we plot these averaged results on a dimensionality scale, with an exponential increase of the number of sensors. We note a slight increase in loss as the number of sensors increase (left plot). However, the plot on the right indicates that this increase might be mostly explained by the additive form of the loss function (Equation 1), as normalizing the loss by the number of sensors produced results with no clear trend.

4.1.2 Communication to the Central Server

To evaluate the communication requirements, we monitor the amount of transmitted values as a percentage of the total number of examples times the total number of sensors. We should



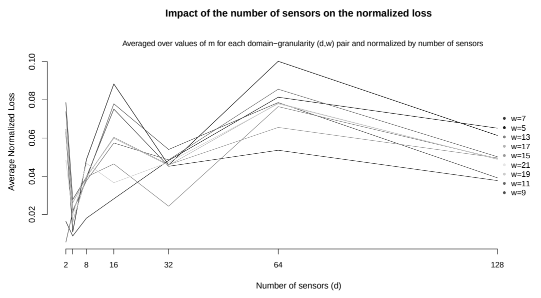
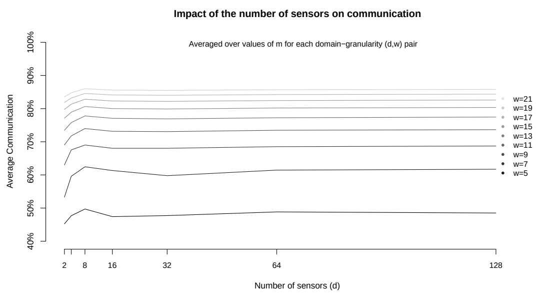


Figure 6: Impact of the number of sensors (from $d = 2$ to $d = 128$) on loss, for each granularity (from $w = 5$ to $w = 21$), averaged over all values of m (from $m = 6$ to $m = 45$). Right plot presents the same results but considering a normalized version of the loss where results are normalized with respect to the number of sensors.



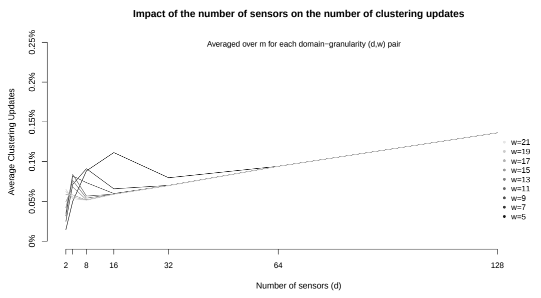


Figure 7: Impact of the number of sensors (from $d = 2$ to $d = 128$) on communication (left plot, in % of examples $\times$ sensors) and processing clustering updates (right plot, in % of examples), for each granularity (from $w = 5$ to $w = 21$), averaged over all values of m (from $m = 6$ to $m = 45$).

stress that, since data is generated at random from Gaussian distributions, the amount of state switching should be much higher than what could be expected from real sensors, which usually produce highly-auto-correlated data. For this reason, the reduction in communication might not be as strong as expected. Nonetheless, the transmission of values is highly dependent on the granularity, since as more states are possible at each sensor, more state switching should occur, hence more transmission is required. On the other side, this outcome does not depend on the number of states to monitor at the central server, as it is only defined by the number of times a sensor sends data to the central server. We can also note that with higher number of sensors the confidence interval for the mean communication narrows, revealing an interesting effect of being easier to estimate communication in wider networks. To evaluate the sensitivity of the system to the number of sensors, we analysed the average result for a given value of granularity (w), averaged over all values of m (as communication is not dependent on this factor). On the left plot of Figure 7 we present these averaged results on the same dimensionality scale, with an exponential increase of the number of sensors. The most relevant fact that rises from these results is that the amount of communication does not depend on the number of sensors. This way, the benefits of reduced transmission rates are extensible to wide sensor networks.

4.1.3 Processing Clustering Updates

Processing clustering updates is probably the most demanding task of the system. Clustering is known to be a NP-hard problem, so it is usually hard to get good results with light processing. To assess the impact of parameters on the processing requirements of the system, we monitor the number of times a new example forced an update of the clustering structure. Results in this matter were challenging. First, we should stress that the number of clustering updates is quite low compare to the total number of data points. Then, with respect to granularity, we could note that there is a clear dependence of the outcome with this parameter. However, it is not the same across different dimensionalities. It seems that processing requirements depend on a combination of w and d altogether (which makes sense since these are the two factors defining the search domain). A more expected result relates to the number of states to monitor, as we note an increase in processing requirements with the increase of m, supporting the notion that a higher number of states being monitored increases the number of less frequent states being included in the clustering process, hence increasing the number of required updates. To evaluate the sensitivity of the system to the number of sensors, we analysed the average result for a given value of granularity (w), averaged over all values of m (as communication is not dependent on this factor). On the right plot of Figure 7 we present these averaged results on the same dimensionality scale, with an exponential increase of the number of sensors. As expected, an increase in the number of sensors widened the domain of the problem, hence requiring more clustering updates. Nevertheless, we should point out that the amount of updates is kept extremely low compared to the total number of examples processed by the system.

4.1.4 Parameter Sensitivity

From the exposed results, we can stress that increasing the granularity (w) will only produce better results until a given limit is reached, possibly depending on the number of clusters to find (k), above which no benefits will be drawn from. Also, an unbounded increase in the number of states to monitor (m) will not, *per se*, yield better results, hence supporting our initial thought that only a small number of states are actually frequent and relevant for the clustering process. A problem that rises from this is that the number of states the system should monitor seems to depend on both the dimensionality (d) and the granularity (w).

4.2 Inter-Parameter Dependencies

From previous results, it seemed clear that as granularity should depend on the number of clusters to find, the number of states to monitor should also depend on the granularity and the dimensionality of the problem. We look for a good relation between the scenario (k and d) and parameters (w and m). Our assumption is that parameters should follow $w \in O(k)$ and $m \in O(dk^{\beta})$, for small β (possibly $\beta = 1$). To study this possibility, we define two factors ω and ϕ, where $w = \omega k + 1$ (allows extra cell which will mostly keep outliers) and $m = \phi w d$.

As first layers in the univariate grids have size $>> 20$, we set $\alpha = 0.05$ stating that a first-layer cell of the univariate grid will be split if it contains more that 5% of the total number of points. The initial range is set to $[0, 1]$. We set $d \in \{2, 3, 4, 5\}$ and $\sigma = 0.1$, varying $k \in \{3, 4, 5\}$, $\omega \in \{1, 2, 3, 4, 5\}$ and $\phi \in \{1, 2, 3, 4, 5\}$. Each scenario was evaluated with results averaged over 10 datasets. All w_i are set with the same value $w = \omega k + 1$. We vary k within small values to inspect the ability of the system to find well-separable clusters.

After aggregating all experiments, we computed Pearson's correlation between the parameters (ω and ϕ) and the resulting loss. The ω parameter reported (as expected) negative correlation with the loss ($\rho = -0.7524$), as better granularity diminishes the error implied by performing clustering on a grid instead of the real values. However, higher granularity also implies higher values for m, so a compromise should be found to optimize computational costs. After running some empirical tests (which will be subject of thorough evaluation in the future), we found that ω should be larger than 1, in order to allow the existence of infrequent cells between frequent ones, hence improving separability.

For the ϕ parameter, the study reported a positive correlation with loss ($\rho = 0.3832$), proving that growing m above a given value will not, by itself, increase the quality of model. Although this might go against the empirical intuition, the higher the m the higher the probability of including infrequent states in the clustering algorithm (because these start to be considered guaranteed top-m). This way, we decided to try a different approach, considering an upper bound on dw^2. After

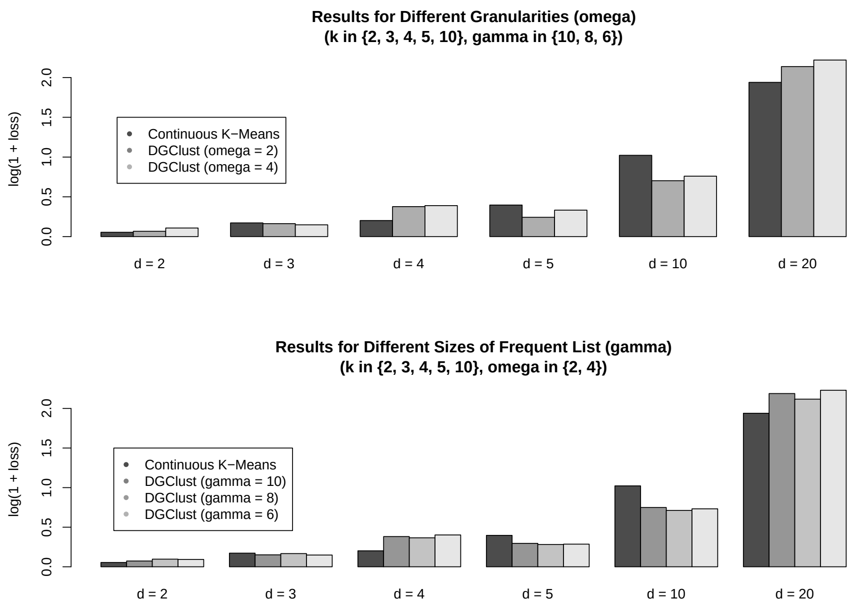


Figure 8: Averaged loss over $\omega \in \{2, 4\}$ and $\gamma \in \{6, 8, 10\}$, and 10 different data sets for each combination of parameters and scenarios, for *DGClust* and *Continuous K-Means*.

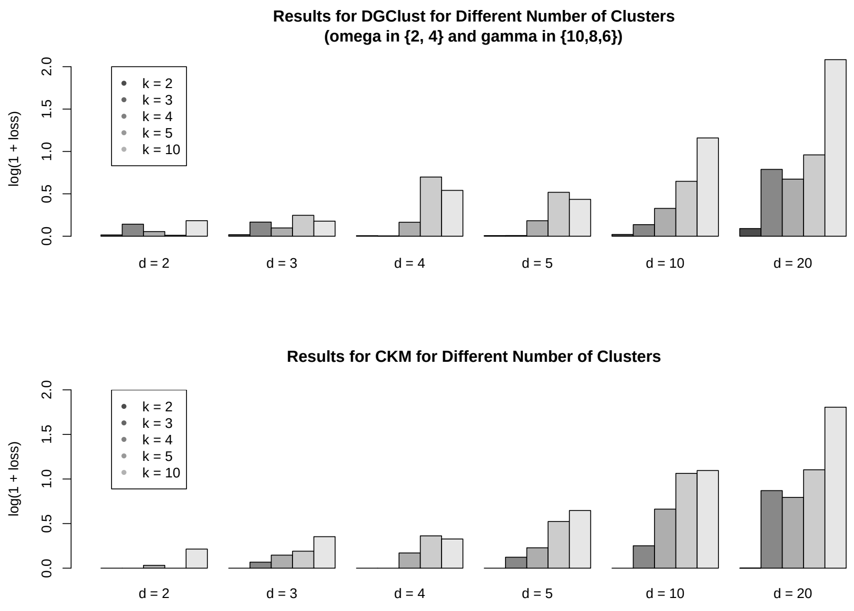


Figure 9: Averaged loss over $k \in \{2, 3, 4, 5, 10\}$ for each fixed parameter over 10 different datasets.

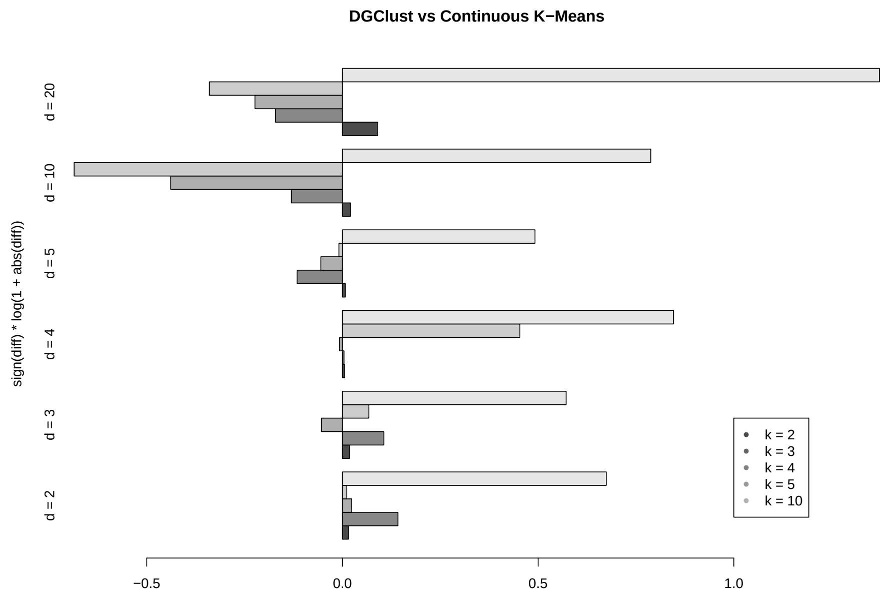


Figure 10: Average loss comparison for different dimensions d and number of clusters k.

some simple testing of admissible values for the parameter, m is then defined as $m = \frac{dw^2}{\gamma}$ with $\gamma \in \{10, 8, 6\}$.

Figures 8 and 9 present the results gathered for different scenarios, comparing with a simple centralized online *k-means* strategy, to which we refer as *Continuous K-Means*, which is a simplification of the STREAM algorithm [23]. This strategy performs a *K-Means* at each chunk of examples, keeping only the centers gathered with the last chunk of data weighted by the amount of points that were assigned to each center. Once again we note how hard it becomes to define a clear relation between w, d and m, although for higher values of k and d we could see some possible progressive paths towards an improvement in the competitiveness of our proposal, especially expressed in Figure 10.

However, the identification of general parameters is always discussable. Although we plan to perform more exhaustive sensitivity tests, in order to achieve at least acceptable ranges for the parameters, we should stress that the flexibility included in the system allows for better deployment in sensor networks and resource restricted environments.

4.3 Application to Physiological Sensor Data

The Physiological Data Modeling Contest Workshop (PDMC) was held at the ICML 2004 and aimed at information extraction from streaming sensors data. The training data set for the competition consists of approximately 10,000 hours of this data, containing several variables: userID, sessionID, sessionTime, characteristic[1..2], annotation, gender and sensor[1..9]. We have concentrated on sensors 2 to 9, extracted by userID, resulting in several experimental scenarios of eight sensors, one scenario per userID.

For each scenario, we run the system with different values for the parameters, and compare the results both with the *Continuous K-Means* and full data *K-Means*, the latter serving as "real" centers definition. Since different sensors produce readings in different scales, we inspect the distribution of each sensor on an initial chunk of data, defining the initial ranges to percentiles 25% and 75%. This process is acceptable in the sensor networks framework as expert knowledge about the range is usually available. Hence, we are also allowing the system to adapt the local

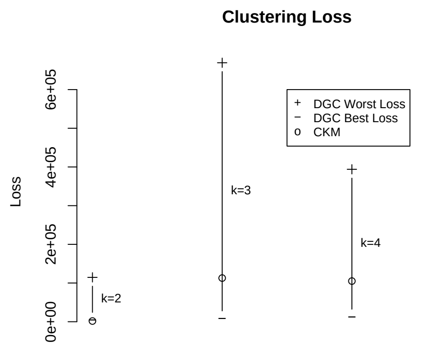


Figure 11: Performance in terms of loss with $k \in \{2, 3, 4, 5\}$ and, for each k, $\omega \in \{2, 4, 6, 8, 10\}$ and $\gamma \in \{10, 8, 6, 4, 2\}$. The circles refer to the loss achieved when a centralized online clustering algorithm is applied in the entire data.

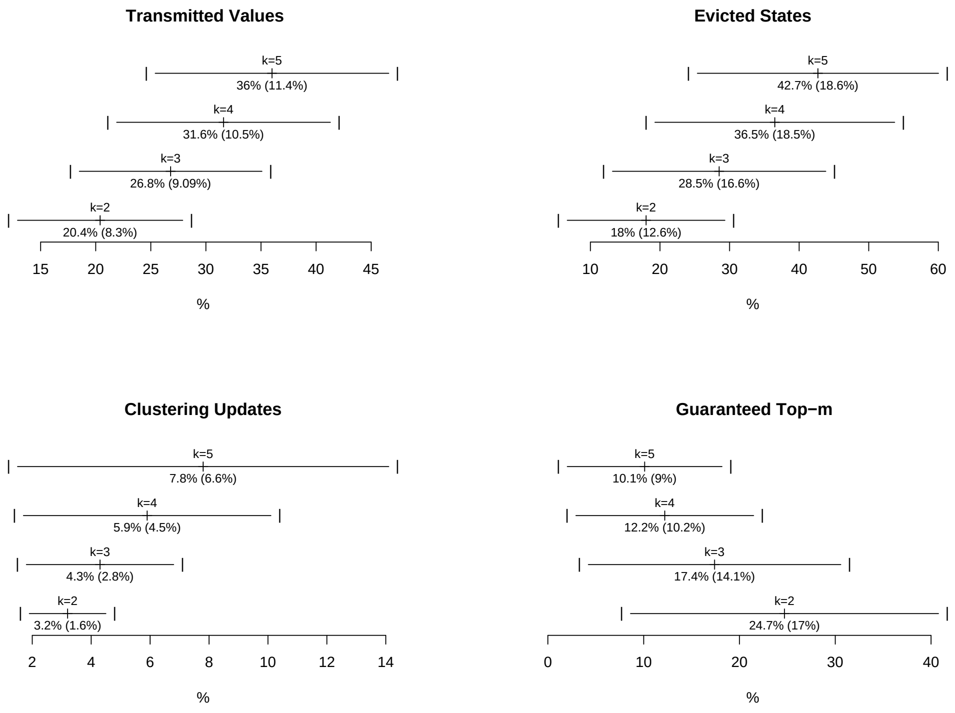


Figure 12: Performance in terms of communicated values (% of total examples × dimensions), evicted states (% of total examples), cluster centers adaptations (% of total examples) and number of guaranteed top-m (% of total m) with $k \in \{2, 3, 4, 5\}$ and, for each k, $\omega \in \{2, 4, 6, 8, 10\}$ and $\gamma \in \{10, 8, 6, 4, 2\}$.

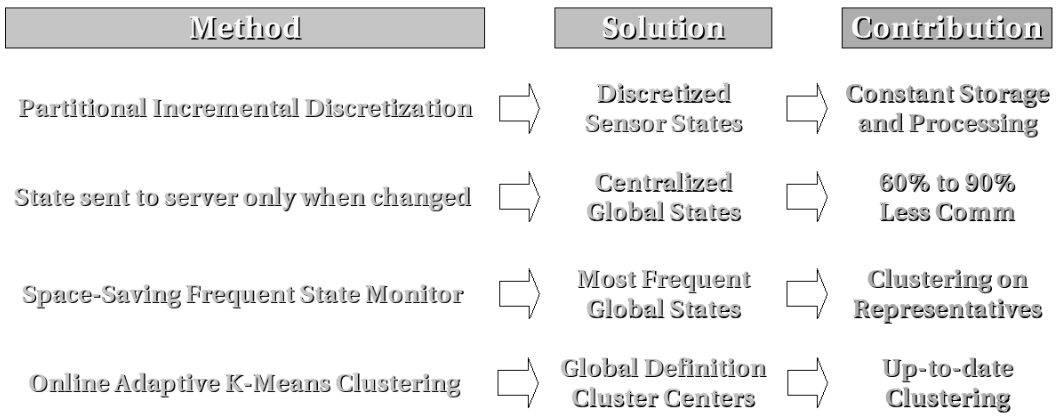

Method
Solution
Contribution
Partitional Incremental Discretization
Discretized Sensor States
Constant Storage and Processing
State sent to server only when changed
Centralized Global States
60% to 90% Less Comm
Space-Saving Frequent State Monitor
Most Frequent Global States
Clustering on Representatives
Online Adaptive K-Means Clustering
Global Definition Cluster Centers
Up-to-date Clustering

Figure 13: DGClust: Main results achieved by applying this strategy.

grids accordingly.

The system ran with $k \in \{2, 3, 4, 5\}$, $\omega \in \{2, 4, 6, 8, 10\}$ and $\gamma \in \{10, 8, 6, 4, 2\}$. Figure 11 presents performance results in terms of loss. Beyond loss, also monitored was the amount of communication and cluster adaptation in each run. Figure 12 presents the resulting performance statistics. Given the characteristics of sensor data, subsequent readings tend to stay in the same interval. Hence the advantage of local discretization: the system transmits only around 30% of the total amount of values, including transmissions of recalculated second layer intervals. Also, we should note that only a small part of the points require cluster center adaptation (less than 10%). Overall we should stress that, given the flexibility of the system, a suitable combination of parameters can yield better results than centralizing the data, while preventing excessive communication in the sensor network.

5 Discussion

The main characteristics of the system, arising also as important advantages, include: compact representation of possibly infinite streams produced in distributed sensors, with possibly different granularities for each sensor; possibility to adapt discretization and central processing to resources available in the network; centralized monitoring of the global state of the network, with lightweight maintenance of frequent states; and fast and adaptable clustering of frequent states to estimate actual clusters centers.

Following the main intuition supporting our research, the advantages of using *DGClust* algorithm are multifaceted:

- Each sensor's data is processed locally, being incrementally discretized into a univariate adaptive grid, with each new value triggering a specific cell of this grid, defining the local state of the sensor – this in fact decreases the system's sensitivity to uncertainty in sensed data.
- Whenever a local site change its state, that is, the triggered cell changes, the new state is communicated to a central site – this highly reduces the network's communication requirements with the central site).
- The central server keeps a small list of counters of the most frequent global states – this is extremely helpful as sensor networks may include thousands of sensors, leading to an exponential number of cell combinations that should be monitored by the central site.
- The final clustering structure is defined and maintained by a simple partitional clustering algorithm, applied on the frequent states, which is capable of online update and adaptation

to the evolution of the data.

Figure 13 summarises the main results obtained by applying this strategy, from which we set focus on the communication gains and the ability to process and cluster distributed sensor data from a streaming point of view.

6 Related Work

The method we present in this paper is related with three other areas of research in data streams: sensor network data processing, frequent items monitoring and distributed data clustering.

6.1 Sensor Data and Networks

Sensors are usually small, low-cost devices capable of sensing some attribute of a physical phenomenon. In terms of hardware development, the state-of-the-art is well represented by a class of multi-purpose sensor nodes called *motes* [9], which were originally developed at UC Berkeley and are being deployed and tested by several research groups and start-up companies. Although common applications are traditionally developed on low-level programming of the *motes*, recent programming languages and environments such as Regiment [22] and EnviroSuite [19], provide high level programming abstractions, allowing more complex programming and usage of sensor devices as processing units for knowledge extraction scenarios.

Sensor networks are composed of a variable number of sensors (depending on the application), which have several distinguishing features: (a) the number of nodes is potentially very large and thus scalability is a problem, (b) the individual sensors are prone to failure given the often challenging conditions they experiment in the field, (c) the network topology changes dynamically, (d) broadcast protocols are used to route messages in the network, (e) limited power, computational, and memory capacity, and (f) lack of global identifiers [3].

In sensor network applications, data routing is usually based on two main approaches: (a) sensors broadcast advertisements for the availability of the data and wait for interested nodes, or; (b) sinks broadcast interest in the data and wait for replies from the sensors. Common problems with these strategies are *implosion* and *overlap* [3]. To prevent it, *data aggregation* uses the limited processing power and memory of the sensing devices to process data online, where data is gathered from a neighborhood of sensor nodes and combined in a receiving node along the path to the sink.

6.2 Frequent Items in Data Streams

The problem of finding the most frequent items in a data stream S of size N is, roughly put, the problem to find the elements e_i whose relative frequency f_i is higher than a user specified support ϕN, with $0 \leq \phi \leq 1$. Given the space requirements that exact algorithms addressing this problem would need [6], several algorithms were already proposed to find the top-k frequent elements, being roughly classified into *counter-based* and *sketch-based* [21]. *Counter-based* techniques keep counters for each individual element in the monitored set, which is usually a lot smaller than the entire set of elements. When an element is seen which is not currently being monitored, different algorithms take different actions in order to adapt the monitored set accordingly. *Sketch-based* techniques provide less rigid guarantees, but they do not monitor a subset of elements, providing frequency estimators for the entire set.

Simple *counter-based* algorithms such as *Sticky Sampling* and *Lossy Counting* were proposed in [20], which process the stream in reduced size. Yet, they suffer from keeping a lot of irrelevant counters. *Frequent* [11] keeps only k counters for monitoring k elements, incrementing each element counter when it is observed, and decrementing all counters when a unmonitored element is observed. Zeroed-counted elements are replaced by new unmonitored element. This strategy is similar to the one applied by *Space-Saving* [21], which gives guarantees for the top-m most frequent elements.

Sketch-based algorithms usually focus on families of hash functions which project the counters into a new space, keeping frequency estimators for all elements. The guarantees are less strict but all elements are monitored. The *CountSketch* algorithm [6] solves the problem with a given success probability, estimating the frequency of the element by finding the median of its representative counters, which implies sorting the counters. Also, *GroupTest* method [7] employs expensive probabilistic calculations to keep the majority elements within a given probability of error. Although generally accurate, its space requirements are large and no information is given about frequencies or ranking.

6.3 Clustering Data Streams

It is known that solving a clustering problem is the equivalent to finding the global optimal solution of a non-linear optimization problem, hence NP-hard, suggesting the application of optimization heuristics [4]. The main problem in applying clustering to data streams is that systems should consider data evolution, being able to compress old information and adapt to new concepts. The range of clustering algorithms that operate online over data streams is wide, including *partitional*, *hierarchical*, *density-based* and *grid-based* methods. A common connecting feature is the definition of unit cells or representative points, from which clustering can be obtained with less computational costs. In this paper we address two types of clustering procedures: *point-based* clustering and *grid-based* clustering.

6.3.1 Point-based Clustering

Several algorithms operate over summaries or samples of the original stream. Bradley et al. [5] proposed the *Single Pass K-Means*, increasing the capabilities of *k-means* for large datasets, by using a buffer where points of the dataset are kept in a compressed way. The *STREAM* [23] system can be seen as an extension of [5] which keeps the same goal but has as restriction the use of available memory. After filling the buffer, STREAM clusters the buffer into k clusters, retaining only the k centroids weighted by the number of examples in each cluster. The process is iteratively repeated with new points. The BIRCH hierarchical method [31] uses *Clustering Features* to keep sufficient statistics for each cluster at the nodes of a balanced tree, the *CF-tree*. Given its hierarchical structure, each non-leaf node in the tree aggregates the information gathered in the descendant nodes. This algorithm tries to find the best groups with respect to the available memory, while minimizing the amount of input and output. Another use of the *CF-tree* appears in [1].

A different strategy is used in another hierarchical method, the *CURE* system [18], where each cluster is represented by a constant number of points well distributed within the cluster, which capture the extension and shape of the cluster. This process allows the identification of clusters with arbitrary shapes on a random sample of the dataset, using Chernoff bounds in order to obtain the minimum number of required examples. The same principle of error-bounded results was recently used in VFKM to apply consecutive runs of k-means, with increasing number of examples, until the error bounds were satisfied [12]. This strategy supports itself on the idea of guaranteeing that the clustering definition does not differ significantly from the one gather with infinite data. Hence, it does not consider data evolution.

6.3.2 Grid-based Clustering

The main focus of *grid-based* algorithms is the so called *spatial data*, which model the geometric structure of objects in space. These algorithms divide the data space in small units, defining a grid, and assigning each object to one of those units, proceeding to divisive and aggregative operations hierarchically. These features make this type of methods similar to hierarchical algorithms, with the main difference of applying operations based on a parameter rather than the dissimilarities between objects.

A sophisticated example of this type of algorithms is *STING* [30], where the space area is divided in cells with different levels of resolution, creating a layered structure. The main features and advantages of this algorithm include being incremental and able of parallel execution. Also, the idea of dense units, usually present in *density-based* methods [13], has been successfully introduced in *grid-based* systems. The *CLIQUE* algorithm tries to identify sub-spaces of a large dimensional space which can allow a better clustering of the original data [2]. It divides each dimension on the same number of equally ranged intervals, resulting in exclusive units. One unit is accepted as dense if the fraction of the total number of points within the unit is higher than a parameter value. A cluster is the largest set of contiguous dense units within a subspace. This technique's main advantage is the fact that it automatically finds subspaces of maximum dimensionality in a way that high density clusters exist in those subspaces.

The inclusion of the notion of dense units in simpler *grid-based* methods presents several benefits. *Statistical Grid-based Clustering* system [24] was especially designed for data stream applications, where clusters are constituted by adjacent dense cells. It works by applying three different divisive methods, based on the statistics of objects belonging to each cell: *μ-partition*, which divides one cluster in two setting the border at the mean of the parent group; *σ-partition*, which divides the group in two, one with 68% of the objects belonging to $[\mu-\sigma, \mu+\sigma]$ (assuming a normal distribution of objects) and another with the remainder tail objects; and a third method which includes the efficient features of the previous, *hybrid-partition*. However, in distributed systems, the increase in communication given the need to keep sufficient statistics may be prejudicial.

6.3.3 Distributed Clustering

Since current applications generate many pervasive distributed computing environments, data mining systems must nowadays be designed to work not as a monolithic centralized application but as a distributed collaborative process. The centralization of information yields problems not only with resources such as communication and memory, but also with privacy of sensitive information. Instead of centralizing relevant data in a single server and afterwards perform the data mining operations, the entire process should be distributed and, therefore, paralleled throughout the entire network of processing units. A recent example of such techniques was proposed by Subramaniam et al., where an online system uses density distribution estimation to detect outliers in distributed sensor data [29].

Recent research developments in clustering are directed towards distributed algorithms for continuous clustering of examples over distributed data streams. In [10] the authors present a distributed majority vote algorithm which can be seen as a primitive to monitor a k-means clustering over peer-to-peer networks. The k-means monitoring algorithm has two major parts: monitoring the data distribution in order to trigger a new run of k-means algorithm and computing the centroids actually using the k-means algorithm. The monitoring part is carried out by an exact local algorithm, while the centroid computation is carried out by a centralization approach. The local algorithm raises an alert if the centroids need to be updated. At this point data is centralized, a new run of k-means is executed, and the new centroids are shipped back to all peers.

Cormode et al. [8] proposed different strategies to achieve the same goal, with local and global computations, in order to balance the communication costs. They considered techniques based on the *furthest point* algorithm [17], which gives a approximation for the radius and diameter of clusters with guaranteed cost of two times the cost of the optimal clustering. They also present the *parallel guessing* strategy, which gives a slightly worse approximation but requires only a single pass over the data. They conclude that, in actual distributed settings, it is frequently preferable to track each site locally and combine the results at the coordinator site. These methods of combining local and central processing are paradigmatic examples of the path that distributed data mining algorithms should traverse.

7 Concluding Remarks and Future Work

In this paper, we have proposed *DGClust*, a new distributed grid clustering algorithm for data produced on wide sensor networks. Its core is based on online discretization of data, frequent state monitoring, and online partitional clustering. These techniques jointly work towards a reduction of both the dimensionality and the communication burdens. Experiments are presented in terms of sensitivity tests and application to a real-world data set, from which some advantages of this approach could be exploited. Current work is concentrated on determining acceptable ranges for the parameters of the system and application to more real-world data. Future work will focus on techniques to monitor the evolution of the clusters, taking advantage from the adaptive update of clusters already implemented in the system. Furthermore, we are preparing the deployment of the system in real wireless sensor networks, in order to better assess the sensitivity and advantages of the system with respect to restricted resources requirements.

Acknowledgment

The work of P.P. Rodrigues is supported by the Portuguese Foundation for Science and Technology (FCT) under the PhD Grant SFRH/BD/29219/2006. The authors thank FCT's Plurianual financial support attributed to LIAAD and CRACS and the participation of Project ALES II under FCT's Contract POSC/EIA/55340/2004 and Project CALLAS under FCT's Contract PTDC/EIA/71462/2006.

References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases*, pages 81–92. Morgan Kaufmann, September 2003.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 94–105, Seattle, Washington, June 1998. ACM Press.

[3] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A Survey on Sensor Networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[4] M. Bern and D. Eppstein. *Approximation Algorithms for NP-hard Problems*, chapter 8. Approximation Algorithms for Geometric Problems, pages 296–345. PWS Publishing Company, 1996.

[5] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 9–15. AAAI Press, 1998.

[6] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, pages 693–703. Springer Verlag, 2002.

[7] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. In *Proceedings of the 22nd Symposium on Principles of Databse Systems*, pages 296–306, 2003.

[8] G. Cormode, S. Muthukrishnan, and W. Zhuang. Conquering the divide: Continuous clustering of distributed data streams. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007)*, pages 1036–1045, 2007.

[9] D. E. Culler and H. Mulder. Smart Sensors to Network the World. *Scientific American*, 2004.

[10] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta. Distributed data mining in peer-to-peer networks. *IEEE Internet Computing*, 10(4):18–26, 2006.

[11] E. D. Demaine, A. Lopez-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *Proceedings of the 10th Annual European Symposium on Algorithms*, pages 348–360, 2002.

[12] P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, pages 106–113, 2001.

[13] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In E. Simoudis, J. Han, and U. Fayyad, editors, *Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, 1996. AAAI Press.

[14] J. Gama and C. Pinto. Discretization from data streams: applications to histograms and data mining. In H. Haddad, editor, *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC 2006)*, pages 662–667, Dijon, France, April 2006. ACM Press.

[15] J. Gama and P. P. Rodrigues. Data stream processing. In J. Gama and M. M. Gaber, editors, *Learning from Data Streams - Processing Techniques in Sensor Networks*, chapter 3, pages 25–39. Springer Verlag, 2007.

[16] J. Gama, P. P. Rodrigues, and R. Sebastião. Evaluating algorithms that learn from data streams. In *Proceedings of the 24th Annual ACM Symposium on Applied Computing (SAC 2009)*, pages 1496–1500, Honolulu, Hawaii, USA, 2009. ACM Press.

[17] T. F. Gonzalez. Clustering to minimize the maximum inter-cluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[18] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In L. M. Haas and A. Tiwary, editors, *Proceedings of the 1998 ACM-SIGMOD International Conference on Management of Data*, pages 73–84. ACM Press, 1998.

[19] L. Luo, T. Abdelzaher, T. He, and J. Stankovic. EnviroSuite: An Environmentally Immersive Programming Framework for Sensor Networks. *ACM TECS*, 5(3):543–576, 2006.

[20] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 346–357, 2002.

[21] A. Metwally, D. Agrawal, and A. E. Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of the 10th International Conference on Database Theory*, pages 398–412. Springer Verlag, 2005.

[22] R. Newton and M. Welsh. Region Streams: Functional Macroprogramming for Sensor Networks. In *DMSN'04 Workshop*, 2004.

[23] L. O'Callaghan, A. Meyerson, R. Motwani, N. Mishra, and S. Guha. Streaming-data algorithms for high-quality clustering. In *Proceedings of the Eighteenth Annual IEEE International Conference on Data Engineering*, pages 685–696. IEEE Computer Society, 2002.

[24] N. H. Park and W. S. Lee. Statistical grid-based clustering over data streams. *SIGMOD Record*, 33(1):32–37, 2004.

[25] P. P. Rodrigues and J. Gama. Clustering techniques in sensor networks. In J. Gama and M. M. Gaber, editors, *Learning from Data Streams - Processing Techniques in Sensor Networks*, chapter 9, pages 125–142. Springer Verlag, 2007.

[26] P. P. Rodrigues and J. Gama. A system for analysis and prediction of electricity load streams. *Intelligent Data Analysis*, 13(3):477-496, 2009.

[27] P. P. Rodrigues, J. Gama, and L. Lopes. Knowledge discovery for sensor network comprehension. In A. Cuzzocrea, editor, *Intelligent Techniques for Warehousing and Mining Sensor Network Data*. IGI Global, 2009.

[28] P. P. Rodrigues, J. Gama, and L. Lopes. Requirements for clustering streaming sensors. In A. R. Ganguly, J. Gama, O. A. Omitaomu, M. M. Gaber, and R. R. Vatsavai, editors, *Knowledge Discovery from Sensor Data*, chapter 4, pages 33–51. CRC Press, 2009.

[29] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *VLDB*, pages 187–198, 2006.

[30] W. Wang, J. Yang, and R. R. Muntz. STING: A statistical information grid approach to spatial data mining. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *Proceedings of the Twenty-Third International Conference on Very Large Data Bases*, pages 186–195, Athens, Greece, 1997. Morgan Kaufmann.

[31] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114. ACM Press, 1996.

A Experimental Evaluation Detailed Results

This appendix presents the complete set of results from the experiments endured in the parameter sensitivity analysis. Data is generated for a given scenario with $k = 3$ clusters, ranging dimensionality in

$$d \in \{2, 4, 8, 16, 32, 64, 128\}.$$

For each scenario, 10 different datasets were created, using $\sigma = .05$ and parameters varying as: the univariate grid granularity ranging in

$$w \in \{5, 7, 9, 11, 13, 15, 17, 19, 21\},$$

and the number of states to monitor raging in

$$m \in \{6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45\}.$$

Figures 14, 15 and 16 present the impact of the parameters on three final outcomes, respectively: loss (to real centroids), communication (from sensors to the server) and processing (of clustering updates). Loss is measured using the loss function of Equation 1. Communication is assess with the transmission rate as a percentage of the total number of examples times the number of sensors. Processing is evaluated as the amount of clustering updates required as a percentage of the total number of examples. In each figure, first plot presents the impact of the number of sensors on the outcome, for each granularity, averaged over all values of m. The remaining plots of the figure present the estimate of the mean outcome (error bars represent the 95% confidence interval) for each combination of parameters (d, w, m).

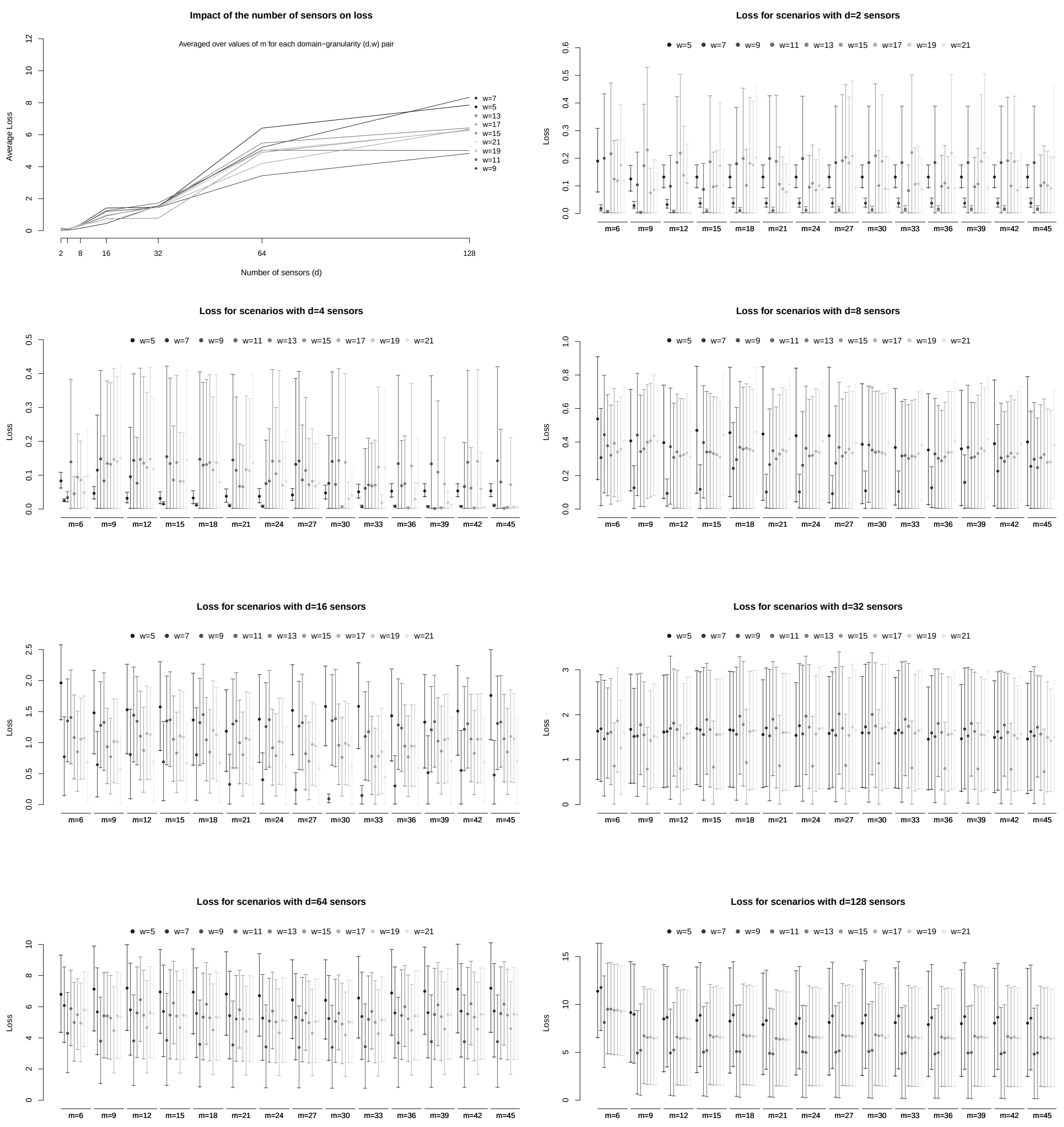


Figure 14: Experimental evaluation in terms of loss to the real centroids. First plot presents the impact of the number of sensors (from $d = 2$ to $d = 128$) on loss, for each granularity (from $w = 5$ to $w = 21$), averaged over all values of m (from $m = 6$ to $m = 45$). The remaining plots present the loss (error bars represent the 95% confidence interval) for each combination of parameters (d, w, m).

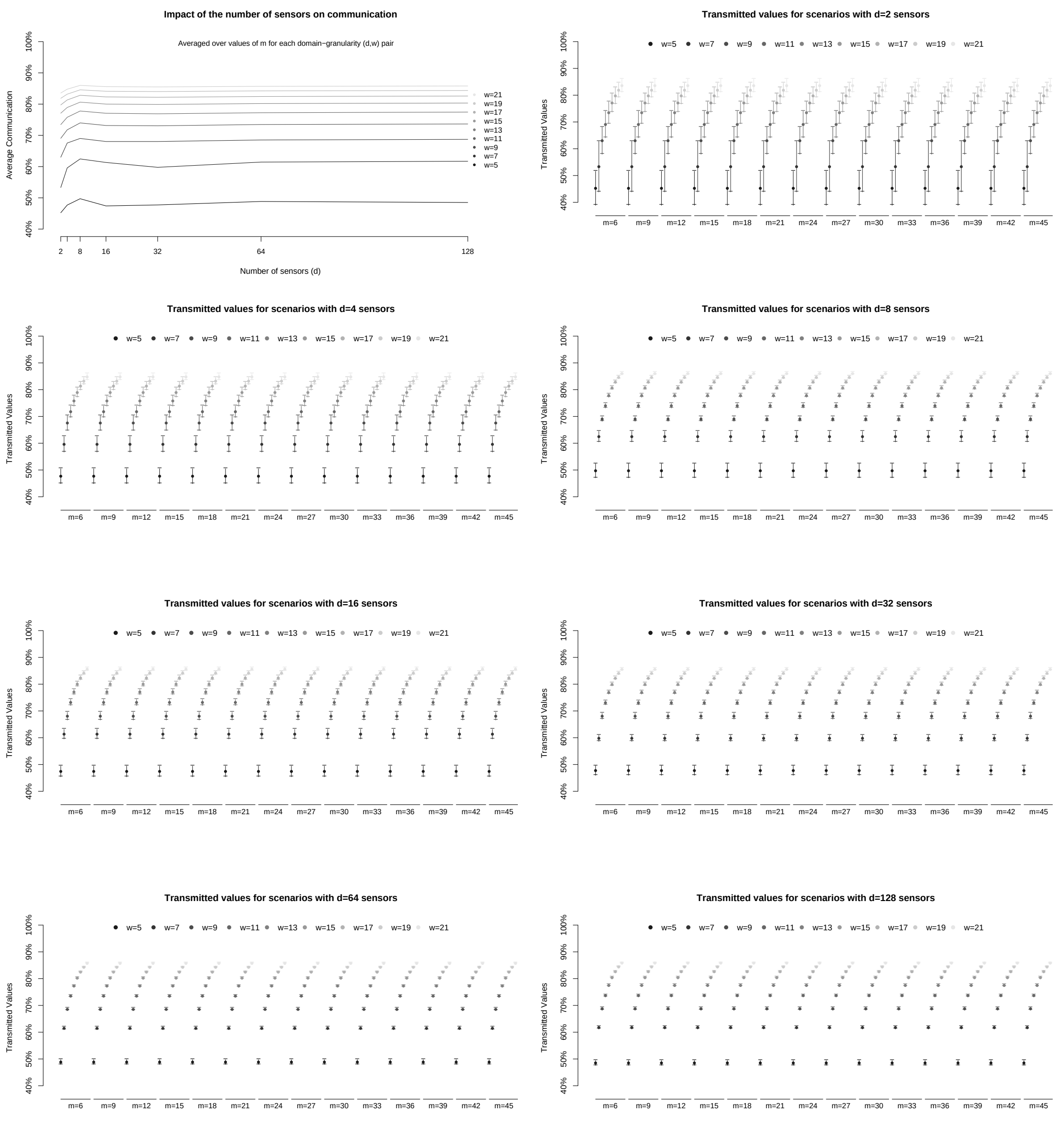


Figure 15: Experimental evaluation in terms of transmitted values (% of examples $\times$ dimensions). First plot presents the impact of the number of sensors (from $d = 2$ to $d = 128$) on transmitted values, for each granularity (from $w = 5$ to $w = 21$), averaged over all values of m (from $m = 6$ to $m = 45$). The remaining plots present the amount of transmitted values (error bars represent the 95% confidence interval) for each combination of parameters (d, w, m).

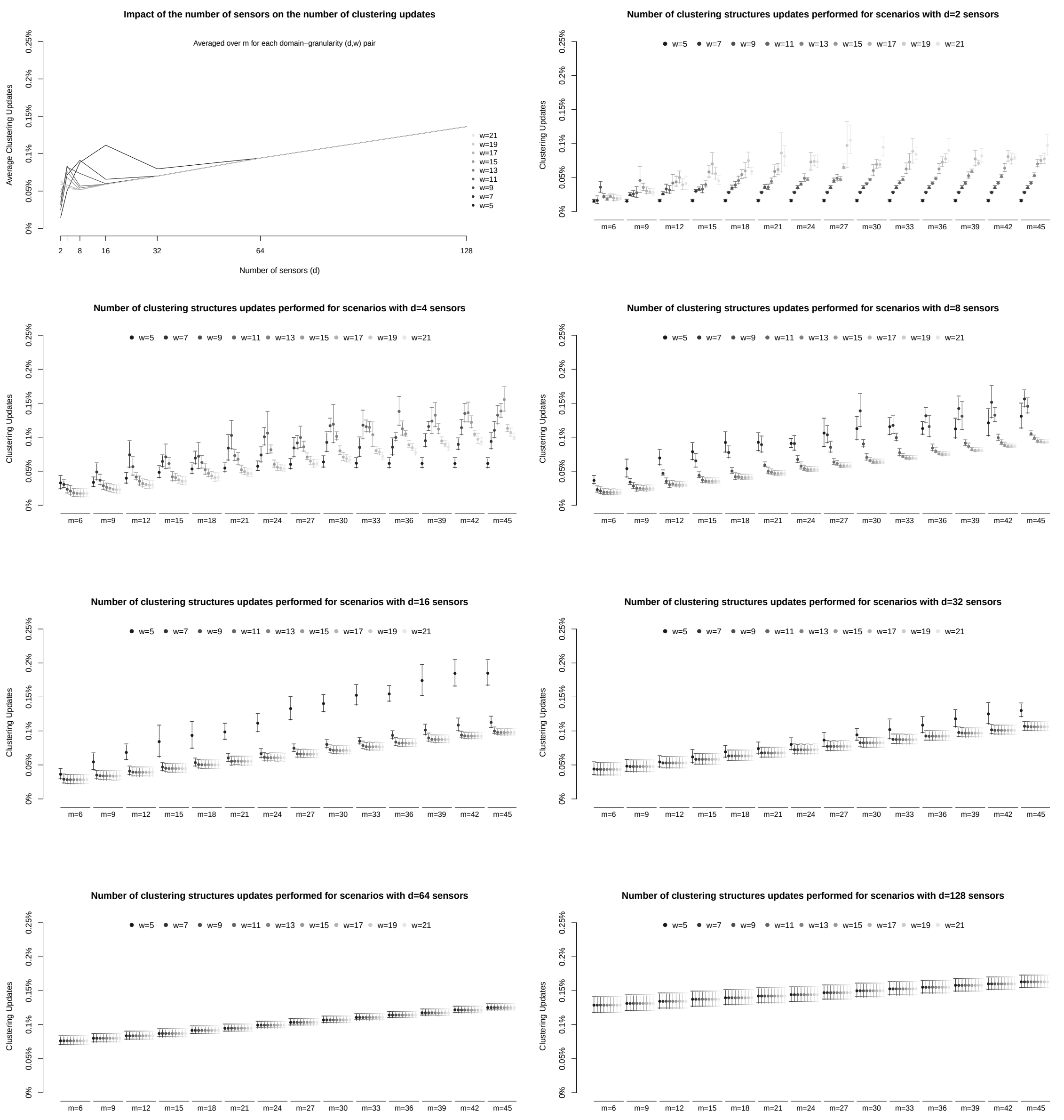


Figure 16: Experimental evaluation in terms of the number of clustering updates (% of examples). First plot presents the impact of the number of clusters (from $d = 2$ to $d = 128$) on the number of clustering updates, for each granularity (from $w = 5$ to $w = 21$), averaged over all values of m (from $m = 6$ to $m = 45$). The remaining plots present the amount of clustering updates (error bars represent the 95% confidence interval) for each combination of parameters (d, w, m).