First International Workshop on Data Mining in Sensor Networks

Chairs: Rasmus Pedersen & Eric Jul

Keynote by Samuel Madden, MIT

Table of Content

An adaptive modular approach to the mining of sensor network data.

Gianluca Bontempi, Yann-Aël Le Borgne*
ULB Machine Learning Group
Université Libre de Bruxelles, Belgium
email: {gbonte,yleborgn}@ulb.ac.be

Abstract

This paper proposes a two-layer modular architecture to adaptively perform data mining tasks in large sensor networks. The architecture consists in a lower layer which performs data aggregation in a modular fashion and in an upper layer which employs an adaptive local learning technique to extract a prediction model from the aggregated information. The rationale of the approach is that a modular aggregation of sensor data can serve jointly two purposes: first, the organization of sensors in clusters, then reducing the communication effort, second, the dimensionality reduction of the data mining task, then improving the accuracy of the sensing task.

1 Introduction.

There are plenty of potential applications for intelligent sensor networks: distributed information gathering and processing, monitoring, supervision of hazardous environments, intrusion detection, cooperative sensing, tracking.

The ever-increasing use of sensing units asks for the development of specific data mining architectures. What is expected from these architectures is not only accurate modeling of high dimensional streams of data but also a minimization of the communication and computational effort demanded to each single sensor unit.

The simplest approach to the analysis of sensor network data makes use of a centralized architecture where a central server maintains a database of readings from all the sensors. The whole analysis effort is localized in the server, whose mission is to extract from the flow of data the high-level information expected to be returned by the monitoring system. If we assume that reasonable-size sensor networks will be made of thousands of nodes, the limitation of this approach is strikingly evident: the number of messages sent in the system as well as the number of variables of the data mining task are too large to be managed efficiently.

It has been suggested in literature that alternative architectures are to be preferred in applications where neighboring sensors are likely to have correlated readings [6]. This is the case of *aggregating systems* which, according to the definition of [10], are systems where the data obtained from the different source nodes can be aggregated before being transmitted along the network. In these systems, we can imagine the existence of intermediary nodes (*aggregators*) having the capability to fuse the information from different sources. Sensor networks for weather forecasting and monitoring are examples of aggregating systems. The authors of [6] showed that a compression of the sensor information can be performed at local level then reducing the amount of communication and the bandwidth required for the functioning of the system.

At the same time techniques of data compression, like Principal Component analysis (PCA) or Independent Component Analysis (ICA) [8], are often used in data mining to reduce the complexity of modeling tasks with a very large number of variables. It is well known in the data mining literature that methods for reducing complexity are beneficial for several reasons: improvement of the accuracy and intelligibility of the model, reduced storage and time requirements.

The rationale of the paper is that a modular organization of the sensor network can be used to jointly address the two main issues in mining sensor network data: the minimization of the communication effort and the accurate extraction of high-level information from massive and streaming datasets.

In particular this paper proposes a data driven procedure to configure a two-layer topology of a sensor network (Figure 1) made of

1. a lower level whose task is to organize the sensors in clusters, compress their signals and transmit the aggregate information to the upper level,
2. an upper level playing the role of a data mining server which uses the aggregate information to

*Supported by the **COMP²SYS** project, sponsored by the HRM program of the European Community (MEST-CT-2004-505079)

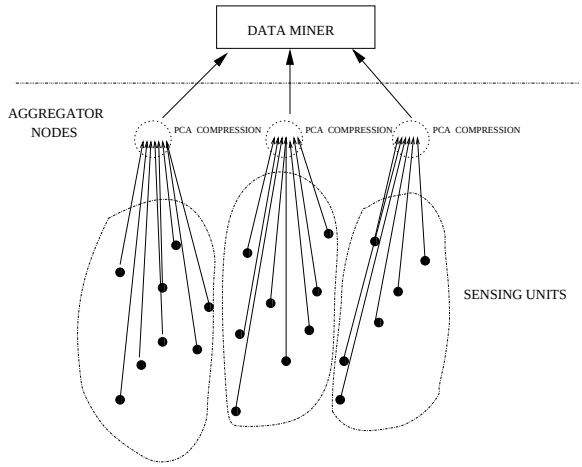


Figure 1: The black dots represent the sensing units. The dotted circles represent the aggregator nodes which carry out the fusion of data coming from neighboring sensors before sending the aggregated signals up to the data mining server.

carry out the required sensing task.

We focus here on problems where the sensors are used to perform a supervised learning (e.g. classification, regression or prediction) task: examples could be the classification of traffic fluidity on the basis of route sensing units or the prediction of a wave intensity on the basis of sensors scattered in the ocean. Our approach consists in using a historical data set to find the best way to combine the measures of neighboring sensors such that the accuracy of the prediction model based on such aggregate measures is optimized. The design procedure relies on an iterative optimization procedure which loops over five steps: (i) a partition of the sensing units in proximity clusters, (ii) the compression of the signals of each cluster of sensors, (iii) the aggregation and transmission of the compressed signals to the upper data mining server, (iv) the training of the prediction model in the data mining server, and (v) the assessment of the partition according to multiple criteria, like the prediction accuracy of the data mining model and the energy and transmission requirements of the resulting network. The best partition which is returned by this multi-criteria optimization procedure can be used as a template for the topological organization of sensors.

An important issue in mining sensor network data concerns the streaming and possibly non stationary nature of data. Non stationarity may be due to changes in the phenomenon underlying the measures as well to sensor malfunctioning and/or modifications of their geographical location. In order to address this aspect we have recourse to adaptive features at both levels of our architecture. At the lower sensor integration level we use an effective sequential implementation of the Principal Component Analysis (PCA) technique: the PAST algorithm [11]. The upper data mining module uses an adaptive lazy learning (LL) technique [1] characterized by a fast training phase and an effective treatment of non stationarity.

The experimental section of the paper presents some preliminary results obtained by adopting the proposed two-layer architecture in the context of a simulated monitoring task: measuring the wavelength of a two dimensional wave in situation of scattering.

2 The problem

Consider a sensor network $\mathcal{S}$ made of S sensors where P is a $[S, 3]$ matrix containing the three-dimensional coordinates of the S sensors and

$$x(t) = \{s_1(t), s_2(t), \ldots, s_S(t)\} \tag{2.1}$$

is the state (or snapshot) of the sensor network at time t. Suppose we intend to employ $\mathcal{S}$ to perform a supervised learning task, for example a regression problem

$$y(t) = f(x(t)) + \varepsilon(t) \tag{2.2}$$

where y is the variable to be predicted at time t on the basis of the state $x(t)$ of the network $\mathcal{S}$ and ε is usually thought as the term including modeling error, disturbances and noise.

If we have available a finite dataset $D_N = \{\langle x(t_i), y(t_i)\rangle, i = 1, \ldots, N\}$ of N input-output observations, this problem can be tackled as a conventional regression problem, by first estimating an approximator of f on the basis of D_N and then using this estimator as a predictor of y.

However, if, like in the case of sensor networks, the number S is huge, the mapping f is non-stationary and the data are collected sequentially, conventional techniques reach rapidly their limits. In particular, the large dimensionality of the problem asks for feature selection problem as well as the streaming aspect of the problem requires sequential (also called recursive) estimation approaches.

This paper proposes an approach to the problem of data mining in sensor networks which tries to conciliate the needs for an accurate prediction of the output y with the constraints related to energy reserves, communication bandwidth and sensor computational power.

The following subsections will rapidly sketch the two computational modules used in our approach: the recursive PCA and the adaptive Lazy Learning. Section 5 will describe how these modules are combined in our architecture for mining sensor networks.

3 PCA compression techniques

As discussed above, each data mining problem in the context of sensor network data with large S has to face the problem of reducing dimensionality. Existing techniques for feature selection (for an up-to-date state of the art on feature selection see [7]) can be grouped into two main approaches: the wrapper and the filter approach. In the wrapper approach [9] the feature subset selection algorithm exists as a wrapper around the learning algorithm, which is often considered as a black box able to return (e.g. via cross-validation) an evaluation of the quality of a feature subset. On the contrary, the filter approach selects features using a preprocessing step independently of the learning algorithm.

In this paper we will adopt the Principal Component analysis (PCA) technique, an instance of the filter approach. PCA is a classic technique in statistical data analysis, feature extraction and data compression [8]. Given a set of multivariate measurements, its goal is to find a smaller set of variables with less redundancy, that would give as good a representation as possible. In PCA the redundancy is measured by computing linear correlations between variables. PCA entails transforming the n original variables $x_1, \ldots, x_n$ into m new variables $z_1, \ldots, z_m$ (called *principal components*) such that the new variables are uncorrelated with each other and account for decreasing portions of the variance of the original variables. Consider a vector x of size n and a matrix X containing N measures of the vector x. The m principal components

$$z_k = \sum_{j=1}^{n} w_{jk} x_j = w_k^T x, \qquad k = 1, \ldots, m \tag{3.3}$$

are defined as weighted sums of the elements of x with maximal variance, under the constraints that the weights are normalized and the principal components are uncorrelated with each other. It is well-known from basic linear algebra that the solution to the PCA problem is given in terms of the unit-length eigenvectors $e_1, \ldots, e_n$ of the correlation matrix of x. Once ordered the eigenvectors such that the corresponding eigenvalues $\lambda_1, \ldots, \lambda_n$ satisfy $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$, the principal component z_k is given by $z_k = e_k^T x$. It can be shown that the PCA problem can be also put in the form of a minimum mean-square error compression of x. This means that the computation of the w_k for the first m principal components is equivalent to find the orthonormal basis $w_1, \ldots, w_m$ that minimizes

$$J_{PCA} = \frac{1}{N} \sum_{t=1}^{N} \|x(t) - \sum_{k=1}^{m} (w_k^T x(t)) w_k\|^2 \tag{3.4}$$

If we denote $W = (w_1, \ldots, w_m)^T$ where W is a matrix of size $[m, n]$ we have

$$J_{PCA} = \frac{1}{N} \sum_{t=1}^{N} \|x(t) - W^T W x(t)\|^2 \tag{3.5}$$

What is appealing in this formulation is that a recursive formulation of this least-squares problem is provided by the Projection Approximation Subspace Tracking (PAST) algorithm proposed by [11]. This algorithm, based on the recursive formulation of the least squares problem, has low computational cost and makes possible an updating of the principal components as new observations become available.

Once the matrix W is computed a reduction of the input dimensionality is obtained by transforming the input matrix X into the matrix $Z = XW^T$ and by transforming the regression problem of dimensionality n into a problem of dimensionality m in the space of principal components.

At this step the question arises of how to choose m. The techniques more commonly used rely either on the absolute values of the eigenvalues or on procedures of cross-validation [8].

4 The Lazy Learning algorithm

In supervised learning literature a possible way to classify learning techniques relies on the dichotomy: local memory-based versus global methods. Global modeling builds a single functional model of the dataset. This has traditionally been the approach taken in neural networks [2] and other form of non-linear statistical regression. The available dataset is used by a learning algorithm to produce a model of the mapping and then the dataset is discarded and only the model is kept. Local algorithms defer processing of the dataset until they receive request for information (e.g. prediction or local modeling). The classical nearest neighbor method is the original approach to local modeling. A database of observed input-output data is always kept and the estimate for a new operating point is derived from an interpolation based on a neighborhood of the query point.

The data mining architecture proposed in this paper adopts the Lazy Learning (LL) algorithm proposed by [1], an instance of the local modeling approach that, on a query-by-query basis, tunes the number of neighbors using a local cross-validation criterion. For a detailed description of the approach see also [5] et references therein. The LL algorithm, publicly available in a MATLAB and R implementation[1], proved to be successful in many problems of non-linear data analysis and time series prediction [5, 3, 4].

[1] http://iridia.ulb.ac.be/∼lazy

This paper illustrates and validates the use of Lazy learning for the task of mining sensor network data. The author deems that this algorithm presents a set of specific features which makes of it a promising tool in the sensor network context:

The reduced number of assumptions. Lazy Learning assumes no a priori knowledge on the process underlying the data. For example, it makes no assumption on the existence of a global function describing the data and no assumptions on the properties of the noise. The only available information is represented by a finite set of input/output observations. This feature is particularly relevant in real datasets where problems of missing features, non stationarity and measurement errors make appealing a data-driven and assumption-free approach.

On-line learning capability. The Lazy Learning method can easily deal with on-line learning tasks where the number of training samples increases or the set of input variables changes with time. In these cases, the adaptiveness of the method is obtained by simply adding new points to the stored dataset or restricting the analysis to the accessible inputs. This property makes the technique particularly suitable for monitoring problems where the number of samples increases with time or the set of available signals may vary due to sensor malfunctioning and/or bad communications.

Modeling non-stationarity. LL can deal with time-varying configurations where the stochastic process underlying the data is non-stationary. In this case, it is sufficient to interpret the notion of neighborhood not in a spatial way but both in a spatial and temporal sense. For each query point, the neighbors are no more the samples that have similar inputs but the ones that both have similar inputs and have been collected recently in time. Therefore, the time variable becomes a further precious feature to consider for accurate prediction.

5 A two-layer architecture for mining sensor network data

The conventional techniques of dimensionality reduction by PCA described in Section 3 aim to reduce the collinearity or linear relationships existing between the different inputs. This technique creates new variables obtained by combining linearly the original inputs. Typically, this linear combination assigns a weight different from zero to all the original inputs.

Consider now the idea of applying the PCA to a problem of regression where data are generated by a sensor network $\mathcal{S}$ with a large number of sensing units S. Although the PCA allows a reduction of the input space from S to m and possibly an improvement of the prediction accuracy, the resulting prediction model needs the values of all the S sensing units in order to perform its computation. This requires inevitably a centralized architecture where all the sensors are required to transmit their measures to the central server.

A distributed architecture cannot take advantage of this way of performing dimensionality reduction. Our approach consists instead in applying the PCA technique not to the whole of sensors but in a modular fashion to subsets made of neighboring sensing units.

Suppose that a partition $\mathcal{P}$ of the S sensors into C clusters of neighboring sensing units is available. Let n_c, $c = 1, \ldots, C$, be the number of sensing units contained into the cth cluster.

The algorithm we propose consists into applying C separate recursive PAST computations to each of the cluster. Let m_c be the number of principal components which are retained for each cluster and z_c the $[m_c, 1]$ vector of transformed variables returned by the PCA applied to the cth cluster.

The original supervised learning problem (2.2) whose input space has dimensionality S is now replaced by a new supervised learning problem featuring an input space $[z_1, \ldots, z_C]$ of dimensionality $\sum_{c=1}^{C} m_c$.

A cross-validation procedure can be adopted to assess the quality of the prediction model trained in the transformed space.

This assessment figure can be used as a measure of quality of the repartition $\mathcal{P}$. It is then possible to implement an iterative procedure that explores the space of possible repartitions aiming to find the one with the best prediction accuracy. This optimization procedure could be turned into a multi-criteria optimization by taking into account together with the prediction accuracy also a measure of the communication cost of the repartition under analysis.

Note that the outcome of the resulting procedure is a repartition of the nodes of the sensor network into C clusters, a procedure for aggregating locally the measures of the n_c nodes composing the cth cluster and a model at the server level which returns the high-level prediction.

6 Experimental results

The two-layer adaptive architecture has been tested on a simulated dataset generated by solving numerically the Helmholtz equation, an elliptic partial differential

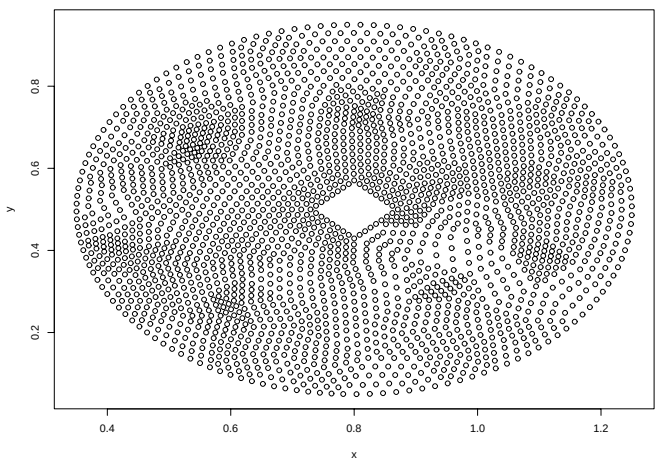


Figure 2: The distribution of the sensing units

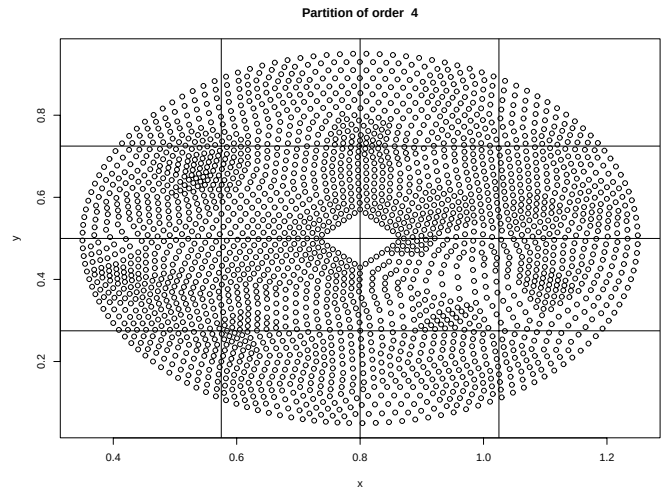


Figure 3: The uniform partition $\mathcal{P}^{(4)}$ of the sensor network in $4^2 = 16$ clusters.

equation,

$$\nabla^2 u + k^2 u = 0 \tag{6.6}$$

where $u(P_x, P_y, t) = s(t)$ is the value of scalar field in the point $[P_x, P_y]$ at time t and k is the wave number.

The Helmholtz equation governs some important physical phenomena, including the potential in time harmonic acoustic and electromagnetic fields. In our example the equation is used to model the waves reflected from a square object in a homogeneous medium. The wave number is related to the wavelength λ by the relation $k = \frac{2\pi}{\lambda}$.

We perform several simulations with 30 different wave numbers ranging from 1 to 146. For each wave number we collect the value of the u field in 50 time instants. Altogether, we collected $N = 1500$ measures of the wave field in a mesh of $S = 2732$ points depicted in Figure 2.

We formulate a regression problem where the goal is to predict the wave number k on the basis of the S measurements at time t returned by the simulated sensor network. The regression is performed on the output $y = k + \varepsilon$ that is a corrupted version of the signal k obtained by adding to k a random gaussian additive noise with standard deviation $\sigma_\varepsilon = \sigma_k/3$.

We consider a sequence of partitions $\{\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \ldots, \}$ where $\mathcal{P}^{(d)}$ is an uniform lattice with $d-1$ divisions along each dimension. This means that the partition $\mathcal{P}^{(d)}$ decomposes the sensor field into $C = d^2$ clusters. For illustration, the partition $\mathcal{P}^{(4)}$ is reported in Figure 3. Note that the partition $\mathcal{P}^{(1)}$ is equivalent to a centralized configuration where all sensors transmit their unprocessed measures to the central server.

Given a partition, a recursive PCA is performed on each cluster to return the first $m_c = 2$ principal components. Then the resulting aggregated signals are transmitted to the data mining server which performs the regression modeling by using the adaptive Lazy Learning algorithm described in Section 4. The quality of the data mining step is assessed by a ten-fold cross validation strategy. At each run of the cross-validation, we use $N_{tr} = 1350$ samples for the training set and the remaining $N_{ts} = 150$ samples for the test set. The prediction accuracy is computed by averaging the Normalized Mean Square Error (NMSE) for the test sets over all ten runs of the cross-validation. Note that NMSE is always a positive quantity and that values smaller than one denotes an improvement wrt the simplest predictor, i.e. the sample average.

We perform three experiments:

1. The first experiment deals with the centralized configuration where all sensor measurements are transmitted with no processing to the data mining server. We perform a recursive PCA with an increasing number m of principal components. The NMSE results for $m = 2, \ldots, 7$ after that N_{tr} samples have been processed are reported in Table 1. This table serves as reference for the results obtained in the modular case.

2. This experiment assesses the prediction accuracy of 6 uniform partitions $\{\mathcal{P}^{(2)}, \ldots, \mathcal{P}^{(7)}\}$ of the whole

m	1	2	3	4	5	6
NMSE	0.782	0.363	0.257	0.223	0.183	0.196

Table 1: Centralized configuration: NMSE for different numbers m of principal components. NMSE is evaluated through a 10-fold cross-validation procedure after N_{TR} examples have been processed.

	$\mathcal{P}^{(2)}$	$\mathcal{P}^{(3)}$	$\mathcal{P}^{(4)}$	$\mathcal{P}^{(5)}$	$\mathcal{P}^{(6)}$	$\mathcal{P}^{(7)}$
NMSE	0.140	0.118	0.118	0.118	0.116	0.114

Table 2: Modular configuration: NMSE obtained with six different partitions of the $S = 2372$ sensors. NMSE is evaluated through a 10-fold cross-validation procedure after N_{TR} examples have been processed.

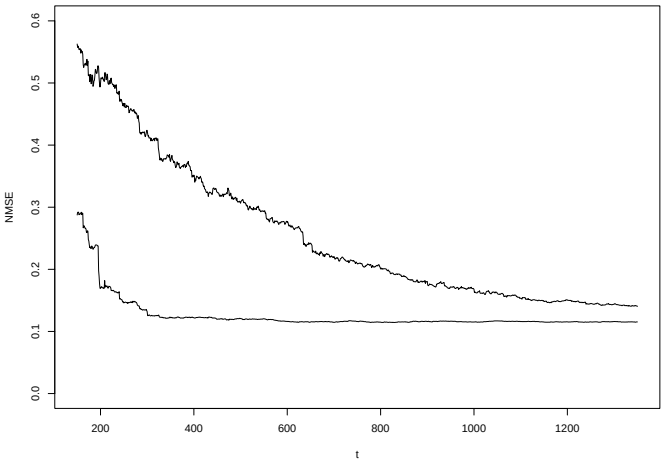


Figure 4: The sequential evolution of the NMSE for the partition $\mathcal{P}^{(2)}$ (upper line) and $\mathcal{P}^{(5)}$ (lower line). The NMSE at each instant t is an average over the ten runs of the cross validation procedure. The x-axis report the number of observations taken into consideration.

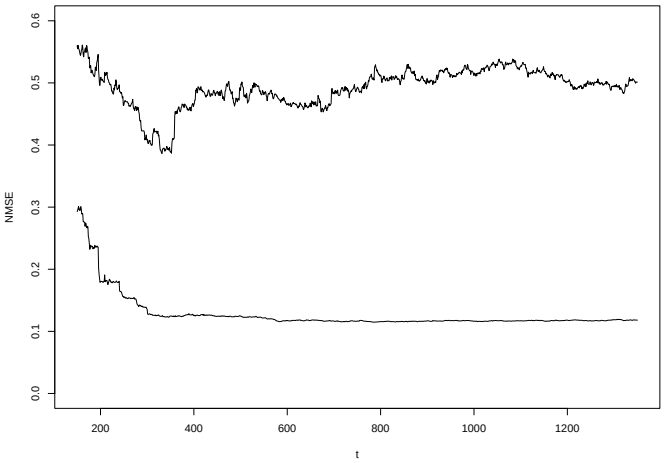


Figure 5: The sequential evolution of the NMSE for the partition $\mathcal{P}^{(2)}$ (upper line) and $\mathcal{P}^{(5)}$ (lower line) in front of random malfunctioning. The NMSE at each instant t is an average over the ten runs of the cross validation procedure. The x-axis report the number of observations taken into consideration.

set of sensors. Table 2 reports the NMSE of the test set after that N_{tr} samples have been processed. Since the data processing is carried out in a recursive fashion, we can also analyze the evolution of the NMSE for the test set, as more observations reach the mining server. Figure 4 reports the evolution of the Normalized Mean Square Error (NMSE) for the partition $\mathcal{P}^{(2)}$ and the partition $\mathcal{P}^{(5)}$.

3. The third experiment assesses the degradation of the accuracy in front of random malfunctioning of the sensors. We consider the 6 partitions assessed in the first experiment and we analyze the evolution of the NMSE when we simulate the occurrence of sensor faults. We assume that at each observation there is a 10% probability that a sensing unit is switched off and a 1% probability that one of the aggregating unit becomes out of order. Note that each malfunctioning is permanent and a sensing unit which breaks down cannot be reactivated. In data analysis terms, this is equivalent to the disappearance of some input variables for the recursive PCA procedure and the lazy learning algorithm, respectively. The NMSE after the processing of N_{tr} observations is reported in Table 3. These figures as well as the evolution of NMSE during the processing of N_{tr} observations (Figure 5), show that the degradation is negligible if the ratio between failure probability and unit number is not too high. This limitation in the system reliability is illustrated in the case of partition $\mathcal{P}^{(2)}$ (Figure 5) for which a dramatic loss in accuracy is observed. By increasing the number of clusters, one increases the robustness of the architecture, which thanks to the recursive feature of its components is able to react accordingly to faults and malfunctioning.

	$\mathcal{P}^{(2)}$	$\mathcal{P}^{(3)}$	$\mathcal{P}^{(4)}$	$\mathcal{P}^{(5)}$	$\mathcal{P}^{(6)}$	$\mathcal{P}^{(7)}$
NMSE	0.501	0.132	0.119	0.116	0.116	0.117

Table 3: NMSE obtained with six different partitions of the $S = 2372$ sensors in front of random malfunctioning. NMSE is evaluated through a 10-fold cross-validation procedure after N_{TR} examples have been processed.

7 Conclusions and future work

The paper proposed an adaptive methodology to mine data in large sensor networks. Previous works focused mainly on addressing issues of energy and transmission bandwidth reduction independently of the sensing task to be performed. This paper advocates the idea that the structure of the processing architecture of a sensor network must take into account also criteria related to the accuracy and quality of the data mining task. This means that the organization of the same sensor network may change according to the type of sensing task (e.g. classification or prediction) and the required quality and precision.

The results shown in this paper are preliminary but open the way to further research whose main lines can be summarized as follows:

- Extensions to non simulated data.
- Combination of accuracy based criteria with energy related cost functions in a multi-criteria configuration procedure.
- Assessment of more sophisticated clustering policies to partition the sensor field.
- Combination of spatial and temporal local correlations.
- Test of nonlinear compression techniques, like ICA.
- Adoption of sequential robust estimation techniques.

References

[1] M. Birattari, G. Bontempi, and H. Bersini. Lazy learning meets the recursive least-squares algorithm. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *NIPS 11*, pages 375–381, Cambridge, 1999. MIT Press.

[2] C. M. Bishop. *Neural Networks for Statistical Pattern Recognition*. Oxford University Press, Oxford, UK, 1994.

[3] G. Bontempi. *Local Learning Techniques for Modeling, Prediction and Control*. PhD thesis, IRIDIA- Université Libre de Bruxelles, 1999.

[4] G. Bontempi, M. Birattari, and H. Bersini. Local learning for iterated time-series prediction. In I. Bratko and S. Dzeroski, editors, *Machine Learning: Proceedings of the Sixteenth International Conference*, pages 32–38, San Francisco, CA, 1999. Morgan Kaufmann Publishers.

[5] G. Bontempi, M. Birattari, and H. Bersini. A model selection approach for local learning. *Artificial Intelligence Communications*, 121(1), 2000.

[6] S. Goel and T. Imielinski. Prediction-based monitoring in sensor networks: Taking lessons from mpeg. *ACM Computer Communication Review*, 31(5), 2001.

[7] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[8] A. Hyvarinen, J. Karhunen, and E. Oja. *Independent Component Analysis*. Wiley, 2001.

[9] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[10] L. Subramanian and R. H.Katz. An architecture for building self-configurable systems. In *IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC 2000)*, 2000.

[11] B. Yang. Projection approximation subspace tracking. *IEEE Transactions on Signal Processing*, 43(1):95–107, 1995.

Data mining in wireless sensor networks based on artificial neural-networks algorithms

Andrea Kulakov
Computer Science Department
Faculty of Electrical Engineering
Skopje, Macedonia

kulak@etf.ukim.edu.mk

Danco Davcev
Computer Science Department
Faculty of Electrical Engineering
Skopje, Macedonia

etfdav@etf.ukim.edu.mk

Abstract

Up to date the algorithms for in-sensor-network data processing were modifications of regression methods like multidimensional data series analysis. In this paper we show that some of the algorithms developed within the artificial neural-networks tradition can be easily adopted to wireless sensor network platforms and will meet several aspects of the constraints for data mining in sensor networks like: limited communication bandwidth, limited computing resources, limited power supply, and the need for fault-tolerance. The analysis of the dimensionality reduction obtained from the outputs of the neural-networks clustering algorithms shows that the communication costs of the proposed approach are significantly smaller, which is an important consideration in sensor-networks due to limited power supply.

In this paper we will present two possible implementations of the ART and FuzzyART neural-networks algorithms, which are unsupervised learning methods for categorization of the sensory inputs. They are tested on a data obtained from a set of several motes, equipped with several sensors each. Results from simulations of purposefully defective sensors show the data robustness of these architectures.

1 Introduction

A centralized data clustering in wireless sensor networks is difficult and often not scalable because of various reasons such as limited communication bandwidth and limited power supply for running the sensor nodes. It is also inefficient given that sensor data has significant redundancy both in time and in space. In cases when the application demands compressed summaries of large spatio-temporal sensor data and similarity queries, such as detecting correlations and finding similar patterns, the use of a neural-network algorithm is a reasonable choice.

The development of the wireless sensor networks is accompanied by several algorithms for data processing which are modified regression techniques from the field of multidimensional data series analysis in other scientific fields, with examples like nearest neighbor search, principal component analysis and multidimensional scaling (e.g. [7], [10]). We argue that some of the algorithms well developed within the neural-networks tradition for over 40 years, are well suited to fit into the requirements imposed to sensor networks for: simple parallel distributed computation, distributed storage, data robustness and auto-classification of sensor readings.

Auto-classification of the sensor readings is important in sensor networks since the data obtained with them is with high dimensionality and very immense, which could easily overwhelm the processing and storage capacity of a centralized database system. On the other hand, the data obtained by the sensor networks are often self-correlated over time, over space and over different sensor inputs, due to the nature of the phenomena being sensed which is often slowly changing, due to the redundant sensor nodes dispersed near each other, and due to the fact that often the sensor readings are correlated over different modalities sensed at one node (e.g. sound and light from cars in traffic control application).

Neural-networks algorithms, on the other hand, use simple computations and do not represent big burden to memory. The proposed adaptations of the ART neural networks models can be easily parameterized according to user needs for greater or lower level of details of the sensor data.

Up to date, the only application of neural-networks algorithms for data processing in the field of sensor networks is the work of Catterall et al. [5] where they have slightly modified the Kohonen Self Organizing Maps model. Even this application was presented to a different kind of audience at a conference for Artificial Life. This has additionally motivated us to bring closer the work done in the field of Artificial Neural Networks to the community of researchers working in the field of Sensor Networks, since some of the problems for the processing of the sensory input data are similar.

Unsupervised learning Artificial Neural Networks typically perform dimensionality reduction or pattern clustering. They are able to discover both regularities and irregularities in the redundant input data by iterative process of adjusting weights of interconnections between a large numbers of simple computational units (called artificial neurons). As a result of the dimensionality reduction obtained easily from the outputs of these algorithms, lower communication costs and thus bigger energy savings can also be obtained.

A neural network algorithm can be implemented in the tiny platform of Smart-It units, which are kind of sensor nodes or motes. Thus instead of reporting the raw-data, each Smart-It unit can send only the cluster number where the current sensory input pattern has been classified. In that way a huge dimensionality reduction can be achieved depending on the number of sensor inputs in each unit. In the same time communication savings will benefit from the fact that the cluster number is a small binary number unlike sensory readings which can be several bytes long real numbers converted from the analog inputs.

Since the communication is the biggest consumer of the energy in the units, this leads to bigger energy savings as well.

In the paper we will review first the ART1 [1] and FuzzyART [2] models. Later some related work will be considered and after that our proposal of three different kinds of architectures for incorporating the Artificial Neural Networks into the small Smart-It units' network will be given. Shortly we will present the hardware platform that has been originally used to obtain the data that we later used to test our proposal and we will give some results of the classifications of the data within different architectures. We will also give results from the simulations where we have purposefully made one of the input sensors malfunctioning in order to show the data robustness of our approach. Finally we will give some discussions and directions for future work.

2 ART and FuzzyART algorithms

Several models of unsupervised Artificial Neural Networks have been proposed like Multi-layer Perceptron (MLP), Self-Organizing Maps (SOMs), and Adaptive Resonance Theory (ART) ([8] and [9]). Out of these we have chosen the ART models for implementation in the field of sensor networks because they do not constrain the number of different categories in which the input data will be clustered. Although the later extensions of MLP and SOMs involve the principle of incrementally growing structure, their topological self-organization is possible only with so called off-line learning cycle separate from the classification cycle. Having two separate cycles is inconvenient in the presence of potentially unlimited stream of input data with no reliable method of choosing the suitably representative subset for a learning cycle. ART algorithms offer another example of topological self-organization of data but they can adapt structure quickly in the so called fast-learning mode explained later.

Adaptive Resonance Theory (ART) has been developed by Grossberg and Carpenter for pattern recognition primarily. Models of unsupervised learning include ART1 [1] for binary input patterns and FuzzyART [2] for analog input patterns.

ART networks develop stable recognition codes by self-organization in response to arbitrary sequences of input patterns. They were designed to solve the so called stability-plasticity dilemma: how to continue to learn from new events without forgetting previously learned information. ART networks model several features such as robustness to variations in intensity), detection of signals mixed with noise, and both short- and long-term memory to accommodate variable rates of change in the environment. There are several variations of ART-based networks: ART1 (three-layer network with binary inputs), Fuzzy ART (with analog inputs, representing neuro-fuzzy hybrids which inherit all key features of ART), their supervised versions ARTMAP and FuzzyARTMAP and many others. ARTMAP models [3], for example, combine two unsupervised modules to carry out supervised learning.

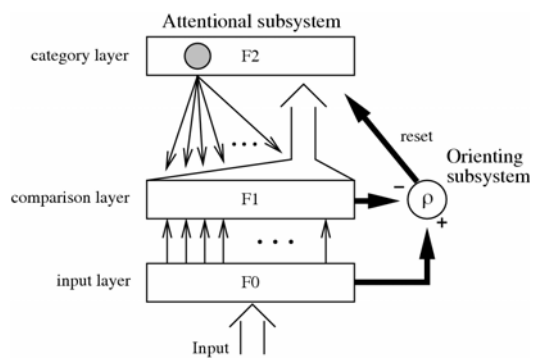


Figure 1: Architecture of the ART network.

In Figure 1 typical representation of an ART Artificial Neural Network is given. Winning F2 category nodes are selected by the attentional subsystem. Category search is controlled by the orienting subsystem. If the degree of category match at the F1 layer is lower than the so called vigilance level ρ, a reset signal will be triggered, which will deactivate the current winning F2 node for the period of presentation of the current input.

An ART network is built up of three layers: the input layer (F0), the comparison layer (F1) and the recognition layer (F2) with N, N and M neurons, respectively. The input layer stores the input pattern, and each neuron in the input layer is connected to its corresponding node in the comparison layer via one-to-one, non-modifiable links. Nodes in the F2 layer represent input categories. The F1 and F2 layers interact with each other through weighted bottom-

up and top-down connections that are modified when the network learns. There are additional gain control signals in the network (not shown in Figure 1) that regulate its operation, but those will not be detailed here.

The learning process of the network can be described as follows: At each presentation of a non-zero binary input pattern x ($x_j \in \{0, 1\}$; $j = 1, 2, \ldots, N$), the network attempts to classify it into one of its existing categories based on its similarity to the stored prototype of each category node. More precisely, for each node i in the F2 layer, the bottom-up activation T_i is calculated, which can be expressed as

$$T_i = \frac{|\mathrm{w}_i \cap \mathrm{x}|}{\beta + |\mathrm{w}_i|} \quad i = 1, \ldots, M \qquad (1)$$

where $|\cdot|$ is the norm operator (for a vector u it is: $|\mathrm{u}| \equiv \sum_{j=1}^{N} u_j$), w_i is the (binary) weight vector or prototype of category i, and $\beta > 0$ is a parameter. Then the F2 node I that has the highest bottom-up activation, i.e. $T_I = \max\{T_i \mid i = 1, \ldots, M\}$, is selected (realizing so called winner-takes-all competition). The weight vector of the winning node (w_I) will then be compared to the current input at the comparison layer. If they are similar enough, i.e. if they satisfy the matching condition:

$$\frac{|\mathrm{w}_I \cap \mathrm{x}|}{|\mathrm{x}|} \geq \rho \qquad (2)$$

where ρ is a system parameter called vigilance ($0 < \rho \leq 1$), then the F2 node I will capture the current input and the network learns by modifying w_I:

$$\mathrm{w}_I^{new} = \eta(\mathrm{w}_I^{old} \cap \mathrm{x}) + (1-\eta)\mathrm{w}_I^{old} \qquad (3)$$

where η is the learning rate ($0 < \eta \leq 1$) (the case when $\eta = 1$ is called "fast learning"). All other weights in the network remain unchanged.

If, however, the stored prototype w_I does not match the input sufficiently, i.e. if the condition (2) is not met, the winning F2 node will be reset (by activating the reset signal in Figure 1) for the period of presentation of the current input. Then another F2 node (or category) is selected with the highest T_i, whose prototype will be matched against the input, and so on. This "hypothesis-testing" cycle is repeated until the network either finds a stored category whose prototype matches the input well enough, or allocates a new F2 node in which case learning takes place according to (3).

As a consequence of its stability-plasticity property, the network is capable of learning "on-line", i.e. refining its learned categories in response to a stream of new input patterns, as opposed to being trained "off-line" on a finite training set.

The number of developed categories can be controlled by setting the vigilance ρ: the higher the vigilance level, the larger number of more specific categories will be created. At its extreme, if $\rho = 1$, the network will create a new category for every unique input pattern.

FuzzyART is an analog version of the ART1 algorithm which takes analog inputs and classifies them in a similar way as ART1. The main ART1 operations of category choice (1), match (2), and learning (3) translate into Fuzzy ART operations by replacing the ordinary set theory intersection operator $\cap$ of ART1 with the fuzzy set theory conjunction MIN operator $\wedge$.

In FuzzyART (but as well in ART1), complement coding of the input vector prevents a type of category proliferation that could otherwise occur when weights erode. Complement coding doubles the dimensionality of an input vector $\mathrm{b} \equiv (b_1, \ldots, b_N)$ by concatenating the vector b with its complement b^c. The input to the FuzzyART network (F0 in Figure 1) is then a 2N-dimensional vector: $\mathrm{I}=\mathrm{B}\equiv(\mathrm{b}, \mathrm{b}^c)$, where $(\mathrm{b}^c)_i \equiv (1- b_i)$. If b represents input features, then complement coding allows a learned category representation to encode the degree to which each feature is consistently absent from the input vector, as well as the degree to which it is consistently present in the input vector, when that category is active. Because of its computational advantages, complement coding is used in nearly all ART applications, and we have used it in our models as well.

The strengths of the ART models include its unique ability to solve a stability-plasticity dilemma, extremely short training times in the fast-learning mode, and an incrementally growing number of clusters based on the variations in the input data. The network runs entirely autonomously; it does not need any outside control, it can learn and classify at the same time, provides fast access to match results, and is designed to work with infinite stream of data. All these features make it an excellent choice for application in wireless sensor networks.

3 Related work

As we mentioned in the introduction, Catterall et al. [5] have slightly modified the Kohonen Self Organizing Maps (SOMs) model. Kohonen SOMs and ART models are similar in a way that they are both prototype-based networks where they both create a set of prototypes and then compare an unknown input vector with the stored prototypes in order to implement the mapping or clustering.

The advantages of SOMs over other Artificial Neural Network models include the ability to provide real-time nearest-neighbor response as well as topology-preserving mapping of the input data. Still, the limitations are extensive off-line learning and most importantly, the need of a predefined map size, i.e. a fixed number of output clusters or categories. In [5] a rather simple and straightforward implementation of the Kohonen neural-network architecture is used, where one cluster unit corresponds to one Smart-It hardware unit.

In many real-world situations, there is no a priori information on variability present in the data stream, so we

can not determine in advance the required number of output clusters in which the input patterns will fit. Thus this straightforward implementation of the Kohonen neural network seems rather rudimentary and the only justification for it can be the mere possibility to apply some principles of Artificial Neural Networks for data processing in wireless sensor networks.

DIMENSIONS [7] is another model where they treat the problems of data storage and handling in sensor systems. DIMENSIONS incorporates spatio-temporal data reduction to distributed storage architectures, introduces local cost functions to data compression techniques, and adds distributed decision making and communication cost to data mining paradigms. It provides unified view of data handling in sensor networks incorporating long-term and short-term storage with increasing lossy compression over time, multi-resolution data access using different wavelet parameters at different hierarchical levels, and spatio-temporal pattern mining.

Several aspects of our approach are common to DIMENSIONS such as spatio-temporal data reduction, limited communication costs, long-term and short-term storage and hierarchical access with different level of details, although these aspects are achieved by completely different algorithms.

4 Proposed architectures of sensor networks

Three types of network architectures are proposed. The results of the classifications of a real-world data will be given later for each of the architectures.

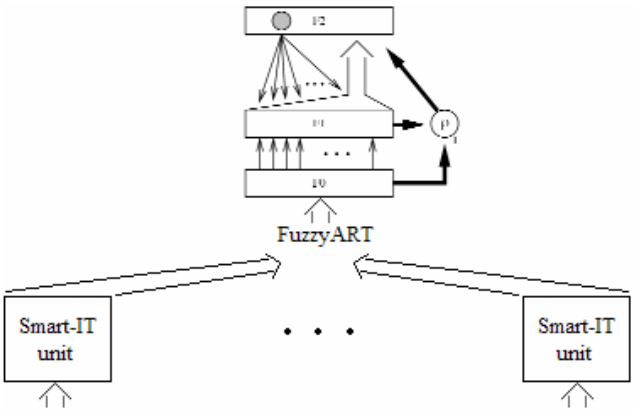


Figure 2: Clusterhead collecting all sensor data from its cluster of units

4.1 One Clusterhead collecting all sensor data First we have made this architecture (Figure 2) to compare the work of Catterall et al. (which used Kohonen SOMs), in order to show that ART model can be used straightforwardly instead of SOMs. This model brings advantages in that we do not have to fix in advance the number of clusters (categories) that the network should learn to recognize. Here the Smart-It units send the sensory readings to one of them chosen to be a Clusterhead, where a FuzzyART network is implemented.

The sensor data can be classified with different vigilance parameter ρ, thus providing a general overall view on the network, (smaller ρ) or more and more detailed views of the network (greater ρ). Depending on the level of details needed at the moment, some of the Clusterheads can be adjusted to different vigilance parameters and later can be queried depending on that parameter.

4.2 Clusterhead collecting only clustering outputs from the other units Each Smart-It unit has FuzzyART implementations classifying only its sensor readings. One of the Smart-It units can be chosen to be a Clusterhead collecting and classifying only the classifications obtained at other units. Since the clusters at each unit can be represented with integer values, the neural-network implementation at the Clusterhead is ART1 with binary inputs. The categories in each unit do not have to be coordinated in any way among each other.

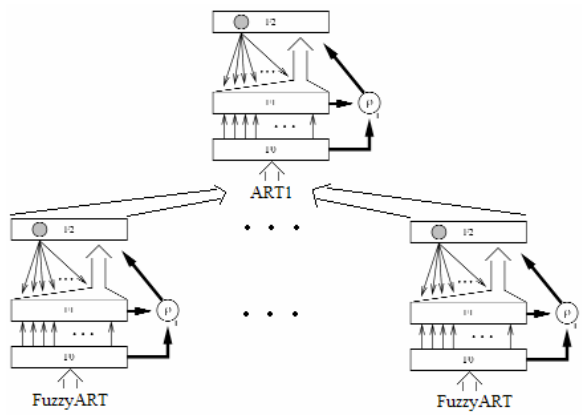


Figure 3: One Clusterhead collecting and classifying the data after they are once classified at the lower level

With this architecture a great dimensionality reduction can be achieved depending on the number of sensor inputs in each unit (in our case it's a 6-to-1 reduction). In the same time communication savings benefit from the fact that the cluster number is a small binary number unlike raw sensory readings which can be several bytes long real numbers converted from the analog inputs.

If the number of sensors in each unit is n, the Clusterhead collects data from k units, and the number of different categories in each unit can be represented by c-byte integer, while the sensor readings are real numbers represented with r bytes, then the communication saving can be calculated as:

$$\frac{n \cdot k \cdot r}{k \cdot c} = \frac{n \cdot r}{c}$$

Since the communication is the biggest consumer of the energy in the units, this leads to bigger energy savings as well.

Another benefit from this architecture is the fact that we can view the classifications at the Clusterhead as an indication of the spatio-temporal correlations of the input data.

5 Hardware platform

The platform for the experiments, from which the data analyzed in this paper were obtained, is a collection of 'Smart-Its' (see [5], for more details). One Smart-It unit embodies a *sensor module*, and a *communication module*, which are interconnected. The core of sensor module is a PIC 16F877 microcontroller clocked at 20 MHz, which offers 384 bytes of data memory and 8Kx14 bits of memory. The sensor module consists of a light sensor, a microphone, 2 accelerometers, a thermometer and a pressure sensor. An RF stack provides wireless communication, at a maximum rate of 125 kbps, which only supports broadcast.

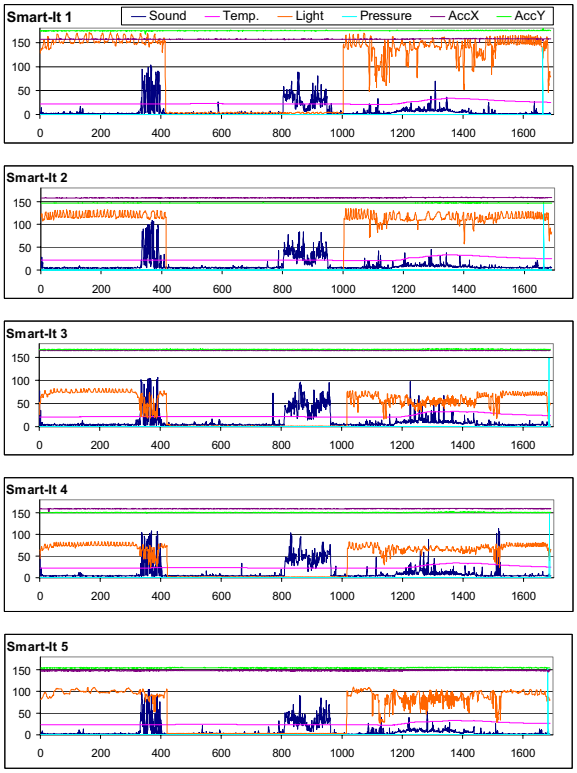


Figure 4: Datasets with sensor values from each of the Smart-It units during several states of the environment ('contexts'): Lights on (1-330), talking people nearby while lights remain on (331-400), lights turned off (401-800), talking people nearby while light remain turned off (801-1000), and heating on (1090-1400).

6 Experimental results

The data used in these experiments were provided courtesy of Catterall et al[1]. All results presented here were produced using datasets containing real-world data. The five datasets (one for each Smart-It) are visualized by time series plots in Figure 4. Note that, although the sensor data are very similar (as the units are physically close to each other), they are not *exactly* the same.

All the experiments were conducted with complement coding of the input vector and fast learning mode. Figure 5 shows one possible classification of these input data in an architecture presented in Figure 2 with vigilance level ρ=0.93. In subsequent chart, the axis shows the recognized category of the sample input, while the ordinate shows the sample sensory input.

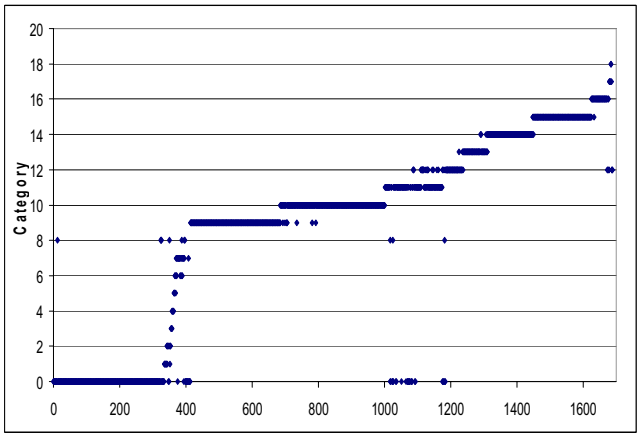


Figure 5: One possible classification of the input data

For testing the data robustness of the models, we have synthetically made one of the sensors at a time defective in way that it gives either a zero constant signal or a random measurement signal. Training was done with vigilance set to 0.93, while testing was done with vigilance set to 0.90. In Figure 6 the effects of the representative sensor errors are shown (sensor numbered 12 and sensor numbered 17, out of 30), where with the ovals are highlighted the regions where the classifications differ from the case when all sensors are functioning correctly. In Regions 1 and 2, the classification of the case when the sensor number 12 gives random values differs from the regular case, while in Region 3, the defective sensor number 17 results in different classification than the regular case. In Region 2, the cases when the 17th sensor gives random or zero values also results in different classifications.

[1] The data files are available for download at this website: http://www.comp.lancs.ac.uk/~catterae/alife2002/

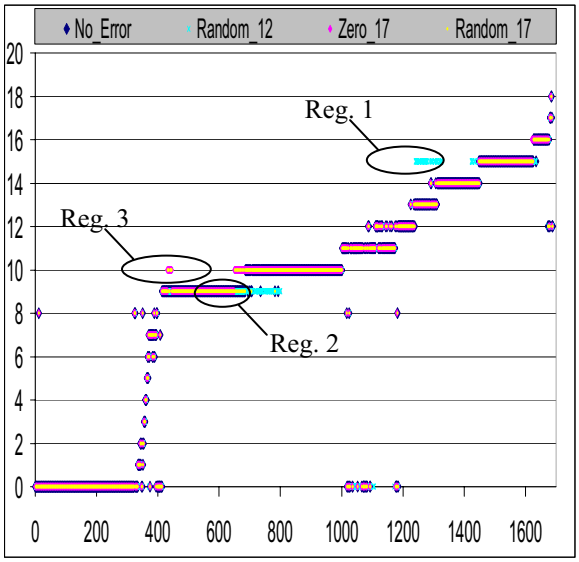


Figure 6: Different classifications when some of the sensors are defective giving zero or random values.

If we want to see the difference when providing different degrees of granularity of the input data, namely for different values of the vigilance parameter ρ (ranging from 0.7 up to 0.99 in our experiments) we get different number of output categories (from 2 up to 370) for the 1700 samples taken as a learning dataset, which can be seen in Table 1. Some of the possible classifications for different vigilance parameter can be seen in Figure 7.

ρ	0.70	0.85	0.9	0.93	0.95	0.97	0.98	0.99
Number of categories	2	3	8	19	36	87	151	370

Table 1 Number of different categories into which the same data is classified depending on the vigilance parameter

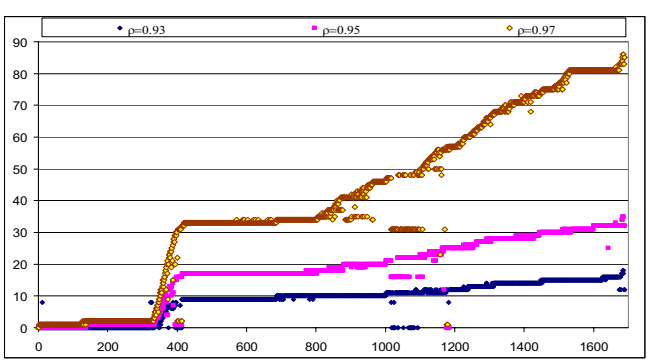


Figure 7: Different numbers of categories appear when we classify data at different levels of details depending on the vigilance parameter ρ

For the second architecture (Figure 3) we have also conducted experiments with the original data and with the synthetically made erroneous data. In Figure 8 we give the results of the classifications of the Clusterhead collecting only the classifications from the other Smart-It units. The training was done with vigilance level of 0.80, while the testing with 0.70. The results show no significant difference among the classifications when all sensors are functioning correctly or when some of the sensors give only zero or random signal (in our case sensors number 12 and 17).

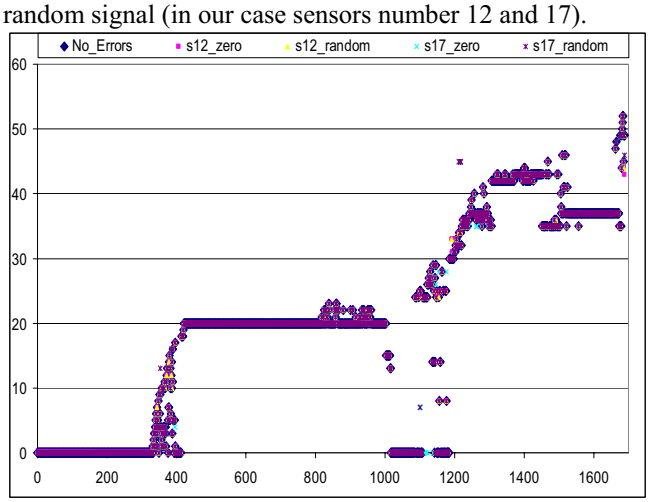


Figure 8: Results of the classifications show significant data robustness of the third architecture with one Clusterhead collecting only clustering outputs from the other units.

Counting the number of cases, it turned out that only 0.75% of the samples have a different classification than the case with no deliberate error in the signal, although the overall percentage of errors in the data were 3.3% (one out of 30 sensors was made erroneous).

7 Discussion

The second proposed architecture (Figure 3) with one Clusterhead collecting only clustering outputs from the other units can be generalized to a hierarchical cascade classification scheme where small Smart-It units at the lowest level will be grouped in small groups having one Clusterhead. Then several Clusterheads can be grouped and their outputs can be classified using a binary input ART1 classifier at a Clusterhead one-level higher and so on, up to a level where the classification will be read by a human user or stored in a database, after achieving a huge dimensionality reduction (see Figure 9).

If at each level classifications from k units are clustered into one Clusterhead (represented with the same number of bytes), the dimensionality reduction after l levels will be:

$$\frac{n \cdot r}{c} \cdot k^{l}$$

For future work we are also considering to apply the supervised learning versions of the ART algorithms, namely ARTMAP and FuzzyARTMAP where along with the sensor input vector, a vector of corresponding "right-answers" is obtained by the user (so called teacher), or possibly automatically from another system.

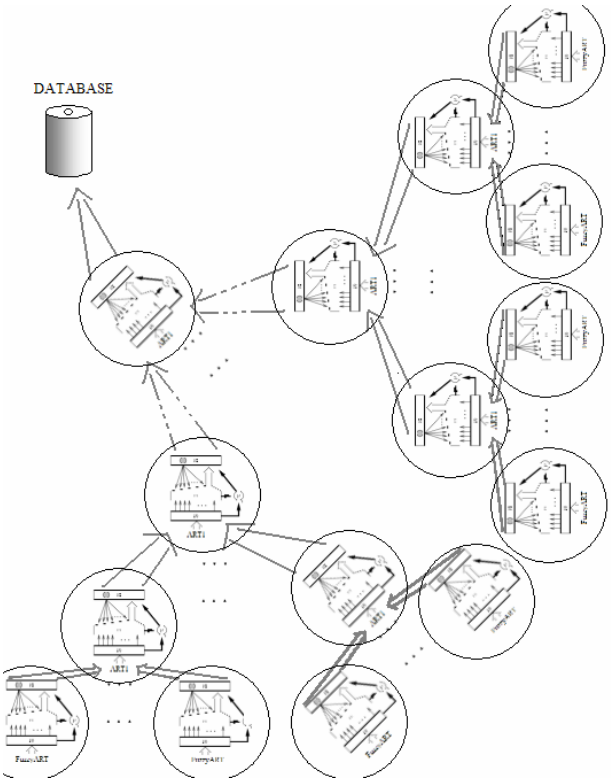


Figure 9: Hierarchical cascades of ART neural-network classifiers implemented in units of a sensor network

8 Conclusion

In this paper we have demonstrated a possible adaptation of one popular model of Artificial Neural Networks algorithm (ART model) in the field of wireless sensor networks. The positive features of the ART class algorithms such as simple parallel distributed computation, distributed storage, data robustness and auto-classification of sensor readings are demonstrated within two different proposed architectures.

One of the proposed architectures with one Clusterhead collecting only clustering outputs from the other units, provides a big dimensionality reduction and in the same time additional communication saving, since only classification IDs (small binaries) are passed to the Clusterhead instead of all input samples.

Results from the simulated deliberately erroneous sensors, where we imitate defective sensors giving only zero or random output, show that the model is robust to small variations in the input.

Data clustering algorithms for data spread over a sensor-network are necessary in many applications based on sensor-networks. The use of limited resources together with the distributed nature of the sensor-networks demands a fundamentally distributed algorithmic solution for data clustering. Hence, distributed clustering algorithms as the ones proposed in this paper, based on artificial neural-networks, seem useful for mining sensor-network data or data streams.

References

[1] CARPENTER, G.A., AND GROSSBERG, S., A massively parallel architecture for a self-organizing neural pattern recognition machine, Computer Vision, Graphics, and Image Processing, vol. 37, pp. 54-115, 1987.

[2] CARPENTER, G.A., GROSSBERG, S., AND ROSEN, D.B., Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system, Neural Networks, vol. 4, pp. 759-771, 1991.

[3] CARPENTER, G.A., GROSSBERG, S., MARKUZON, N., REYNOLDS, J.H., AND ROSEN, D.B., Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps, IEEE Transactions on Neural Networks, vol. 3, pp. 698-713, 1992.

[5] CATTERALL, E., VAN LAERHOVEN, K., AND STROHBACH, M., Self-Organization in Ad Hoc Sensor Networks: An Empirical Study, in Proc. of Artificial Life VIII, The 8th Int. Conf. on the Simulation and Synthesis of Living Systems, Sydney, NSW, Australia, 2002.

[6] FRITZKE, B., A Growing Neural Gas Network Learns Topologies, in Tesauro, G., Touretzky, D.S., and Leen, T.K. (eds.) Advances in Neural Information Processing Systems 7, MIT Press, Cambridge MA, USA, 1995.

[7] GANESAN, D., ESTRIN, D., HEIDEMANN, J., DIMENSIONS: Why do we need a new Data Handling architecture for Sensor Networks?, in Proc. Information Processing in Sensor Networks, Berkeley, California, USA, 2004.

[8] GROSSBERG, S. Adaptive pattern classification and universal recoding: I. parallel development and coding of neural feature detectors, Biological Cybernetics, vol. 23, pp. 121-134, 1976.

[9] GROSSBERG, S. Adaptive Resonance Theory in Encyclopedia Of Cognitive Science, Macmillan Reference Ltd, 2000.

[10] GUESTRIN, C., BODIK, P., THIBAUX, R., PASKIN M., AND MADDEN, S., Distributed Regression: an Efficient Framework for Modeling Sensor Network Data, in Proceedings of IPSN'04, April 26–27, 2004, Berkeley, California, USA, 2004.

Distributed Pre-Processing of Data on Networks of Berkeley Motes Using Non-Parametric EM

Ian Davidson * S. S. Ravi †

Abstract

Inexpensive sensor networks, such as those constructed using Berkeley motes, record many missing or absurd values due to low battery levels, transmission errors, sensor device errors and node failures. However, many mining algorithms do not easily handle data with missing values and the mining literature illustrates that removing records with missing values yields worse results than if missing values are intelligently filled in. A powerful method of filling in missing values is the expectation maximization (EM) algorithm which maximizes the complete (both observed and missing) likelihood. However, typical implementations of EM require a parametric model which is prohibitive for sensor networks as the M-step typically requires collecting and transmitting the expected values for missing data to a central base station. Furthermore, distributing either the E-step or the M-step onto the network is infeasible for parametric models as Berkeley motes have neither a hardware floating point unit (FPU) nor sufficient memory to implement them. In this paper, we develop a non-parametric version of EM specifically for sensor networks. We formally show that the E-step can be solved in polynomial time. Though the problem associated with the M-step is **NP**-complete, a straightforward heuristic is possible. Therefore, our approach provides an example of *generalized* EM. Our preliminary empirical results indicate that the algorithm can restore many of the missing values. Future work will change the likelihood function that EM maximizes to include appropriate mining tasks.

Keywords: Sensor Networks, Missing Values, Mining, Non-parametric, Expectation Maximization.

1 Introduction and Motivation

A sensor network consists of nodes, each of which is equipped with a set of sensors for collecting data from the environment and additional hardware that allows the nodes to communicate with each other in a wireless fashion [1, 16]. In our work, each node is a Berkeley mote that consists of a Mica2 radio board and a MTS300 sensor board which can be used to sense temperature, light intensity and the sound level. The transmission range of each node is not more than 100 feet in ideal situations (open area, no walls or other forms of interference). We treat the sensed values as binary with 0 (false) representing typical indoor readings and 1 (true) representing above typical values. Typical values for the sensed parameters are as follows: light intensity < 500 lumens, temperature < 25.0 degrees Celsius and sound level < 20 decibels. Collecting binary values is quite common for simple sensor networks that are designed to be deployed over areas that are in excess of a square mile [3].

In practice, a sensor network may report many missing or absurd values for some of the quantities measured by the sensors. This may be due to tranmission failures, faulty sensor readings, obstructions of the sensor devices and low battery levels. For example, with our Berkeley mote network in an indoor environment, we found that even when the motes are only about ten feet from the base station, over the course of three hours, approximately three percent of packets were lost. When there are walls or other structures between a mote and the base station, the packet loss is even greater. Furthermore, when polling light and temperature readings every five seconds over the course of three hours, five percent of temperature readings and four percent of light readings were absurd (i.e., represented values that are physically impossible). When the same experiments were conducted with the nodes in a room different from the base-station, the packet loss increases to 23%, and 26% of temperature readings and 21% of light readings were absurd or missing. Therefore, even though Berkeley motes offer great potential for wide-scale deployment, techniques to deal with missing and/or absurd values generated by such networks need to be developed before the sensor network data can be mined.

We could ignore records that contain missing values. However, many mining algorithms do not easily handle missing values. Throwing away records containing

*Department of Computer Science, University at Albany - State University of New York, Albany, NY 12222. Email: davidson@cs.albany.edu.

†Department of Computer Science, University at Albany - State University of New York, Albany, NY 12222. Email: ravi@cs.albany.edu.

missing values removes legitimate values as well. For example, in our experiments mentioned above, with the motes and the base-station being in different rooms, we would have thrown away on average 23% of all records per five second snap-shot of the network. Filling in missing values using even basic schemes leads to improved results [6]. An approach that is used in many commercial data mining tools is to fill in each missing value with the most likely value [7]. This is equivalent to calculating the likelihood probability distribution over the ith column's values (θ_i) from only the *observed data*. Then, the most probable value is chosen to fill in the missing value. Formally, if $y_{l,i}$ denotes the missing value of the lth record's i^{th} column, then $y_{l,i} = argmax_j P(y_{l,i} = j|\theta_i)$.

Better estimates of what the missing values should be are attainable if: 1) More complex models beyond marginal probabilities are used and 2) The missing data values influence the model selection. Formally, this involves calculating the *complete* likelihood (i.e., the likelihood of both the observed and missing data). However, when the missing data is part of the likelihood calculation, no tractable solution for maximum likelihood estimation exists. Instead, a common approach is to use the expectation maximization (EM) algorithm to converge to a local maxima of the complete data likelihood.

Unfortunately, the parametric form of the EM algorithm is not amenable to sensor networks as the second step (the M-step) involves transmitting the expectation of the missing values to a central location (i.e. the base station) for aggregation. Since the EM algorithm may take many iterations to converge, this approach may require many rounds of data transmission, leading to quick depletion of the battery power at the nodes. Furthermore, since motes lack FPU hardware, neither the E-step nor the M-step can be distributed onto the network with parametric models.

In this paper we propose a non-parametric version of EM specifically for sensor networks. Our approach is designed to minimize power consumption and the necessary computations can be distributed over the network. To our knowledge, only two other papers have outlined non-parametric version of EM [2, 17]. The paper by Caruna [2] presents an EM *style* algorithm which only involves one step while the paper by Schumitzky [17] allows only a very restricted model space. Furthermore, both approaches are not readily applicable to sensor networks. We shall focus on how to fill in the missing values so that the resulting sensor network data can be mined using a variety of techniques.

The remainder of the paper is organized as follows. We begin by overviewing parametric and non-parametric EM specifically for sensor networks and then discuss our particular non-parametric model. We formally show that in this model, the computational problem associated with the E-step can be solved in polynomial time and that the corresponding problem for the M-step is **NP**-complete. However, a simple heuristic for the M-step exists, thus our algorithm is an example of *generalized* EM. Other examples of generalized EM include the Baum Welch algorithm commonly used to learn hidden Markov models [4]. We then illustrate approximations that allow distribution of the E and M steps onto the sensor network. Our empirical results presented next, illustrate the usefulness of our algorithm. Finally, we discuss and conclude our work.

2 Parametric and Non-Parametric EM for Sensor Networks

When the sensor network reports a combination of observed (X) and missing (Y) values, our aim is to calculate the complete (missing and observed) maximum likelihood estimate $P(X, Y|\theta)$. This involves finding both the most probable model and actual values for the missing data. No tractable solution for this problem exists; instead, we attempt to calculate the *expectation* of the complete data likelihood $E_{P(Y|X,\theta)}[Y, X|\theta]$. To do this, we use the two step EM algorithm. The first step (the E-step) calculates the expectation of the missing nodes values ($P(Y|X, \theta)$). Given the expectations for the missing values, the second step (the M-step) calculates the value for θ that maximizes the expected complete data likelihood.

Instead of using a parametric model, we use a non-parametric model, namely the Ising spin-glass model, that is commonly used in image processing [5]. The model consists of a set $V = \{v_1, \ldots, v_n\}$ of nodes, one for each sensor node. To be consistent with the Ising spin-glass model, we assume that the nodes whose values are known take on one of the discrete values from $\{-1, +1\}$, instead of 0 and 1. Nodes whose values are missing are assumed to have the value '?'. Extensions that allow the nodes to take on many discrete values are possible and will be left for future work, though as mentioned earlier, it is typical to report binary values from Berkeley motes. Each node v_i has a set $N(v_i)$ of neighboring nodes. Therefore, the non-parametric model of the sensor network can be considered as the graph $G(V, E)$ which represents the nodes and their neighbors. Some nodes will have legitimate filled in values while others will have missing values or absurd values that can be treated as missing. In contrast to our non-parametric model, a parametric model could identify a subset of key nodes, with the sensor values at other nodes being modeled according to their distance to these key nodes and by a parametric probability

distribution (e.g. a Gaussian with parameters μ_x, σ_x, μ_y and σ_y). We now derive the probability density (mass) function for a sensor network using our non-parametric model. The the probability mass function ($P(G)$) of a particular configuration of sensor values is a function of the Hamiltonian ($H(G)$) of the underlying graph. Equations defining $H(G)$ and $P(G)$ are given below.

Definition 2.1. *Let $G(V, E)$ be an undirected graph. Suppose each node $v_i \in V$ is assigned a label $\ell_i \in \{+1, -1\}$.*

$$H(G) = -\sum_{\{v_i, v_j\} \in E} \ell_i \, \ell_j. \tag{2.1}$$

$$P(G) = 1 - \frac{2^{H(G)}}{2^{|E|}} \tag{2.2}$$

As can be seen, our probability density function is a function of the edges (E) in the graph and the node labels ($\ell_1, \ldots, \ell_n$). As in the Ising spin-glass model, we sometimes refer to $H(G)$ as the **energy** associated with the given configuration. Note that the smaller the value of energy, the larger is the probability of the corresponding configuration and that the smaller the number of conflicts between a node's value and its neighbors, the lower the energy. Our model, namely the graph structure, contains no continuous parameters. Hence we use the term non-parametric to describe the model.

3 Our Non-Parametric EM Algorithm

We present our non-parametric EM algorithm along with a sketch of how the E and M steps were derived. Later sections provide the formal details of these derivations.

A key problem with the EM algorithm is initialization. However, in our situation, we can choose an initial model (graph) that is obtained by connecting each node to every other node that is within its transmission distance (100 feet for Berkeley motes). Our EM algorithm will then iteratively fill in the expected missing values and change the model by removing nodes so as to increase the probability.

3.1 The E-Step - A Sketch Given a graph $G(V, E)$, where V is the set of nodes in the sensor network, and a subset P of nodes that have fixed values, the goal of the E-step is to fill in the values for the nodes in $V - P$.

Calculating the expected values for the missing value nodes is equivalent to minimizing the Hamiltonian $H(G)$ shown in Equation (2.1). Minimizing $H(G)$ is achieved by filling in the missing values so as to minimize the number of conflicts (differences) between neighbors. Since missing value nodes can be neighbors of other missing value nodes, filling in the missing values is not straightforward. Fortunately, this computation is tractable using the following approach. (For simplicity, we leave out several technical details in the following discussion. These details are provided in Section 4.3.)

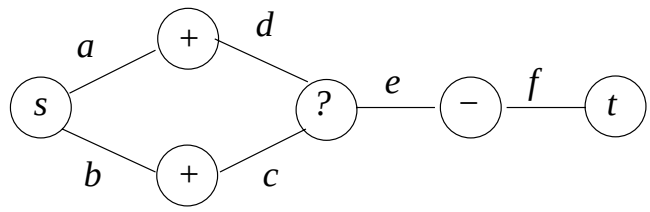


Figure 1: A Simple Illustration Why the MinCut of G_1 Minimizes the Hamiltonian

To the graph G, we add two nodes (s and t), where s has the value $+1$ and t the value -1. All of the nodes in P whose value are $+1$ are connected to s and those whose value are -1 are connected to t. This new graph is called G_1. All edges have unit weights, except that the edges involving either s or t have their weight as ∞. Then, a minimum weight edge cutset of G_1 is the minimum number of edges whose removal will create two node-disjoint subgraphs: a positive subgraph where each node has the value $+1$ or '?' and a negative subgraph where each node has the value -1 or '?'. Determining the minimum weight edge cutset of a graph can be done in polynomial time (see for example, [18]). The subgraph that a missing value node is part of determines the value to be assigned to the node. Figure 1 shows the intuition behind why a minimum cut of the graph is needed. Clearly, in that figure, the missing value should be filled in as '+'. Removing edges marked c and d creates two appropriate subgraphs but it produces the wrong missing value; removing just the edge marked e corresponds to a minimum cut and hence produces the correct result. The E-step can be approximated and easily distributed onto the network by allowing a node with a missing value to set its value to the most commonly occurring value among its neighbors. This approximation will return exactly the same solution as the one based on minimum cut, unless two missing value nodes are neighbors of each other. When there are edges between two nodes with missing values, the two approaches *may* return different results.

Our algorithm shown in Figure 2 formalizes the above discussion. (Some details in the figure rely on definitions presented in Section 4.3.)

3.2 The M-Step - A Sketch Here, we are given a graph $G(V, E)$ with all node values instantiated. (Recall

Input: An undirected graph $G(V, E)$; a subset P of nodes such that each node in P has been assigned a label from $\{+1, -1\}$. (These labels cannot be changed.)

Output: A label from $\{+1, -1\}$ for each node of $V - P$ such that $H(G)$ is minimized.

Algorithm:

1. Construct the auxiliary graph $G_1(V_1, E_1)$ from G.
2. Find an s-t edge cutset C^* of minimum weight in G_1.
3. Construct graph $G_2(V_2, E_2)$, where $V_2 = V_1$ and $E_2 = E_1 - C^*$, from G_1 by deleting the edges in C^*.
4. **for** each node $v \in V - P$ **do**

   if there is a path from s to v in G_2

   then Assign the label $+1$ to v

   else Assign the label -1 to v.

Figure 2: Algorithm for the E-step

that all missing values were filled in by the E step.) We must now compute the most probable model (G^*) for this data. Since our model space is the set of all possible graphs with *upto* $|V| = n$ nodes, we can change the graph G by deleting edges or nodes. In this paper, we have chosen to remove nodes as this approach is more amenable to our aim of reducing power consumption. The nodes removed from G to obtain G^* are put to sleep to conserve power. These nodes may be subsequently awakened for other applications.

In Section 5, we show that the problem of determining which nodes to remove so as to minimize the Hamiltonian is **NP**-complete. Therefore, obtaining an exact solution to this problem (i.e. finding G^* that maximizes Equation (2.2)) is computationally intractable. However, we can easily find a graph structure G' that is more probable than G using the algorithm shown in Figure 3. (The definition of the g function used in Step 2(a) of the algorithm is given in Section 5.) If the variable k used in that figure is chosen to be 1, then the computation can be distributed onto the network, since each node needs to check only its immediate neighbors to determine whether it should go to sleep. Our M-step is an example of *generalized* EM [15], since the calculation yields a *more* probable model, but not necessarily the *most* probable model. Nodes which are put to sleep can be considered as outliers since removing them from

Input: An undirected graph $G(V, E)$ where each node in V has been assigned a label from $\{+1, -1\}$; a subset $P \subseteq V$ of nodes such that there is a path in G from from each node in $V - P$ to some node in P.

Output: Graph G' obtained from G by deleting a subset P' of P. (In G' also, there is a path from from each node in $V - P$ to some node in $P - P'$.) The procedure tries to obtain a graph G' whose energy is as low as possible.

Note: The g function used in Step 2(a) below is defined in Section 5.

Procedure:

1. Choose a positive integer k. (The running time increases with k.)
2. **loop**
   (a) By trying all subsets of P up to size k, find a subset P' such that $g(P') > 0$ and deleting P' does not violate the path property mentioned above. If there is no such subset, go to Step 3.
   (b) Delete the nodes in P' from P. (The graph also needs to be modified accordingly.)
3. Output the resulting graph G'.

Figure 3: Heuristic Procedure for the M-step

the network increases the probability of the entire network/graph.

4 Derivation of the E-Step

4.1 Viewing E-Step as Energy Minimization In the E-step, we need to calculate the expected values of the nodes with missing values. Using a non-parametric model, this is simply determined by the neighbors of the node. Formally:

$$\mathrm{P}(v_i = j) = \sum_{x \in N(v_i)} \frac{1 - \delta(j, x)}{|N(v_i)|} \tag{4.3}$$

where $N(v_i)$ denotes the neighbors of node v_i and δ is the Kronecker delta function ($\delta(p, q) = 1$ if $p = q$ and 0 otherwise).

A common simplification [14] of the E step is to use a "winner take all" approach. This leads to the following E step.

$$\begin{aligned} v_i &= 1 \text{ if } argmax_j P(v_i = j) = 1 \\ &= 0 \text{ otherwise} \end{aligned}$$

Filling in the missing values with their most probable values is equivalent to calculating the *ground state* of the spin-glass model. The ground state involves setting the missing values in such a way that the probability of the configuration (given by Equation (2.2)) is maximized. This, in turn, is equivalent to minimizing the energy value given by Equation (2.1). We now explain how this minimization problem can be solved in polynomial time.

4.2 Graph Theoretic Preliminaries The model space considered in this paper consists of undirected graphs that are simple in the sense that they have no multiedges or self-loops. Let $G(V, E)$ be an undirected graph. When each node $v_i \in V$ is assigned a value $\ell_i \in \{+1, -1\}$, the Hamiltonian function (or the **energy function**) $H(G)$ (Equation (2.2)) can be rewritten using the following definitions.

(a) Each edge $\{v_i, v_j\}$ such that $\ell_i \neq \ell_j$ is called a **conflict edge**.

(b) Each edge $\{v_i, v_j\}$ such that $\ell_i = \ell_j$ is called an **agreement edge**.

LEMMA 4.1. *Let N_c and N_a denote respectively the number of conflict and agreement edges in G. Then, $H(G) = N_c - N_a$. Alternatively, $H(G) = 2\,N_c - |E|$.*

Proof: From the expression for $H(G)$ (Equation (2.1)), it is easy to see that each conflict edge contributes $+1$ to $H(G)$ and that each agreement edge contributes -1 to $H(G)$. Therefore, $H(G) = N_c - N_a$. Since each edge is either a conflict edge or an agreement edge, we have $|E| = N_c + N_a$. Therefore, $H(G)$ is also equal to $2\,N_c - |E|$. ■

The following is an easy observation which will be used later.

OBSERVATION 4.1. *Suppose $G(V, E)$ is an undirected graph where each node is labeled $+1$ or -1. If there is a path in G between a node labeled $+1$ and a node labeled -1, then the path has at least one conflict edge.* ■

4.3 An Efficient Algorithm for the E-Step The E-step of the EM algorithm for filling in missing sensor node values solves the following combinatorial problem.

Minimum Energy Label Assignment (MELA)

Instance: An undirected graph $G(V, E)$ and a subset $P \subseteq V$ of "preassigned" nodes; that is, each node in P has been assigned a label from $\{+1, -1\}$.

Requirement: Assign a label from $\{+1, -1\}$ to each node in $V - P$ such that $H(G)$ is minimized.

It is assumed that in G, there is a path from each node in $V - P$ (i.e., each node with a missing value) to a node in P. This assumption enables the E-step to assign values to missing nodes in an unambiguous fashion.

Our algorithm for the MELA problem relies on a transformation to the problem of finding a **minimum weight $s - t$ edge cut** in an undirected graph. The definition of such an edge cut is given below.

DEFINITION 4.1. *Let $G(V, E)$ be an undirected graph with a nonnegative weight $w(e)$ for each edge $e \in E$. Let s and t be two distinct vertices in V. An **s-t edge cutset** for G is a subset $E' \subseteq E$ such that in the graph $G'(V, E - E')$, there is no path between s and t. A **minimum weight s-t edge cutset** for G is an edge cutset whose total weight is minimum.*

The following well known result shows that minimum weight edge cutsets can be found efficiently (see for example [18]).

THEOREM 4.1. *Given an undirected graph $G(V, E)$ with a nonnegative weight $w(e)$ for each edge $e \in E$ and two distinct vertices s and t in V, a minimum weight s-t edge cutset for G can be computed in $O(|E| + |V| \log |V|)$ time.* ■

Recall that in the MELA problem, the nodes in the set $P \subseteq V$ have preassigned labels which cannot be changed. Throughout this section, we will use E^P denote the set of edges where each edge has both of its endpoints in P. Let N_c^P and N_a^P denote the number of conflict and agreement edges in E^P. (Thus, $|E^P| = N_c^P + N_a^P$.) The contribution H^P of the edges in E^P to the Hamiltonian of the graph G is therefore given by $H^P = N_c^P - N_a^P$. Note that no matter how labels are assigned to the nodes in $V - P$, the edges in E^P will always contribute H^P to the value of $H(G)$.

We now discuss how the MELA problem can be solved efficiently. Let $G(V, E)$ and $P \subseteq V$ denote the given instance of the MELA problem. Consider the auxiliary edge weighted graph $G_1(V_1, E_1)$ constructed from G as follows.

(a) $V_1 = V \cup \{s, t\}$, where s and t are two new nodes (i.e., $s \notin V$ and $t \notin V$).

(b) $E_1 = (E - E^P) \cup E_s \cup E_t$, where $E_s = \{\{s, v_i\} : v_i \in P \text{ and } \ell_i = +1\}$, and $E_t = \{\{t, v_i\} : v_i \in P \text{ and } \ell_i = -1\}$.

(c) For each edge $e \in E_1$, the weight of e, denoted by $w(e)$ is chosen as follows: if $e \in E$, then $w(e) = 1$; otherwise, $w(e) = \infty$.

We note that the auxiliary graph G_1 has a trivial s-t edge cutset of weight $|E - E^P|$. Thus, no minimum weight s-t edge cutset of G_1 can use any of the edges incident on the nodes s and t. In other words, any minimum weight s-t edge cutset of G_1 is a subset of $E - E^P$. The following lemma shows the role played by auxiliary graph in solving the MELA problem.

LEMMA 4.2. *Let $G(V, E)$ and $P \subseteq V$ constitute a given instance of the* MELA *problem. Let $H^*(G)$ denote the minimum value of the Hamiltonian function over all assignments of labels to the nodes in $V - P$. Let $G_1(V_1, E_1)$ denote the auxiliary graph of G constructed as discussed above and let W_1^* denote the minimum weight of an s-t edge cutset in G_1. Then, $H^*(G) = H^P + 2\,W_1^* - |E - E^P|$, where H^P is the contribution due to the edges in E^P.*

Proof: We prove that result in two parts.

Part 1: Here, we prove that $H^*(G) \geq H^P + 2\,W_1^* - |E - E^P|$. Consider an assignment of labels from $\{+1, -1\}$ to the nodes in $V - P$ such that the value of $H(G)$ is equal to $H^*(G)$. Let C denote the set of all the conflict edges from $E - E^P$ in the resulting assignment. As mentioned earlier, the edges in E^P contribute H^P to $H(G)$, regardless of the label assignment to the nodes in $V - P$. From Lemma 4.1, the contribution to the Hamiltonian due to the edges in $E - E^P$ is $2|C| - |E - E^P|$. Therefore, $H^*(G) = H^P + 2|C| - |E - E^P|$. Now, we have the following claim.

Claim: C is an s-t edge cutset for G_1.

Proof of Claim: Suppose C is not an s-t edge cutset for G_1. Then, there is a path from s to t in the graph $G_2(V_1, E_1 - C)$. In this path, let x be the node that is adjacent to s and let y be the node that is adjacent to t. Thus, the label of x is $+1$ and that of y is -1. Hence, by Observation 4.1, there is a conflict edge in this path. By our construction of graph G_1, this conflict edge is from the edge set $E - E^P$. This contradicts the assumption that C contains all the conflict edges from $E - E^P$, and the claim follows.

In view of the claim, G_1 has an s-t edge cutset of weight at most $|C|$. Since W_1^* is the minimum weight of an s-t edge cutset of G_1, we have $|C| \geq W_1^*$. Therefore, $H^*(G) = H^P + 2\,|C| - |E - E^P| \geq H^P + 2\,W_1^* - |E - E^P|$. This completes the proof of Part 1.

Part 2: Here, we prove that $H^*(G) \leq H^P + 2\,W_1^* - |E - E^P|$. This is done by finding an assignment of labels to the nodes in $V - P$ such that the value of the Hamiltonian for the resulting assignment is at most $H^P + 2\,W_1^* - |E - E^P|$.

Consider algorithm in Figure 2. Using the assumption that there is a path in G from each node in $V - P$ to a node in P, it is easy to see that Step 4 of the algorithm assigns a label from $\{+1, -1\}$ to each node of $V - P$. Further, the assignment ensures that the only conflict edges from $E - E^P$ in the resulting assignment are those in C^*. Therefore, by Lemma 4.1, the value of the Hamiltonian function $H(G)$ for this assignment of labels to the nodes in $V - P$ is given by $H(G) = H^P + 2\,|C^*| - |E - E^P|$. Since C^* is a minimum weight s-t edge cutset, we have $|C^*| = W_1^*$. Therefore, $H(G) = H^P + 2\,W_1^* - |E - E^P|$. Since there is an assignment of labels to the nodes in $V - P$ such that the Hamiltonian function of G has a value of $H^P + 2\,W_1^* - |E - E^P|$, it follows that $H^*(G) \leq 2\,W_1^* - |E - E^P|$. This completes the proof of Part 2 as well as that of the lemma. ■

A direct consequence of the above lemma is that the algorithm in Figure 2 computes an optimal solution to the MELA problem. The running time of the algorithm is dominated by Step 2, where a minimum weight s-t edge cutset of G_1 is constructed. As mentioned in Theorem 4.1, this step can be carried out in $O(|E| + |V| \log |V|)$ time. The following theorem summarizes the above discussion.

THEOREM 4.2. *The E-step of the EM algorithm for filling in the missing sensor values can be solved in $O(|E| + |V| \log |V|)$ time, where $|V|$ is the number of nodes and $|E|$ is the number of edges in the given sensor network.* ■

5 Derivation of the M-Step

The M-step requires finding the most probable model given the observed and filled in missing values (from the E-step). We model the M-step by a graph problem where the goal is to delete a set of nodes so that the energy of the resulting graph is as small as possible. Since the goal of the EM algorithm is to fill in the missing values, only nodes that had preassigned labels in the E-step are candidates for deletion. (In other words, nodes whose values are to be filled in cannot be deleted.) When a node is deleted from a graph, all the edges incident on that node are also deleted. Recall that in the graph used in the E-step, there must be a path from each node with a missing value to a node with a value $+1$ or -1. Therefore, the deletion process used in the M-step must ensure that this path property continues to hold in the resulting graph. A precise formulation of the problem is as follows.

Minimum Energy Node Deletion (MEND)

Instance: An undirected graph $G(V, E)$ where each node in V has been assigned a label from $\{+1, -1\}$; a subset $P \subseteq V$ of nodes with preassigned values. (Note: The E-step assigned values to the nodes in $V - P$.)

Requirement: Find a subset $P' \subseteq P$ so that in the graph G' obtained by deleting from G the nodes in P', each node in $V - P$ has a path to some node in $P - P'$ and $H(G')$ is a minimum over all such subsets.

As stated above, MEND is an optimization problem. A decision version of the problem can be obtained in an obvious manner by introducing a parameter B and changing the requirement to the following question: Is a set of nodes P' whose deletion produces a graph G' such that G' has the path property mentioned above and the energy of G' is at most B? For convenience, we will use MEND to denote both the decision and optimization versions of the problem. (The usage will be clear from the context.)

We now show that the MEND problem is computationally intractable. Our proof uses a reduction from the following problem which is known to be **NP**-complete [12].

Exact Cover by 3-Sets (X3C)

Instance: A set $X = \{x_1, x_2, \ldots, x_n\}$, where $n = 3q$ for some positive integer q; a collection $Y = \{Y_1, Y_2, \ldots, Y_m\}$ of subsets of X, where $|Y_j| = 3$, $1 \leq j \leq m$.

Question: Is there a subcollection $Y' = \{Y_{j_1}, Y_{j_2}, \ldots, Y_{j_q}\}$ consisting of q sets such that the union of the sets in Y' is equal to X?

THEOREM 5.1. *The* MEND *problem is* ***NP****-complete.*

Proof: It is easy to see that MEND is in **NP**. To prove **NP**-hardness, we use a reduction from the X3C problem defined above. Given an instance I of the X3C problem, we create an instance I' of the MEND problem as follows. For each element $x_i \in X$, we create a node v_i, $1 \leq i \leq n$. Similarly, for each subset $Y_j \in Y$, we create a node w_j, $1 \leq j \leq m$. Let $V_1 = \{v_1, v_2, \ldots, v_n\}$ and $V_2 = \{w_1, w_2, \ldots, w_n\}$. The node set V for the graph $G(V, E)$ is given by $V = V_1 \cup V_2$. The edge set E is constructed as follows. If $Y_j = \{x_{j_1}, x_{j_2}, x_{j_3}\}$, then we add the edges $\{w_j, v_{j_1}\}$, $\{w_j, v_{j_2}\}$ and $\{w_j, v_{j_3}\}$ to the graph. This completes the construction of the graph G. Each node w_j is assigned the label $+1$ $(1 \leq j \leq m)$ and each node v_i is assigned the label -1 $(1 \leq i \leq n)$. The set P from which nodes can be deleted is V_2. Note that each edge in the graph is a conflict edge. Further, the graph has exactly $3m$ edges. Therefore, the initial energy value of G is $3m$. The energy bound B for the graph after node deletion is set to n. This completes the construction of the MEND problem instance I'. It is clear that the construction of I' can be carried out in polynomial time. We now argue that there is a solution to the MEND instance I' if and only if there is a solution to the X3C instance I.

Suppose there is a solution to the X3C instance I given by $Y' = \{Y_{j_1}, Y_{j_2}, \ldots, Y_{j_q}\}$. A solution to the MEND instance is obtained by deleting from G all the nodes from V_2 *except* w_{j_1}, w_{j_2}, ..., w_{j_q}. Let G' denote the resulting graph. Note that each set in Y' has exactly three elements and Y' is a solution to the X3C instance I. Therefore, in G', each node from V_2 is of degree three and each node from V_1 is of degree one. Hence, G' has the path property mentioned above. Further, the number of edges in G' is exactly n, and each edge is a conflict edge. Therefore $H(G') = n$ as required. Thus, we have a solution to the MEND instance I'.

Now, suppose there is a solution consisting of the node set P' to the MEND instance I'. Let G' denote the graph after the nodes in P' are deleted from G. We have the following claim.

Claim: G' contains exactly $q = n/3$ nodes from V_2.

Proof of Claim: Let V_2' denote the set of nodes from V_2 in G'. We have two cases to consider.

Case 1: Suppose $|V_2'| < q$. Note that each node in V_2' is of degree three. Thus, the total number of edges from the nodes in V_2' is at most $3(q-1) = 3q - 3 < n$. These $3q - 3$ edges are incident on the n nodes in V_1. Thus, at least one node in V_1 has degree zero. Such a node does not have a path to a node in $P - P'$. This is a contradiction since the solution to I' must satisfy the path property.

Case 2: Suppose $|V_2'| > q$. Again, each node in V_2' is of degree three. Thus, the total number of edges from the nodes in V_2' is at least $3(q+1) = 3q + 3 > n$. Each of these is a conflict edge and there are no agreement edges in G'. Thus, the energy of G' is greater than n. This is a contradiction since the solution to I' has an energy value of at most n. The claim follows.

From the above claim, it can be seen that each node from V_2 in G' has a degree of three and that each node in V_1 has a degree of one. It follows that the sets corresponding to the nodes from V_2 in G' form a solution to the X3C instance I. This completes the proof of Theorem 5.1. ■

In view of Theorem 5.1, it is unlikely that the M-step of the EM algorithm can obtain a global minimum energy configuration in polynomial time. So, it is reasonable to look for heuristic methods that reduce the energy iteratively and reach a local minimum. We discuss one such method below.

Suppose we are given an undirected graph $G(V, E)$ with a $+1$ or -1 label for each node and a subset $P \subseteq V$ of candidate nodes that can be deleted. Consider any nonempty subset P^1 of P. Of the edges incident on the nodes in P^1, let N_c^1 and N_a^1 denote respectively the number of conflict edges and agreement edges. Define the **gain** of P^1, denoted by $g(P^1)$, to be the value

$N_c^1 - N_a^1$. Let us call P^1 a **candidate deletion set** if $g(P^1)$ is *greater than* zero and the graph G' obtained by deleting P^1 from G continues to have the path property mentioned above. The reason why P^1 is useful is that the graph G' obtained by the deleting P^1 has a smaller energy value than G. (In fact, $H(G') = H(G) - g(P^1)$.) It will be prohibitively expensive in terms of running time to find a candidate deletion set by trying all possible subsets of P. To make the process efficient, we *fix* an integer k and try only subsets of size at most k. Thus, in each iteration, this method will examine $O(|P|^k)$ subsets. This is reasonable for small values of k. As mentioned earlier, for $k = 1$, the computation can be distributed over the sensor network. When a candidate deletion set is found, the deletion of the corresponding nodes is guaranteed to decrease the energy. When no candidate deletion set of size at most k is available, the procedure stops with a local minimum solution. An outline of the resulting heuristic algorithm is shown in Figure 3.

6 Empirical Results

We present preliminary results for our work. Using the TOSSIM sensor network simulator and MATLAB, we created an artifical sensor network on a uniformly spaced 50×50 grid, with a sensor node at each grid point. Each node has its eight immediate neighbors as its initial neighborhood. It is helpful to remember the our M step removes nodes from the network to increase the probability of the entire network. These removed nodes can be considered as anomalies/outliers.

For our simple *BOX* example (see Figure 4), we randomly removed 10% of all values and then applied our distributed non-parametric EM algorithm to restore them. Examples of networks with missing and then restored values are shown in Figures 5 and 6. Note that a few nodes around the boundary of the simple pattern are turned off. We repeated this experiment ten times. As expected, we found that on average only 0.4% of all missing values were restored to their **incorrect** values for this simple problem.

For our *CIRCLES* example (see Figure 7), we randomly removed 10% of all values and then applied our algorithms to restore them. Examples before and after the restoration data sets are shown in Figures 8 and 9. We found that on average only 1.0% of all missing values were **not** restored to their correct values. We also found that many of the nodes along the boundaries of adjacent circles were turned off as they can be considered anomalous.

As a test of our algorithm's ability to turn off nodes that are outliers, we constructed a *RANDOM* example (see Figure 10) where each node's value is generated randomly. We then randomly removed 10% of all values and applied our algorithms to restore them. Examples before and after restoration data sets are shown in Figures 11 and 12. We found that on average, 2.1% of all missing values were restored to their incorrect values. More importantly, our algorithm turned off the vast majority of nodes indicating that most of the node values were anomalies as would be expected for random data.

7 Conclusion and Future Work

We have presented preliminary results from a line of research for mining using the resource constrained Berkeley mote sensor network platform. We believe that this is an important area as Berkeley motes are an inexpensive and popular sensor network platform that is commercially available in kit form. The basic sensing boards can record temperature, light and noise. Often this information is converted to binary data according to a user settable threshold. The information generated from these motes regularly contains missing or absurd values due to transmission errors, low battery levels, sensor reading errors and node failures. However, many mining algorithms do not easily handle missing data.

In this paper, we examined the use of EM algorithm to fill in the missing values as is the standard practice in statistics. However, parametric EM would quickly consume the battery life of the motes as it requires repeated transmission of the expected values to a central base-station for aggregation. Instead, we propose a novel *non-parametric* EM algorithm that is motivated by the Lattice spin-glass literature. The E-step in our approach leads to a problem that can be solved efficiently. Even though the problem associated with the M-step is is computationally intractable, we proposed a heuristic that finds a more probable (but not necessarily the most probable) model. Therefore, our approach is an example of *generalized* EM [15].

A significant benefit of using non-parametric models is that the E and M steps can be distributed onto the sensor network. This would not be possible for parametric EM since the motes lack floating point hardware and the memory-space needed to execute the corresponding algorithms.

Our distributed non-parametric EM algorithm, functionally, fills in the values for nodes that do not report a value or report an absurd value. In addition, since our M step removes nodes from the network/graph to increase the overall probability, we are effectively removing outliers. Future work will investigate more complex likelihood functions that incorporate the mining task as well data pre-processing. For example, clustering is easily facilitated by the introduction of a latent vari-

able (Q) which is the cluster ID, for each node. The complete data likelihood to maximizes would then be $P(Q, X, Y|\theta)$.

Our preliminary experimental results show that for synthetic sensor network data, the algorithm is capable of restoring the vast majority of missing values correctly. However, additional empirical work must be done to determine and overcome the pragmatic problems in the approach.

References

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci, "Wireless Sensor Networks: A Survey", *Computer Networks*, Vol. 38, 2002, pp. 393–422.

[2] R. Caruna, "A Non-Parametric EM-Style Algorithm for Imputing Missing Values", AI-Stats 2001.

[3] *Communications of the ACM*, Special Issue on Wireless Sensor Networks, June 2004.

[4] R. O. Duda, P. E. Hart and D. G. Stork, *Pattern Classification* (2nd edition), Wiley Interscience, 2000.

[5] I. Davidson and G. Paul, "Locating Secret Messages in Images", 10th SIG KDD Conference 2004.

[6] P. Domingos and M. Pazzani, "Beyond independence: Conditions for the optimality of the simple Bayesian Classifier", ICML, 1996.

[7] H. Edelistein, Two Crows Data Mining Report, Two Crows Corporation 1999.

[8] E. Elnahrawy and B. Nath, "Poster Abstract: Online Data Cleaning in Wireless Sensor Networks", *Proc. SenSys'03*, Los Angeles.

[9] E. Elnahrawy and B. Nath, "Cleaning and Querying Noisy Sensors", *Proc. WSNA'03*, San Diego, CA.

[10] E. Elnahrawy and B. Nath, "Statistical Approaches to Cleaning Sensor Data", book chapter in *Distributed Sensor Networks*, Edited by S. S. Iyengar and R. R. Brooks, CRC Press, 2003.

[11] E. Elnahrawy and B. Nath, "Context-Aware Sensors", *Proc. European Workshop on Wireless Sensor Networks*, 2003, pp. 77–93.

[12] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman and Co., San Francisco.

[13] *IEEE Computer*, Special Issue on Sensor Networks, Aug. 2004.

[14] M. Kearns, Y. Mansour, and A. Y. Ng, . "An information-theoretic analysis of hard and soft assignment methods for clustering", *Proc. UAI 1997*.

[15] T. Mitchell, *Machine Learning*, McGraw-Hill, 1997.

[16] C. S. Raghavendra, K. M. Sivalingam and T. Znati (Editors), *Wireless Sensor Networks*, Kluwer Academic Publishers, Norwell, MA, 2004.

[17] Schumitzky, A., NonParametric EM Algorithms for Estimating Prior Distributions, TR 90-2, Department of Mathematics, University of Southern California.

[18] M. Stoer, F. Wagner, "A Simple Min-Cut Algorithm", *J. ACM*, Vol. 44, No. 4, July 1997, pp. 585–591.

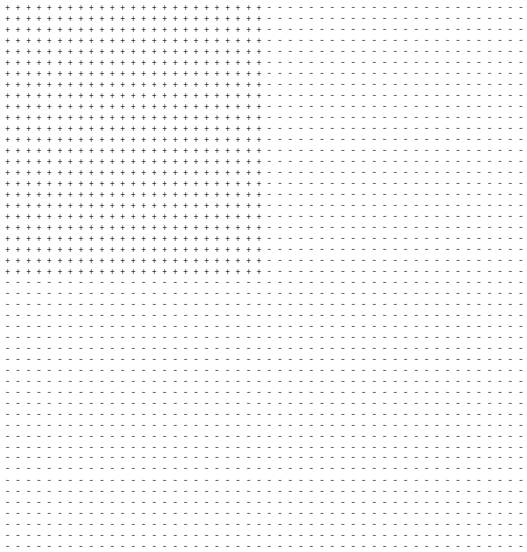


Figure 4: Original Box Data for a 50×50 uniformally spaced sensor network.

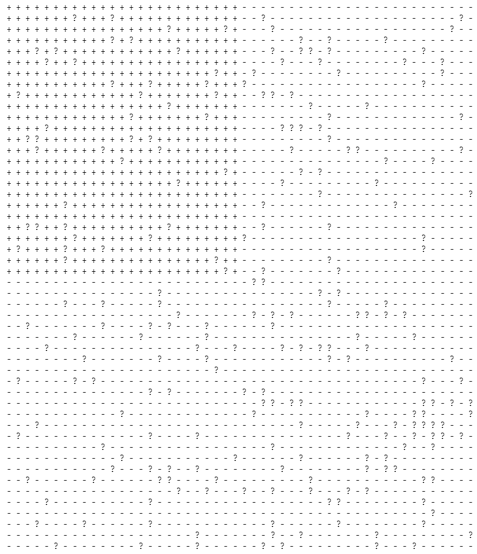


Figure 5: Box Data with Missing Values for a 50×50 uniformally spaced sensor network. The symbol '?' indicates a missing value.

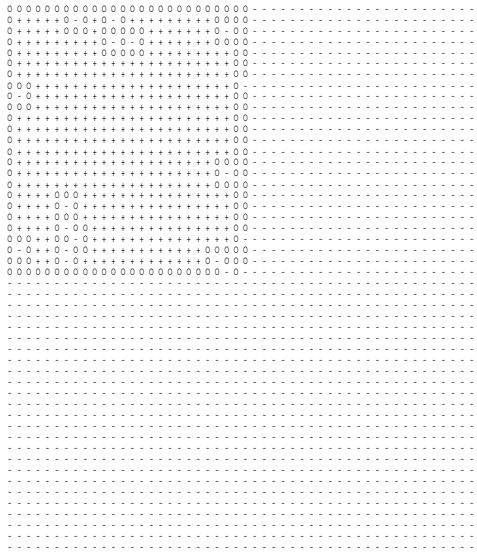


Figure 6: Recovered Box Data for a 50×50 uniformally spaced sensor network. The symbol 'O' indicates a node to be turned off.

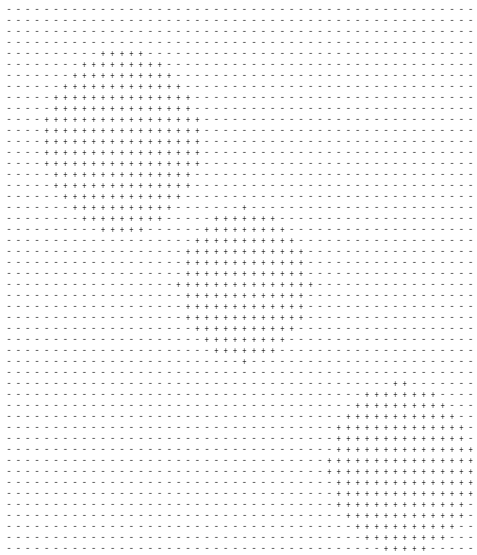


Figure 7: Original Multiple Circles Data for a 50×50 uniformally spaced sensor network.

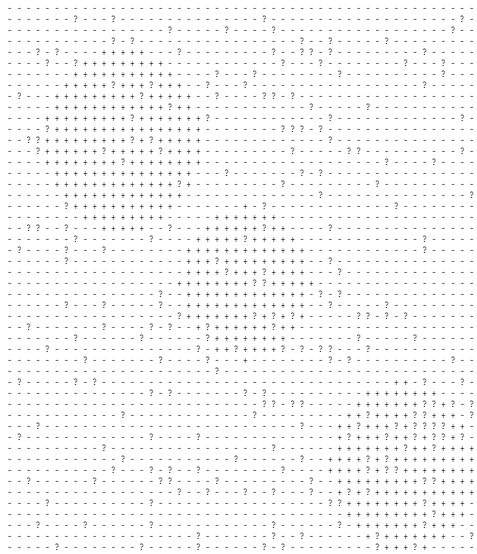


Figure 8: Multiple Circles Data with Missing Values for a 50×50 uniformally spaced sensor network. The symbol '?' indicates a missing value.

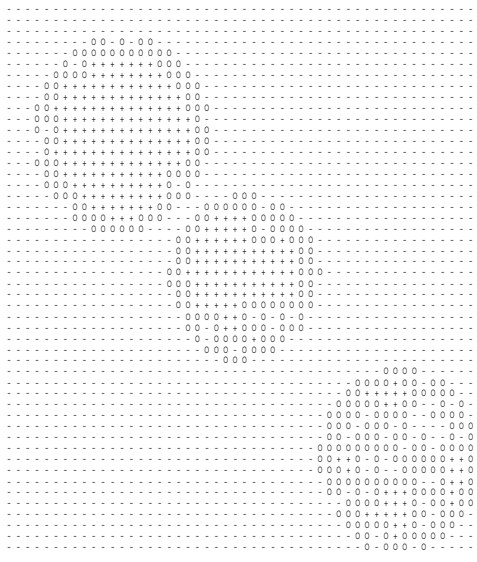


Figure 9: Recovered Multiple Circles Data for a 50×50 uniformally spaced sensor network. The symbol 'O' indicates a node to be turned off.

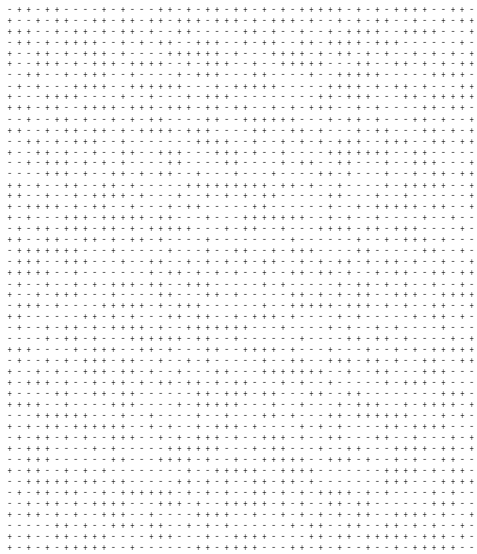


Figure 10: Original Random Data for a 50×50 uniformally spaced sensor network.

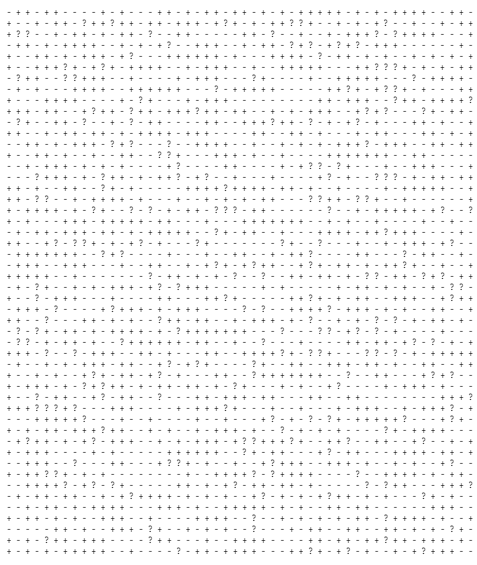


Figure 11: Random Data with Missing values for a 50×50 uniformally spaced sensor network. The symbol '?' indicates a missing value.

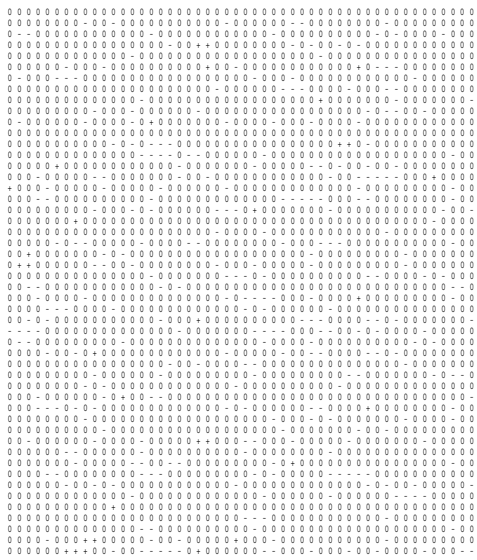


Figure 12: Recovered Random Data for a 50×50 uniformally spaced sensor network. The symbol 'O' indicates a node to be turned off.

A Distributed Approach for Prediction in Sensor Networks

Sabine M. McConnell and David B. Skillicorn
School of Computing
Queen's University
{mcconnell,skill}@cs.queensu.ca

Abstract

Sensor networks in which the sensors are capable of local computation create the possibility of training and using predictors in a distributed way. We have previously shown that global predictors based on voting the predictions of local predictors each trained on one (or a few) attributes can be as accurate as a centralized predictor. We extend this approach to sensor networks. We also show that, when the target function drifts over time, sensors are able to make local decisions about when they need to relearn to capture the changing class boundaries.

Keywords: sensor networks, prediction accuracy, classification task, distributed data mining, anomaly detection, local models

1 Introduction

As sensors increasingly become active devices, with their own processing power, rather than simply passive input devices, new possibilities for implementing distributed algorithms in a network of sensors become possible. One important class of such algorithms are predictors, which use the sensor inputs to predict some output function of interest – one topical example is the use of ocean overpressure sensors and seismic detectors to predict a tsunami.

Sensor networks are of two broad kinds: peer-to-peer or hubbed. In a peer-to-peer (or *ad hoc*) network, each sensor has access to some neighbors and the overall network resembles a random graph. In a hubbed network, the network structure is a tree, where leaves are sensors, the root is some more substantial computational device, and the internal nodes may either be ordinary sensors, resemble the root node, or something in between. We will consider only hubbed sensor networks.

The trivial solution to learning a predictor in such a sensor network is of course simply to transmit the data from all of the sensors to the root, and carry out all of the predictor training and prediction at the central site. This has several disadvantages: moving the raw data to the root increases power consumption, uses bandwidth and introduces latency. In addition, the sensors remain passive devices.

We show that prediction in a hubbed sensor network can be distributed in the following way: each sensor learns a local predictive model for the global target classes, using only its local input data. Only the predicted target class for each reading is then transmitted to the root, which determines the appropriate prediction by voting the target classes of the local predictors. This approach has a number of desired properties: it is no less (and potentially even more) accurate than a centralized predictive model; it requires only the target class prediction to be transmitted from the sensor, rather than the entire raw data; and it allows each sensor to locally determine its behavior and respond to its environment, for example deciding when to relearn its local model in response to changes in the input data.

The contributions of this paper are:

- A framework for building and deploying predictors in sensor networks that pushes most of the work out to the sensors themselves. This builds on our earlier work on distributed data mining. We show that prediction performance is not negatively impacted by using the framework instead of a centralized learning approach.

- We show how the use of local predictive models enables sensors to respond to changes in data by relearning when their local predictive accuracy changes. This creates new possibilities, such as allowing sensors to predict only some target classes, for example those representing anomalies, therefore further reducing the required bandwidth.

Because sensors only transmit model predictions, and not raw data, our approach can also be used in settings where privacy is a concern, for example when sensors belong to different organizations or governments.

2 Related Work

The deployment of data mining techniques in sensor networks is an open research problem. For a general discussion of the characteristics and requirements of sensor networks see the survey by Akyildiz *et al.* [1]. One application is the distributed stream mining system Vedas [9] by Kargupta *et al.*, which monitors data streams obtained in moving vehicles before reporting them to a base. More recently, Bandyopadhyay *et al.* [2] introduced a distributed clustering algorithm based on the k-means clustering algorithm in a peer-to-peer environment.

The current limitation of data mining applications in sensor networks is that existing distributed data mining techniques impose heavy demands on computation and/or communication. In addition to the trivial approach of sending all collected data from the individual sensors to the root, where any standard data-mining technique can then be applied, two techniques for distributed data mining are known: the Collective Data Mining (CDM) framework introduced by Kargupta *et al.* [8], and metalearning [5].

The CDM framework encompasses a variety of data mining techniques, while metalearning is algorithm independent. The CDM framework is built upon the fact that any function, i.e. the function to be learned by the data mining algorithm, can be expressed as a (possible infinite) sum of basis functions and their corresponding coefficients. It is those coefficients that are learned from the local datasets and then combined into a global model. This is computationally expensive, and the sum is only approximated, which puts an upper boundary on the accuracy that can be achieved. In metalearning, outputs from classifiers built at separate sites and from separate datasets are combined by sending them as input to a second level of classifiers, which then produce the final result. Because metalearning requires additional resources, both for the transmission of the models as well as the computation of the models themselves, it is not well suited to applications in sensor networks.

3 Learning Predictors in Sensor Networks

Provided that each of the sensors provides an attribute value (reading) at the same time, the global view of the collected data is a matrix. Each row of the matrix corresponds to a (global) reading and each column to an attribute. Each sensor may collect only a single attribute, so that it is responsible for only one column of the dataset, or it may collect multiple attributes. A target class is associated with each global reading; for *training* data, there is a further column containing these target classes. After deployment, the goal is to predict the target class for each global reading.

In our approach, each sensor builds a local predictor that predicts the target class based only on the attribute(s) available to it locally. At first, it might seem as if such predictors would be too poor to be useful, especially given the amount of research directed towards more heavyweight distributed data-mining techniques as described above. However, we have shown [10] that, even when a model is built from each attribute individually, predictive performance is not necessarily worse than that of a centralized predictor. This follows partly from the same reason that Bayesian classifiers are often useful, even when the independence assumption on which they depend is violated. Domingos and Pazzani [6] attribute the success of the naive Bayes classifier to the fact that even though the class probabilities are estimated wrongly, their ranking is preserved, i.e. the class that should be assigned to with the highest probability is still the highest under the naive Bayes assumption. It should also be noted that many predictive model

building algorithms do not account for attribute correlation well, even when they purport to.

Our model of a sensor network is as follows. The network is a tree, with a powerful computational device at the root, and sensors, with limited power, processing, and memory capabilities at the leaves. Bandwidth in the network is a scarce resource, in part because transmission by the sensors consumes power. In addition, available resources vary across different sensor networks. Sensors gather data synchronously (logically if not physically), and the overall goal is to predict some function of all data inputs.

During the learning phase, each sensor builds a predictor for the global target class based on values of its local input data. The accuracy of each local predictor can be determined using test data for which the correct target class predictions are known. During deployment, each sensor receives data from the outside world, and sends its target class predictions for each of these readings (directly or through other sensors) to the root. The root uses voting to choose a global target class from these individual predictions. Note that the root can tell whether each individual sensor's prediction was 'right' or 'wrong' by whether it agreed with the overall vote, and this can act as a surrogate for prediction accuracy for each local predictive model.

There are several variations in this overall approach, depending on the capabilities of the sensors and the particular predictive technique being used. Some predictive techniques require access to all of the training data at once (for example, decision trees), while others require only one reading at a time (for example, neural networks). In both cases, sensors must have enough memory to store the model itself and perhaps some statistics about it; but in the former case, a sensor must also have enough memory to hold the entire training data for its input attribute(s).

When only one reading is stored at a time, the training regime is given in Algorithm 1. For model building techniques that must see the data multiple times (for example, the epochs required in backpropagation training of neural networks), the data must be transmitted multiple times from the root. Hence small memory at the sensors is bought at the cost of increased communication. The test regime is outlined in Algorithm 2. During deployment, each sensor classifies each new input it collects according to the local model and then sends its prediction to the base. Voting can either be unweighted, so that each sensor's prediction makes the same contribution to the global prediction, or weighted by factors such as the test set accuracy of each sensor's local predictor or the confidence with which local models assign input values to target classes. The number of target classes is typically much smaller than the range of values in the attribute domain, and so much less bandwidth is required to transmit predictions than raw data. It should be noted at this point that even a model that requires the entire training data for its construction can sometimes be modified to learn incrementally. A number of incremental algorithms are known, for example those proposed by Thomas [11] and Utgoff [12]. This implies that adaptations of these techniques could be deployed even in sensor networks with extremely limited resources.

Algorithm 1 Learning with extremely limited resources

for all objects in the training data **do**
  Root sends values for attribute i to sensor i
end for
Sensor i uses the attribute value to build its local model

Algorithm 2 Testing with extremely limited resources

for all objects in the test data **do**
  Root sends values for attribute i to sensor i
  Sensor i uses its local predictor to predict the target class
  Sensor i sends its prediction to the root
  The root votes using the predictions of each sensor to get a global prediction
end for
The root computes the overall prediction accuracy using the target class labels of the test data
The root computes the accuracy of each local predictor and sends this to each sensor

Algorithm 3 Learning with larger resources

for all sensors **do**
  Root sends the columns corresponding to each sensor's attributes and a copy of the target attribute to each sensor
  Each sensor builds a local predictor from this data
end for

Algorithm 4 Testing with larger resources

for all sensors **do**
  Root sends the columns corresponding to each sensor's attributes and a copy of the target attribute to each sensor
  Each sensor uses its local predictor to generate a list of predicted target classes and sends them to the root
end for
Root votes using the target class predictions from each sensor to get a global prediction for each test-set reading

When each sensor has a significant amount of memory available, a less rigid training and testing regime is possible as outlined in Algorithms 3 and 4, respectively. After testing, the root can send each sensor both its per-reading accuracy and its global accuracy. The sensors are deployed exactly as above. The amount of storage required at each sensor has to be sufficient to store the larger of the training and test set data, in addition to the target class in each case. These strategies can be used with any weak learner. The local deployed error rate is then defined as the error rate of the locally built model as compared to the results achieved by the global model. The local deployed error rate is measured against the global error rate, which implies that relative changes rather than absolute values are of interest.

4 Trend detection

In some sensor networks, the target function may change over time, and the predictive model may need to be relearned. This is difficult to handle in a centralized way since a drop in the global predictive accuracy could signal either a change in the target function or a problem with the predictor. Assuming that not all local predictors will fail at the same time due to random data fluctuations, and because the global predictive model depends on many local predictors, a change in a local predictor's accuracy can then be used to trigger *local* relearning to improve that predictor's accuracy in response to changes in input data.

After deployment, the root should send the correct class label for each prediction back to each sensor. A sensor can use this to track its accuracy on observed inputs (which might be expected to match its training accuracy, or at least to remain stable over time) and/or to label each of the recent observed readings with its correct class label. Through the knowledge of class labels, a sensor can then discover that its predictions are starting to be less effective, which might trigger new local behavior (relearning its predictive model), or new global behavior (relearning all of the predictors or, alternatively, removing this sensor from predictions). If weighted voting is used, a sensor whose predictions start to become less accurate is automatically downweighted at the root. The availability of global target classes also means that recent data can be used as training data for relearning local models.

5 Experiments

5.1 Basic distributed data mining The feasibility of the distributed voting approach was demonstrated using the following experiment that simulates distributed execution on an artificial dataset. A more detailed evaluation of the effectiveness of this approach, utilizing a variety of real life datasets containing a large range of numbers of classes, samples and attributes of varying type can be found in [10].

Datasets were generated by drawing from two normal distributions in 10-dimensional space, choosing various different separations for the means in each dimension, and different magnitudes for the variances. The target class is the distribution from which each row was drawn. 500 samples were drawn from each distribution, giving a dataset with 1000 rows and 10 columns.

Each sensor receives a single column of this

dataset, together with the corresponding class labels. This is the worst case scenario, since a sensor might be gathering more than one signal at a time. These data were further separated into training and test sets using the out-of-bag estimator procedure suggested by Breiman [3]: samples are drawn with replacement until 1000 samples have been selected. The remaining rows, typically about one-third of the original dataset, are used as the test set. A confidence measure for each sensor's predictive model is then obtained by using this test set. Such a confidence measure is expected to be as accurate as if the test set were of the same size as the training set, so that confidence intervals will be small for this data. Global prediction accuracies are computed in two ways: simple voting, and voting weighted by the probability with which a sensor assigns a particular reading to a class.

The predictive model used is the J48 decision tree implemented in WEKA (`www.cs.waikato.ac.nz/ml/weka/`), although the approach will work for any weak learner. The achieved accuracies for both voting schemes on this dataset are shown in Table 1. The accuracies achieved by building a single decision tree on the whole dataset (the centralized solution) are included for comparison.

From these results, it can be seen that the overall classification accuracy for both voting approaches is comparable to or better than that of the centralized approach datasets in all cases. In addition, the weighted voting approach outperforms the simpler voting scheme. For an explanation of the observations and a more general experimental evaluation, see [10].

5.2 Effect of trends during deployment We now consider the effect of a target function, and hence a class boundary, that changes with time. In the following experiments, we generated data similar to that described above, but moved the means of the distributions incrementally.

A dataset of a 1000 samples was drawn from two normal distributions whose means were centered at the origin and (1,1,...,1) with variance one. Ten further datasets, each of size 1000, were generated from two normal distributions with the same variance; the means of these two distributions were moved in lockstep in axis-parallel steps according to the schedule shown in Table 2 for a total distance of 20 in each dimension. For example, attribute 1 changed by 10% of its total change at each iteration, while attribute 3 made the total change only during the last iteration. The extension to the previous deployment is that each sensor tracks its local deployed error rate (which is derived from the global prediction, not from the 'true' prediction) When this accuracy changes sufficiently, the sensor relearns its local model, using recent data and the target class labels reported to it by the root. Algorithm 5 outlines the approach.

Algorithm 5 Relearning during deployment

```
Create a local model in each sensor
Classify each new reading according to the local
model
if local accuracy deviates then
  Relearn the local model
end if
Send the resulting classifications to the base
Combine the predictions from the sensors in the
base using a voting scheme
```

The deviation that triggers relearning was taken to be 5 percentage points of prediction accuracy for each attribute. Figures 1 and 2 show how the accuracy changes as the target function changes, how relearning is triggered, and how the prediction accuracy subsequently improves.

Figure 3 shows the effect of the data change and relearning on the overall accuracy achieved at the base after combining the classifications sent from the sensors using both of the voting schemes.

This strategy assumes the change in class boundaries will not require relearning in all attributes simultaneously, for then the global class labels would not be appropriate surrogates for the correct prediction. This assumption is reasonably robust, since a simultaneous relearning would indicate a drastic change in the target function, which is unlikely in most realistic settings.

We make the following observations.

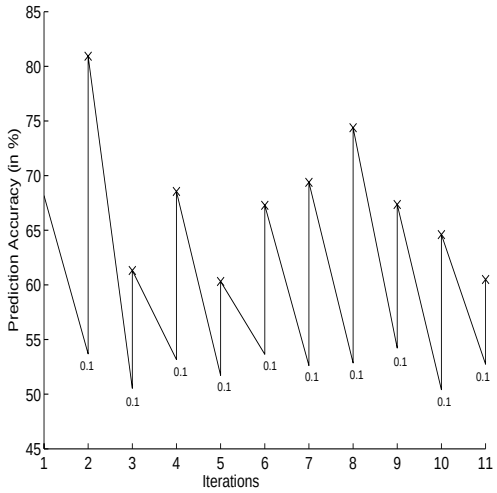
- *Relearning corresponds to changes in target function.* For each change in the target class

Distance between means in each dimension	**Variance**	**Number of dimensions =attributes**	**Simple voting**	**Weighted voting**	**Centralized prediction**
1	0.5	10	98.10 (0.88)	99.28 (0.30)	94.31 (1.43)
1	0.5	5	96.61 (0.65)	96.61 (0.65)	94.96 (1.07)
1	1	10	88.05 (2.43)	89.03 (1.61)	82.75 (2.12)
1	1	5	81.16 (3.08)	81.16 (3.08)	77.70 (2.02)
2	0.5	10	100 (0)	100 (0)	99.11 (0.56)
2	1	10	98.81 (0.48)	99.15 (0.31)	95.10 (1.21)
2	0.5	5	99.97 (0.09)	99.97 (0.09)	99.79 (0.25)
2	1	5	96.98 (0.65)	96.98 (0.65)	94.38 (1.26)

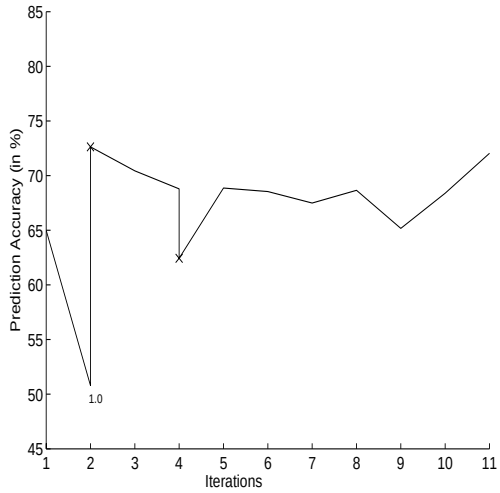
Table 1: Mean global prediction accuracies over 10 trials, using simple and weighted voting, on datasets with different class mean separation and variances. The values in parentheses indicate standard deviations.

Iteration	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Attribute 1	0.0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Attribute 2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Attribute 3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
Attribute 4	0.0	0.5	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0
Attribute 5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.4	0.2
Attribute 6	0.0	0.4	0.0	0.4	0.0	0.0	0.1	0.0	0.1	0.0	0.0
Attribute 7	0.0	0.2	0.0	0.2	0.0	0.2	0.0	0.2	0.0	0.2	0.0
Attribute 8	0.0	0.2	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0
Attribute 9	0.0	0.5	0.2	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.1
Attribute 10	0.0	0.3	0.0	0.0	0.3	0.0	0.0	0.0	0.3	0.0	0.1

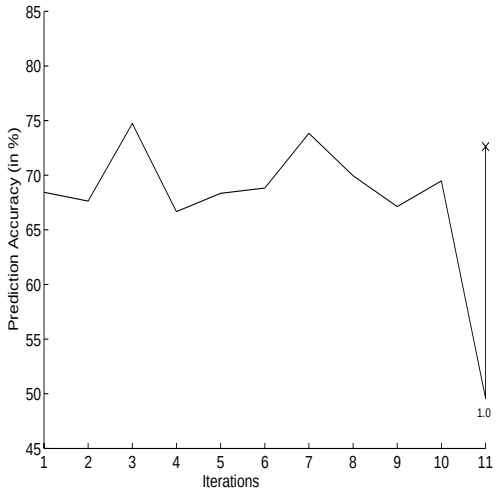
Table 2: Change of dimension means in each iteration (as % of the total change)



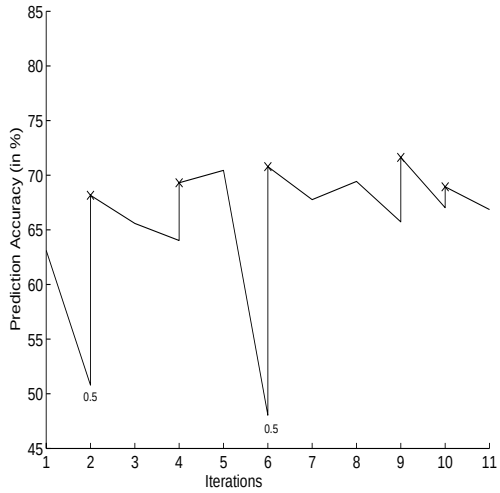
(a) Attribute 1



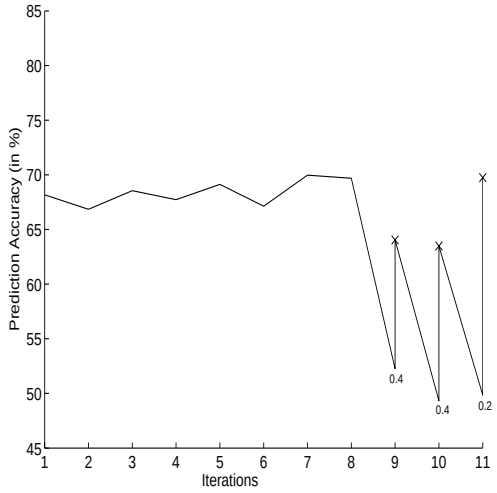
(b) Attribute 2



(c) Attribute 3

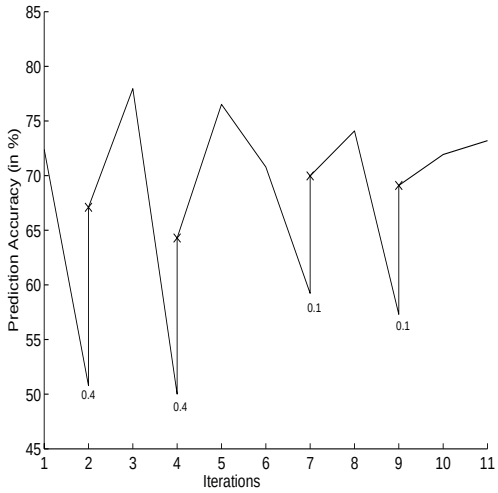


(d) Attribute 4

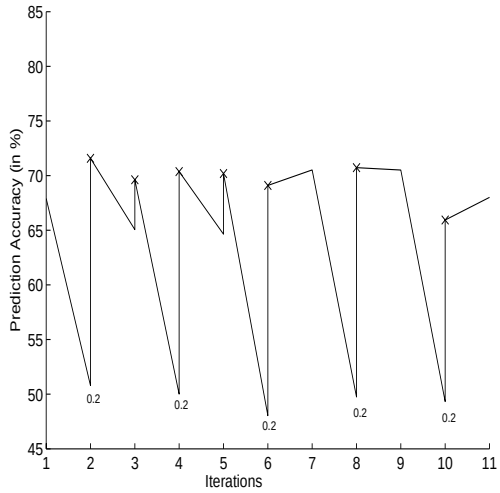


(e) Attribute 5

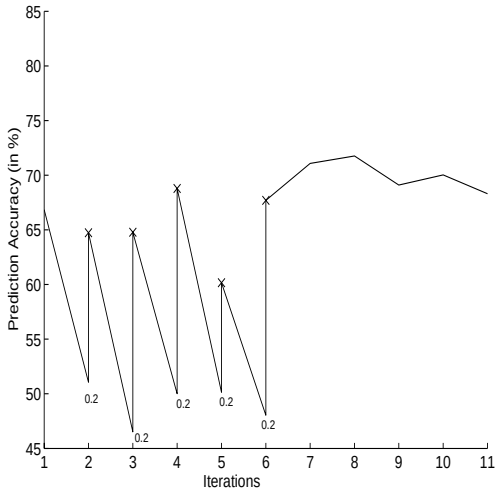
Figure 1: Accuracies for Attributes 1 through 5 over time. Values marked with an x indicate the accuracies after relearning was triggered. Numerical values indicate the percentage change of the target class centers in that dimension (and so for that attribute).



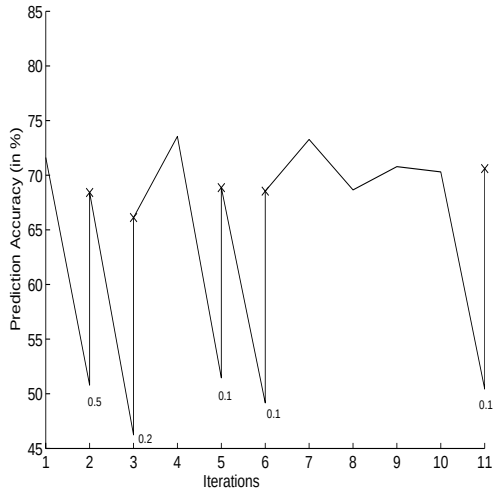
(a) Attribute 6



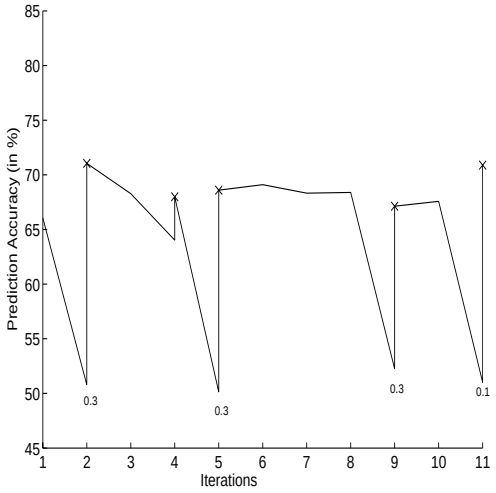
(b) Attribute 7



(c) Attribute 8



(d) Attribute 9



(e) Attribute 10

Figure 2: Accuracies for Attributes 6 through 10 over time. Values marked with an x indicate the accuracies after relearning was triggered. Numerical values indicate the percentage change of the target class centers in that dimension (and so for that attribute).

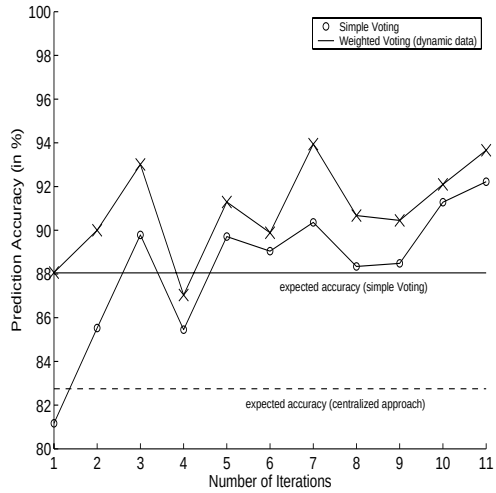


Figure 3: Overall prediction accuracy with moving class boundaries.

centers, and independent of the amount of change, relearning is triggered for the local models. This is true for all attribute values and across all iterations. For example, the model describing attribute 1 is relearned at each iteration after the initial one, corresponding to the change of the distribution mean in that dimension.

- *Relearning occurs occasionally even without changes in the data.* This is seen for example in the plot for attribute 10, where relearning occurred in iteration 4. It was observed that over different runs of the experiment, relearning without corresponding target class changes varied in both frequency and the timing of its occurrence. We therefore suspect that this is in response to occasional fluctuations in the datasets due to the randomness of initial construction and selection of the training and test sets.

- *The overall classification accuracy for both the simple and the weighted voting scheme is as good or better than that of the centralized data mining approach.* We know from the first experiment that the overall accuracies achieved for a dataset with a separation between distribution means of 1 in each dimension and a variance of 1 are 88.05% and 89.03% for the simple and weighted voting scheme, respectively. In addition, the estimated value for the accuracy for a centralized approach is 82.75%. From the results depicted in Figure 3, we see that the overall accuracies achieved through the majority voting schemes with relearning is equivalent or better to those achieved by a centralized data mining approach.

- *The global accuracy at the root is better than the individual accuracies obtained from the local models.* This is due in part to the ensemble effect, which allows the combined accuracy to be superior to each of the predictors contained in the ensemble, provided that the local predictors are both accurate and make different errors on different data [7].

It should be noted here that the transmission of a probability along with the class prediction from the sensors to the base requires additional communication and might not be feasible if resources are limited. However, the accuracy achieved by the simple voting scheme is equivalent to the accuracy achieved by a centralized approach and therefore sufficient.

We have assumed that sensor reading, sensor transmission of local prediction, and voting at the root are synchronous. This is a moderately strong assumption, since it would require a common clock.

In fact, this requirement can be relaxed in several ways, which will be discussed in more detail in a subsequent paper. For example, the root may recalculate the vote whenever a new prediction is reported from a sensor, freeing sensors to report their predictions asynchronously. A major application of sensors networks is as anomaly detectors for complex anomalies that can only be detected by concerted changes in the data at several sensors. This can be modelled as a two-class problem (Safe vs. Alarm). The amount of communication required is greatly reduced if sensors only report predictions for the Alarm class; the root can then predict an Alarm in response to some number of local predictions of Alarm by the sensors.

6 Conclusions

In this paper, we have presented a framework for building and deploying predictors in sensor networks in a distributed way, by building local models at the sensors and transmitting target class predictions rather than raw data to the root. At the root, local predictions are combined using weighted or unweighted voting. This framework is appropriate for the limited resources found in sensor networks, due to power, bandwidth and computational limits. We have also showed how the use of local predictive models enables sensors to respond to changes in targets by relearning local models when their local predictive accuracy drops below a threshold. This enables effective distributed data mining in the presence of moving class boundaries, and also creates new possibilities, for example the use of sensors to detect anomalies, even when the criteria for an anomaly changes over time. Finally, because only model predictions rather than data are transmitted, the framework is also suitable for settings where data confidentiality is a concern.

References

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, *A Survey on Sensor Networks*, IEEE Communications Magazine, August 2002, pp. 102–114.

[2] S. Bandyopadhyay, C. Gianella, U. Maulik, H. Kargupta, K. Liu, and S. Datta, *Clustering Distributed Data Streams in Peer-to-Peer Environments*, 2004, Accepted for publication in the Information Science Journal, in press.

[3] L. Breiman, *Out-of-bag Estimation*, 1996, Technical Report, Statistics Department, University of California, Berkeley, Ca.

[4] L. Breiman, *Random Forests*, Machine Learning, 45(1), 2001, pp. 5–32.

[5] P. Chan and S. Stolfo, *Experiments in Multistrategy Learning by Meta-Learning*, Proceedings of the Second International Conference on Information and Knowledge Management (Washington, DC), 1993, pp. 314-323.

[6] P. Domingos and M. Pazzani, *Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier*. In Proceedings of the Thirteenth International Conference on Machine Learning (ICML), 1996

[7] L. Hansen, and P. Salamon, *Neural Network Ensembles* , IEEE Transactions on Pattern Analysis and Machine Intelligence,(12) 1990, 993–1001.

[8] H. Kargupta, B. Park, D. Hershberger, and E. Johnson, *Collective Data Mining: A New Perspecive Towards Distributed Data Mining*, Advances in Distributed Data Mining, Eds: H. Kargupta and P. Chan, AAAI/MIT Press, 1999.

[9] H. Kargupta, R. Bhargava, K. Liu, M. Powers, P. Blair, S. Bushra, J. Dull, K. Sarkar, M. Klein, M. Vasa, and D. Handy, *VEDAS: A Mobile and Distributed Data Stream Mining System for Real-Time Vehicle Monitoring*, Proceedings of the SIAM International Data Mining Conference, Orlando, 2004.

[10] S. McConnell, and D. Skillicorn, *Building Predictors from Vertically Distributed Data*, Proceedings of the 14th Annual IBM Centers for Advanced Studies Conference, Markham, Canada, 2004, pp.150–162.

[11] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka, *An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases*, Knowledge Discovery and Data Mining, 1997, pp. 263–266.

[12] P.Utgoff, *An Improved Algorithm for Incremental Induction of Decision Trees*, Proceedings of the International Conference on Machine Learning, 1994, pp. 318–325.

Local Hill Climbing in Sensor Networks

Denis Krivitski* Assaf Schuster† Ran Wolff‡

Abstract

Consider a sensor network comprised of thousands of battery operated sensors that collect data and transfer it to a number of gateways sensor, which have a more permanent power supply and which then transmit the data to the system's operators. Assume that although there may be quite a few available gateways (e.g., passer by vehicles, overflying airplanes, ground facilities, etc.) bandwidth and power considerations, or restrictions at the operators' side only permit that a fixed number of these facilitate the communication at every given time. The question then arises which are the k potential gateways which will best support the sensors. Note that this problem should be continuously solved because the availability of the potential gateways, as well as the ability of sensors to communicate with them and the importance of the data generated by each sensor may vary considerably over time.

The problem described above is an instance of the well known facility location problem (FLP) which deals with finding the optimal way to provide a service to a (possibly) very large number of clients. FLP has been studied in many application areas, including the placement of distribution centers in the retail domain, of proxy servers in Internet domain, and of relay stations in the telecommunication domain. Since FLP is NP-hard, a hill-climbing heuristic is regularly used to provide an approximate solution. In this paper we demonstrate that in some cases hill climbing can be solved using a *local* algorithm. Local algorithms are important for sensor networks because they have superb message pruning capabilities and because they perform their entire computation in-network. A sensor taking part of a local algorithm computes an exact result using, in many cases, data it gathers from just its nearest neighborhood. In such cases local algorithms are far better than any centralized algorithm, both in terms of message efficiency and of convergence time.

*Computer Science Dept., Technion – Israel.
†Computer Science Dept., Technion – Israel.
‡CSEE Dept., U. Maryland, Baltimore County.

1 Introduction

Determining the location of facilities which provide system related services is a major issue for any large distributed system. The resource limited scenario of a sensor network makes the problem far more acute. A well placed resource (cache server, relay, high-powered and more accurate sensor, etc.) can tremendously increase the lifespan and the productivity of tens and hundreds of battery operated sensors. In many cases, however, the optimal location of such services depends on dynamic characteristics of the sensors (e.g., their remaining power), of the environment (e.g., level of radio frequency white noise), or of the phenomena they monitor (e.g., frequency of changes). Thus, optimal placement cannot be computed apriori, independently of the system's state.

As an example for a practical facility location problem that may occur in sensor networks consider a network that has two kinds of sensors: Thousands of cheap, low power, motion sensing devices which are distributed from the air, covering the area randomly; and a few dozens of relays with large batteries and large range transceivers, that are placed in strategic points by ground transportation. The purpose of the relays is to collect data from the sensors and to transmit it to a command station whenever it is asked for. However, the question remains how to best utilize the relays which in themselves have but limited resources. It would make sense to shut down a relay if there is but mild activity in its nearby surrounding and report that activity via other relays. Note however, that both the amount of activity, the remaining resources of the relays and of the motion sensors, and the environmental conditions in which they all operate may influence the decision and that these are time varying. The optimal solution, thus, has to be regularly adjusted.

The facility location problem (FLP) has been extensively studied in the last decade. Like many other optimization problems, optimal facility location is NP-Hard [15][13]. Thus, the problem is often subjected to a hill-climbing heuristic [16, 10, 5]. Hill-climbing is a simple and effective heuristic search technique in which it is assumed that a reasonable global optimum can be reasonably approached if on each search step the al-

gorithm chooses the direction that maximizes the immediate gain. The strength of the method is in its simplicity. It has been extensively tried for optimum search in exponential domains in problems such as genetic algorithms, clustering, etc. A rather surprising result by Arya at el. [1] states that, for FLP, hill climbing achieves a constant factor approximation of the globally optimal solution [1].

To our best knowledge FLP has never been studied specifically in a distributed setting. Nevertheless, it is easy to see how distributed formulation of related hill-climbing algorithms such as k-means and k-median clustering [6, 9, 8] can be adopted to solve distributed FLP. We note, however, that all previous work on distributed clustering assume tight cooperation and synchronization between the processors containing the data and a central processor that collects the sufficient statistics needed in each step of the hill-climbing heuristic. Such central control is not practical in wireless networks both from an energy requirements perspective and because it is prune to errors in the occurrence of even single failures. Even more importantly, central control is unscalable in the presence of dynamically changing data because any such change must be reported to the center, for fear it might alter the result.

In contrast, the most important features which qualify an algorithm for sensor networks are the following: the ability to perform in a router-less network (i.e., to be driven by data rather than by address), the ability to calculate the result in-network rather than collect all of the data to a central processor (which would quickly exhaust bandwidth [11]), and the ability to locally prune redundant or duplicate computation. These three feature typify *local* algorithms.

A local algorithm is one in which the complexity of computing the result does not directly depend on the number of participants. Instead, each processor usually computes the result using information gathered from just few nearby neighbors. Because communication is restricted to neighbors, a local algorithm does not require message routing, performs all computation in-network, and in many cases is able to locally overcome failures and minor changes in the input (provided that these need not change its output). Local algorithms have been mainly studied in the context of graph related problems [2, 18, 19, 3, 17, 20, 22]. Most recently, a local algorithm was presented [23] which demonstrated that local algorithms can be devised for complex data analysis tasks, specifically, data mining of association in distributed transactional database. The algorithm presented in [23] features local pruning of false propositions (candidates), in-network mining, asynchronous execution, and resilience to changes in the data and to partial failure during the execution of the algorithm.

In this work we develop a local algorithm that solves a specific version of FLP. One where uncapacitated resources (i.e., ones which can serve any number of clients) can be placed in any k out of m possible locations. Initiating our algorithm from a fixed resource location, say, in the first k locations, we show that the computation needed to agree on single hill-climbing step – moving one resource to a free location – can be reduce to a group of majority votes. We then use a variation of the local majority voting algorithm presented in [23] to develop an algorithm which locally computes the exact same solution a hill-climbing algorithm would compute, had it been given the entire data. Our algorithm demonstrates that whenever the cost of a step is summed across the different sensors, a hill-climbing heuristic can be computed using a local, in-network algorithm.

In a series of experiments employing networks of up to 10,000 simulated sensors we prove that our algorithm has good locality, incurs reasonable communication costs, and quickly convergences to the correct answer whenever the input stabilizes. We further show that when faced with constant data updates the vast majority of sensors continue to compute the optimal solution. Most importantly, the algorithm is extremely robust to sporadic changes in the data. So long as these do not change the global result they are pruned locally by the network.

The rest of this paper is organized as follows. We first describe our notations and formally define the problem. Then, in Section 3, we give our version for the majority voting algorithm originally described in [23]. Section 4 describes local k-facility location algorithm as a demonstrator of a hill climbing algorithm. In Section 5 preliminary experimental results are described. Section 6 ends the paper with some conclusions and open research problems.

2 Notations, Assumptions, and Problem Definition

A large number N of processors are given, which can communicate with one another by sending messages. We assume that communication among neighboring processors is reliable and ordered. An assumption which can be enforced using standard numbering, ordering and retransmission mechanisms. For brevity of this paper we assume an undirected communication tree. As shown in [4], such a tree can be efficiently constructed and maintained using variations of Bellman-Ford algorithms [7, 12]. Last, we assume fail-stop failure and that when for some reason a processor is disconnected

or reconnected its neighbors are reported.

Given a database DB containing *input points* $\{p_1, p_2, \ldots, p_n\}$ and a set M of m possible *locations*, and a cost function $d : DB \times M \rightarrow \mathcal{R}^+$, the task of a *k-facility location* algorithm is to find a set of *facilities* $C \subset M$ of size k, such that the cumulative distance of points from their nearest facility $\sum_{p_i \in DB} \min_{c \in C} d(p_i, c)$ is minimized.

Relating these definitions to the example given in the introduction, the database can include the list of events occurring in the last hour. Each event would have a heuristic estimate of the importance of the event. Furthermore, each sensor would evaluate its hop distance from every relay and multiply this to the heuristic importance of each event to produce its cost. Given this input, a facility location algorithm would compute the best combination of relays such that the most important events would need not travel far before they reach the nearest relay. The less important events, we assume, would be suppressed either in the sensor which has produced them, or in-network by other sensors.

An *anytime* k-facility location algorithm is one which, at any given time during its operation, outputs a placement for the location such that the cost of this ad hoc output improves with time until the optimal solution is found. A *distributed* k-facility location algorithm would compute the same result even though DB is partitioned into N mutually exclusive databases $\{DB^1, \ldots, DB^N\}$ and each of which is deposited with a separate processor, and these are then allowed to communicate by passing message to each other. A *local* k-facility location algorithm is a distributed algorithm whose performance does not depend on N but rather corresponds with the difficulty of the problem instance at hand.

The *hill-climbing* heuristic for k-facility location begins from an agreed upon placement of the facilities (henceforth, *configuration*). Then, it finds a single facility and a single empty location, such that by moving the facility to that free location the cost of the solution is reduced to the largest possible degree. If such a step exists, the algorithm changes the configuration accordingly and iterates. If any configuration which can be stepped into by moving just one facility has a higher cost than the current configuration the algorithm terminates and outputs the current configuration as the solution.

This paper presents a local, anytime algorithm which computes the hill-climbing heuristic for k-facility location. Note that in order to apply this algorithm to any other hill-climbing problem, it is enough to describe the start point and the mechanism by which the next possible steps are created and their cost (or gain) evaluated. Having done that, the rest of the algorithm remains the same.

3 Local Majority Voting

Our k-facility location algorithm reduces the problem to a large number of majority votes. In this section, we briefly describe a variation of the local majority voting algorithm from [23] which we use as the main building block for the algorithm. The algorithm assumes that messages sent between neighbors are reliable and ordered and that processor failure is reported to the processor's neighbors. These latter assumptions can easily be enforced using standard numbering and retransmission, ordering, and heart-beat mechanisms, correspondingly. The algorithm makes not assumption on the timeliness of message transfer of failure detection.

Given a set of processors V, where each $u \in V$ contains a zero-one poll with c^u votes, s^u out of which are one, and given the required majority $0 < \lambda < 1$, the objective of the algorithm is to decide whether $\sum_u s^u / \sum_u c^u \geq \lambda$. Equivalently, the algorithm can compute whether $\Delta = \sum_u s^u - \lambda \sum_u c^u$ is positive or negative. We call Δ the number of excess votes.

The following local algorithm decides whether $\Delta \geq 0$. Each processor $u \in V$ computes the number of excess votes in its own poll $\delta^u = s^u - \lambda c^u$, and stores the number of excess votes it reported to each neighbor v in δ^{uv}, and the number of excess votes on which it has been reported by v in δ^{vu}. Processor u computes the total number of excess votes it knows of, as the sum of its own excess votes and those reported to it by the set G^u of its neighbors $\Delta^u = \delta^u + \sum_{v \in G^u} \delta^{vu}$. It also computes the number of excess votes it agreed on with every neighbor $v \in G^u$, $\Delta^{uv} = \delta^{uv} + \delta^{vu}$. When u chooses to inform v about a change in the number of excess votes it knows of, u sets δ^{uv} to $\Delta^u - \delta^{vu}$ – by that setting Δ^{uv} to Δ^u, and then sends δ^{uv} to v. When u receives a message from v containing some δ it sets δ^{vu} to δ – by that updating both Δ^{uv} and Δ^u. Processor u outputs that the majority is of ones if $\Delta^u \geq 0$, and of zeros otherwise.

The crux of the local majority voting algorithm is in determining when u must send a message to a neighbor v. more precisely, the question is when can sending a message be avoided, despite the fact that the local knowledge has changed. In the algorithm presented here there are two cases in which a processor u would send a message to a neighbor v: when u is initialized and

when the condition $(\Delta^{uv} > \Delta^u \geq 0) \vee (\Delta^{uv} < \Delta^u < 0)$ evaluates true. Note that u must evaluate this condition upon receiving a message from a neighbor v (since this event updates Δ^u and the respective Δ^{uv}), when its input bit switches values, and when an edge connected to it fails (because then Δ^u is computed over a smaller set of edges and thus may change). This means the algorithm is event driven and requires no form of synchronization.

The analysis in [23] reveals that the good performance of the algorithm above, in terms of message load and convergence time, stems directly from its locality. The average (as well as the worst) processor would terminate after it has collected data from just a small number of nearby neighbors – its environment. The size of this environment depends on the difference, in the nearby surrounding of the processor, between the average vote and majority threshold. If the two differ by as much at five percent then the size of the environment can be expected to be limited to a few dozens. It should be noted that in the worst case, that of a complete tie, all votes must be counted and the algorithm would become global.

In order to use the local majority voting algorithm for k-facility location we slightly modify it. We add the ability to suspend and reactivate the vote using corresponding events. A processor whose voting has been suspended will continue to receive messages and to modify the corresponding local variable, but will not send any messages. When the vote is activated the processor will always check if it is required to send a message as a result of the information received while in suspended state. The pseudo-code of the modified algorithm is given in Alg. 1.

Algorithm 1 Local Majority Vote

Input of processor u: the local poll c^u, and support s^u, its set of neighbors G^u
Global constants: the majority threshold λ
Local variables: $\forall v \in G^u : \delta^{uv}, \delta^{vu}$, $active^u$.
Definitions: $\delta^u = s^u - \lambda c^u$, $\Delta^u = \delta^u + \sum_{v \in G^u} \delta^{vu}$, $\Delta^{uv} = \delta^{uv} + \delta^{vu}$
Initialization:
$\delta^{uv} = \delta^{vu} = 0$, $\forall v \in G^u$, $active^u = false$
SendMessage(v) $\forall v \in G^u$
On activate: set $active^u \leftarrow true$
On suspend: set $active^u \leftarrow false$
On RecieveMessage δ from $v \in G^u$: $\delta^{vu} \leftarrow \delta$
On notification of failure of $v \in G^u$:
$G^u \leftarrow G^u \setminus \{v\}$
On notification of a new neighbor v:
$G^u \leftarrow G^u \cup \{v\}$
On any of the above events and on change in δ^u:
For all $v \in G^u$, if $(\Delta^{uv} > \Delta^u \geq 0) \vee (\Delta^{uv} < \Delta^u < 0)$ then SendMessage(v)
Procedure SendMessage(v):
If $active^u = true$
– $\delta^{uv} \leftarrow \Delta^u - \delta^{vu}$
– Send $\langle \delta^{uv} \rangle$ to v

4 Majority Based k-Facility Location

The local k-facility location algorithm, which we now present, is based upon three fundamental ideas: The first is to have every processor optimistically perform hill-climbing steps without waiting for a conclusive choice of the globally optimal step. Having done such steps, the processor continues to validate the agreement of the steps it took with the globally correct one. If there is no agreement then these speculative steps are undone, and better one are chosen. The second idea is to choose the optimal step not by computing the cost of each step directly, but rather by voting for each pair of possible steps which of them is more costly (i.e., more popular). The third idea is a pruning technique by which many of these votes can be avoided altogether; avoiding unnecessary votes is essential because, as we further explain below, computing votes among each pair op optional steps might be arbitrarily more complicated than finding the best next step.

4.1 Optimistic computation of an ad hoc solution. Most parallel data mining algorithm use synchronization to validate that their outcome represents the global data $\bigcup_u DB^u$. We find this approach impractical for large-scale distributed system – specifically if one assumes that the data may change with time, and thus the global data can never be determined. Instead, when performing parallel hill-climbing, we let each processor proceed up hill whenever it had computed the best step according to the data it currently possesses. Then, we use local majority voting (as we describe next) to make sure that processors which have taken erroneous steps would eventually be corrected. In the event that a processor gets corrected computation associated with configurations that were wrongly chosen is put on hold. These configurations are put aside in a designated cache in case additional data that will accumulate would prove them correct after all.

We term the sequence of steps selected by processor u at a given point in time its *path* through the space of possible configurations and denote it $R^u = \langle C_1^u, C_2^u, \ldots, C_l^u \rangle$. C_1^u is always chosen the k first lo-

cations in M. C_l^u is the ad hoc solution C^u. u refrains from developing another configuration following a given C_l^u when no possible step can improve on the cost of the current configuration, or when two or more steps still compete on providing the best improvement.

Since the computation of all of the configurations a long every processor's path is concurrent , messages sent by the algorithm contain a *context* – the configuration to whom they relate. Since the computation is also optimistic, it may well happen that two processors u and v intermediately have different paths R^u and R^v. Whenever u receives a message in the context of some configuration $C \notin R^u$, this message is considered *out of context*. It is not accepted by u but rather is stored in u's out of context messages queue. Whenever a new configuration C enters R^u, u scans the out of context queue and accepts messages relating to C in the order by which they were received.

4.2 Locally computing the best possible step. For each configuration $C_a^u \in R^u$ processor u computes the best possible step as follows. First, it generates the set of possible configurations $Next\,[C_a^u]$, such that each member of $Next\,[C_a^u]$ is a configuration which replaces one of the members of C_a^u with a non-member location from $M \setminus C_a^u$. Next, for each $C \in \{C_a^u\} \cup Next\,[C_a^u]$ and each $p \in DB^u$, the cost incurred by of p in C is computed such that $cost\,(p, C) = \min_{x \in C} \{d\,(p, x)\}$. Finally, for every $C_i, C_j \in Next\,[C_a^u]$, where $i < j$, processor u initiates a majority vote $Majority^u_{C_a^u}\,\langle i, j\rangle$ which compares their relative costs and eventually computes $\Delta^u_{C_a^u}\,\langle i, j\rangle \geq 0$ if the global cost of C_i is larger than that of C_j (as we explain below). Correctness of the majority vote process guarantee that the best configuration $C_{i_{best}} \in \{C_a^u\} \cup Next\,[C_a^u]$ would eventually have negative $\Delta^u_{C_a^u}\,\langle i_{best}, j\rangle$ for all $j > i_{best}$, and positive $\Delta^u_{C_a^u}\,\langle j, i_{best}\rangle$ for all $j < i_{best}$. Hence, the algorithm would optimistically choose C_i as the next configuration whenever C_i has the maximal number of majority votes indicating it is the better one (even if some votes indicate differently).

To determine which of two configurations has the better cost using a majority vote we set for every processor u $\delta^u\,\langle i, j\rangle = \sum_{p \in DB^u} cost\,(p, C_i) - cost\,(p, C_j)$. This can be done for any $\delta^u\,\langle i, j\rangle \in [-x, x]$ by choosing, for example, $c = 2x$, $\lambda = 1/2$, and $s = x - \lambda$. Note that, as shown in [23], s^u and c^u can be set to arbitrary numbers and not just to zero or one. Further note that for every C_i, C_j

$$\sum_{p \in DB} cost\,(p, C_i) - \sum_{p \in DB} cost\,(p, C_j) =$$

$$\sum_u \sum_{p \in DB^u} [cost\,(p, C_i) - cost\,(p, C_j)]$$

. Hence, if the vote comparing the cost of C_i to that of C_j determine that $\Delta^u\,\langle i, j\rangle \geq 0$ then this proves the cost of C_i is larger than that of C_j.

Note that since every majority vote is performed using the local algorithm described in Sec. 3, the entire computation is also local. Eventual correctness of the result and the ability to handle changes in DB^u or G^u also follow immediately from the corresponding features of the majority voting algorithm.

4.3 Pruning the set of comparisons. The subsections above shows how it is possible to reduce k-facility location into a set of majority votes. However, the reduction they offer overshoots the objective of the algorithm. This is because while a k-facility location really only require that the *best* possible configuration is calculated given a certain configuration, the reduction above actually computes a *full order* on the possible configurations. This is problematic because for some inputs computing a full order may be arbitrarily more difficult (and hence, less local and more costly) than computing just the best option.

To overcome this problem the algorithm is augmented with a pruning technique which limits the progress of comparisons such that only a small number of them actually take place. Given a configuration C, processor u denote its *best* possible configurations the ones with maximal number of majority votes indicating that they are less costly than other possible configurations. It denotes *contending* those possible configurations which are indicated to be less costly than one of the best configurations. Processor u keeps track of its best and its contending configurations and of the best and contending configurations of its neighbors in G^u. For this purpose u reports with every message it sends which configurations it currently considers best or contending. u retain majority votes that compare a configuration to either its own or its neighbors' best and contending configurations active. The rest of the majority votes it suspends, meaning that u would not send messages relating to them even if it accepts messages or the data changes. Needless to say, such pruning technique might cause the algorithm to halt before it reaches the correct result. We will publish the proof that our technique does not do so elsewhere.

4.4 Pseudocode of the algorithm. The pseudocode of the algorithm is given in Algorithm 2. The pseudocode relies on an underlying majority voting algorithm which supports the following calls: initialize, activate and suspend, recieveMessage, and sendMes-

sage. Whenever the k-facility location algorithm receives a message it first analyzes it and decides to which instance of the majority vote (if any) it should be tunneled. When an underlying majority vote sends a message, it is tunneled through the k-facility location algorithm, which adds a context to the message and then forwards it to its destination.

4.5 Generalization. We claim that the algorithm we described in this section can be generalized to any other problem that is solved using hill-climbing. To demonstrate this claim, consider a Genetic Algorithm. Genetic Algorithms implement a probabilistic search of a very large domain. The algorithm is initiated with a set of points (population) in the domain (or rather, its binary representation) and proceed as follows: In every generation (i.e., cycle or turn) the current population is ranked according to a heuristic gain function and the least promising individuals (i.e., points) discarded. Then, the remaining individuals are subjected to crossbreeding (i.e., new individuals are created from a mixture of their parents) and are subjected to some degree of random mutation. The resulting population is normalized to the same size as the starting population and then constitute the input for the next round. The process is continued until a stopping rule is satisfied, which may be as simple as a fixed number of generations.

Note that it is possible for every sensor to retain its own copy of the population at every generation and perform the same kinds of mutation and breeding. The only part were sensor need to cooperate is in ranking the population. If the gain function is the sum of local gains at each sensor (e.g., how well does the individual fit as a descriptor of the local event), then this ranking can be done by the local algorithm described above. This is because finding the m worst individuals is equivalent to finding a candidate that has m candidates whose global cost (gain) is lower then his. This m candidates would not participate in the next generation.

5 Experiments

To evaluate the performance of the algorithm we ran it on simulated networks of up to ten thousands processors using up to a thousand input points in each processor. Our experiments test for two main properties of the algorithm: Its on-going behavior when the data is constantly altered and its dependency on the different operational parameters: the number of processors, locations and facilities. We measure the performance of the algorithm using two quality metrics: the percentage of processors which compute the exact solution at any given time, and the cost of the solution computed by the average processor (using cost of the exact solution as reference). For the rest of our measurements we used bulk mode runs, where all processors were initialized at once and the data was not change during the run. Communication cost was measured in raw algorithm messages, without quantifying neither the cost of protocol headers nor the beneficial effect of buffering. We also measured convergence time, using the *time till* $X\%$ quality metric – how long does it take until 80, 90, and 95 percent of the processors reach the exact solution. And locality of the algorithm – from how many processors does the typical processor collected data before 80, 90, or 95 percent of the processors reach the exact solution.

The data we use is a bi-dimensional mixture of normal distributions. We repeat each experiment several dozens of times, synthesizing new datasets each time. The synthesized input points are then equally divided among the processors. Note that previous experiments have shown that the underlying majority voting algorithm retains its local behavior when the data is biased (i.e., nonuniform). Because the behavior of distributed algorithms may be dependent on network topology, we repeat our experiments for three different topologies: Internet-like topology generated by state-of-the-art BRITE [21] simulator, de Bruijn topology [14] that simulates a network with fixed expansion rate, and bi-dimensional Grid topology which have been used in sensor network deployment.

5.1 Ongoing clustering. In this experiment we tried to realistically simulate a typical working scenario of the algorithm, in which the distribution of the data is stationary, but the data is continuously updated over time with new samples. To simulate dynamic data which retains a stationary distribution we first wait until the result converges and then begin randomly switching the databases of different processors with one another. We switch ten pairs of databases in each simulator clock cycle, which is equivalent to changing the data of every processors during a typical edge's delay. Finally, we stop switching input points and wait until the algorithm comes to a halt.

As the results in Figure 1 show, although we change the inputs of nearly every processor before the average processor have a chance to report the change to its neighbors, no more than two percents of the processors are wrong about the result during the entire dynamic period. Furthermore, the cost of the result computed by the average processor (exact or not) is about 0.15% larger than that of the exact solution. Finally, rapidness in which the algorithm converges once we stop changing the data hints that the processors that are wrong about the result are never really far from the exact one.

Algorithm 2 Local k-Facility Location

Global constants: the number of facilities k, the set M of m possible locations
Input of processor u: a database DB^u containing points $[p_1^u, p_2^u, \ldots]$, the set of its neighbors G^u, the distances of points from the possible locations $d : DB^u \times M \to \mathbb{R}^+$
Local variables:
A vector $R^u = \langle C_1^u, \ldots, C_l^u \rangle$ of configurations, where $C_i^u \subset M$ and $|C_i^u| = k$
A message queue $OutOfContext^u$
For each configuration $C \in R^u$:
– a set of majority votes referred to as $Majority_C^u \langle i,j \rangle$ with corresponding inputs $\delta_C^u \langle i,j \rangle$ and outputs $\Delta_C^u \langle i,j \rangle$
– for each neighbor $v \in G^u$ a set $B\&C_C^{vu}$
Definitions:
For each $p \in DB^u$ and $C \subset M$, $cost\,(p, C) = \min\limits_{x \in C} \{d\,(p, x)\}$
$Next\,[C] = \{C\} \cup \{\forall i \in C, j \in M \setminus C : C \setminus i \cup j\}$
For any C and $C_i, C_j \in Next[C]$, $cost_C^u \langle i,j \rangle = \sum\limits_{p \in DB^u} cost\,(p, C_i) - cost\,(p, C_j)$.
For any $C \in R^u$ and $C_i \in Next\,[C]$,
$Better_C\,(C_i) = \{C_j \in Next\,[C] : (i < j \wedge \Delta_C^u \langle i,j \rangle < 0) \vee (i > j \wedge \Delta_C^u \langle j,i \rangle \geq 0)\}$
For any $C \in R^u$ $Best\,[C] = \arg \max\limits_{C_i \in Next[C]} |Better_C\,(C_i)|$
For any $C \in R^u$ $Contending\,[C] = \{C_i \in Next\,[C] : \exists C_j \in Best\,[C]\ \ such\ that\ C_j \in Better_C\,(C_i)\}$
For any $C \in R^u$ $Active\,[C] = Best\,[C] \cup Contending\,(C_a^u) \bigcup \left(\bigcup\limits_{v \in G^u} B\&C_C^{vu} \right)$
Initialization:
$R^u = \langle C_1^u \rangle$ such that C_1^u are the k first items of M. $B\&C_{C_1^u}^{vu} = \emptyset$, for all $C_i, C_j \in Next\,[C_1^u]$ such that $i < j$, initialize a majority vote $Majority_{C_1^u}^u \langle i,j \rangle$ with input $\delta_{C_1^u}^u \langle i,j \rangle = cost_{C_1^u}^u \langle i,j \rangle$
On message $C, \langle i,j \rangle, \delta, B\&C^v$ **from** $v \in G^u$**:**
If $C \notin R^u$ enqueue the message in $OutOfContext^u$
Else $C = C_a^u \in R^u$ for some a
– Call **RecieveMessage(δ)** in the context of the majority vote $Majority_{C_a^u}^u \langle i,j \rangle$
– Set $B\&C_{C_a^u}^{vu} \leftarrow B\&C^v$
Call **on change** for C_a^u
On SendMessage δ **from** $Majority_{C_a^u}^u \langle i,j \rangle$ **to some** $v \in G^u$
– Send $C_a^u, \langle i,j \rangle, \delta, Best\,(C_a^u) \cup Contending\,(C_a^u)$ to v
On notification of failure of $v \in G^u$: $G^u \leftarrow G^u \setminus \{v\}$. For all $C_a^u \in R^u$ call **on change** C_a^u
On notification of a new neighbor v**:** $G^u \leftarrow G^u \cup \{v\}$. For all $C_a^u \in R^u$ call **on change** C_a^u
On change in C_a^u**:**
Send **activate** message to every $Majority_{C_a^u}^u \langle i,j \rangle$ such that $C_i \in Active\,[C_a^u]$ and **suspend** message to the rest
If $Best\,(C_a^u) = C_a^u$ then purge from C_{a+1}^u through C_l^u from R^u and set $l = a$
Else if $Best\,(C_a^u)$ contains a single configuration C_b then
– if $C_{a+1}^u \neq C_b$
– – purge C_{a+1}^u through C_l^u from R^u
– – append C_b to R^u as C_{a+1}^u and set $l = a + 1$
– – for all $C_i, C_j \in Next\,[C_b]$ such that $i < j$, initialize a majority vote $Majority_{C_b}^u \langle i,j \rangle$ with $\delta_{C_b}^u \langle i,j \rangle = cost_{C_b}^u \langle i,j \rangle$
– – call **on change** C_{a+1}^u
Else $Best\,(C_a^u)$ contains more than one configuration
– purge C_{a+1}^u through C_l^u from R^u and set $l = a$
Output C_l^u as the ad hoc solution

Figure 1: Behavior in dynamic environment, de Bruijn topology, $N = 2048$, $n/N = 100$, $m = 10$, $k = 2$

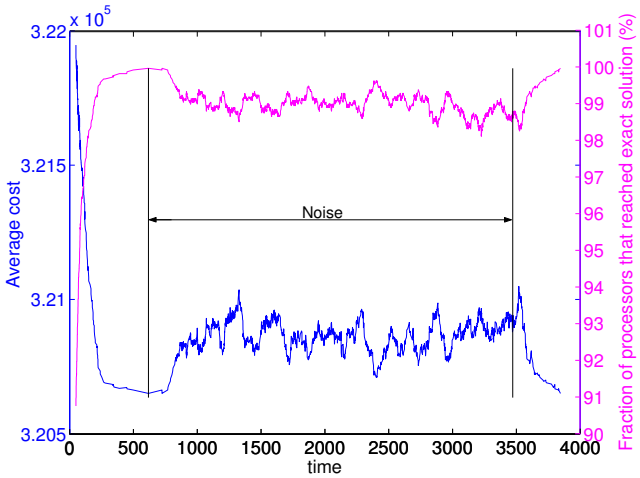
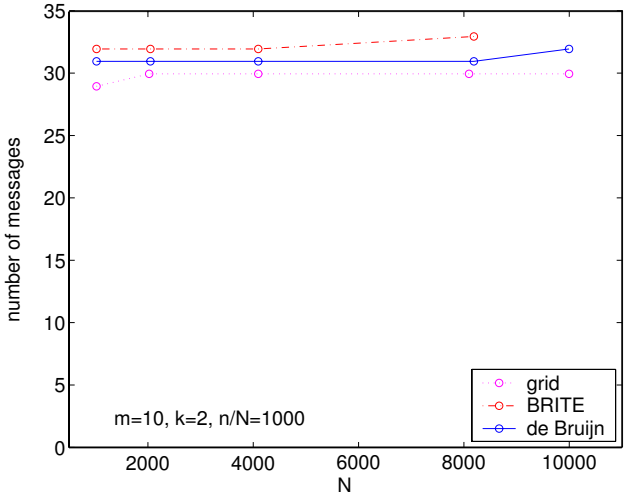


Figure 2: Messages per processor



5.2 Communication cost. We evaluate the communication cost of the algorithm by counting the number of messages sent by the average processor during an entire bulk run. To overcome the effect of paths of different lengths we show in Figure 2 the number of messages per hill-climbing step. Communication load is measured in terms of raw messages. We neither took into account the effect of protocol headers, nor have tried to piggyback messages with on another so as to reduce their count.

Figure 2 depict the number of messages the for three different network topologies described earlier. As can be seen, each processor sends about thirty messages per step. Considering that to make a step in this experiment the best of 17 possible configuration was to be found, these costs seem very reasonable. We expect that using buffering and piggybacking techniques these costs can be further reduced. It is of utmost importance to note that communication costs are almost unaffected by the network size.

5.3 Convergence time and locality. Scalability is the most important property of an algorithm intended for large-scale distributed systems. We execute the algorithm on networks of between one thousand and ten thousands processors. As the results in Figure 3 show, the time until 80, 90, and 95 percent of the processors compute the exact result does not depend on the number of processors in the system. For a grid topology (Figure 3(b)) the time is almost constant, no matter what the size of the network is. We note that in the grid topology the algorithm's performance seem unaffected by scale. In BRITE and de Bruijn (Figure 3(a)) and 3(c), respectively), performance changes in a non systematic way. We attribute this to differences between the networks generated for experiments with different sizes and to the random choice of a spanning tree. As the maximal degree of processors in the network increases, the spanning tree would be more affected by the random choices made in its construction.

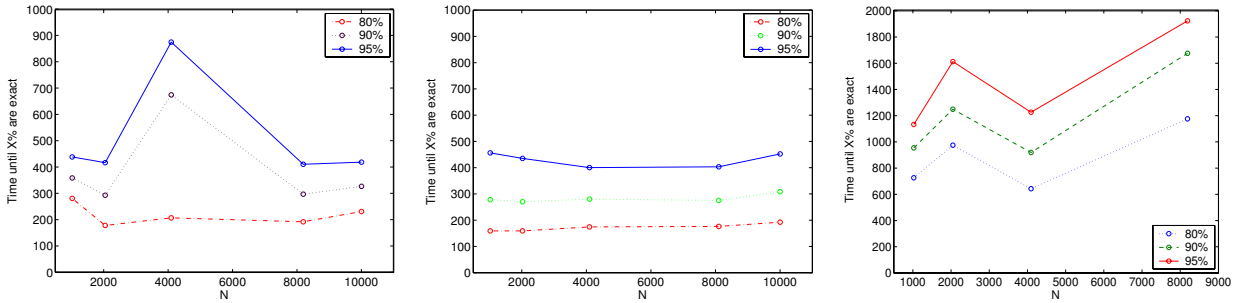
The following set of figures, Figure 4(a) through 4(c), explains the scale-up results. As can be seen, the number of input points a processor learns of, per single majority vote, is almost constant. Since this number represents a dependence on a nearly constant number of other processors, runtime cannot vary too much. For instance, for 95 percent of the processors to derive the exact result in a BRITE topology the average processor needs to collect the input of just five other processors. The time this takes can vary from that of an average edge to five times this figure.

6 Conclusions and Further Research

We have described a new k-facility location algorithm suitable for sensor networks. The qualities which qualifies the algorithm for this kind of systems are its message efficiency, its strong local pruning, and its ability to efficiently sustain failures and changes in the input. All these qualities stems from the algorithm's *local* nature.

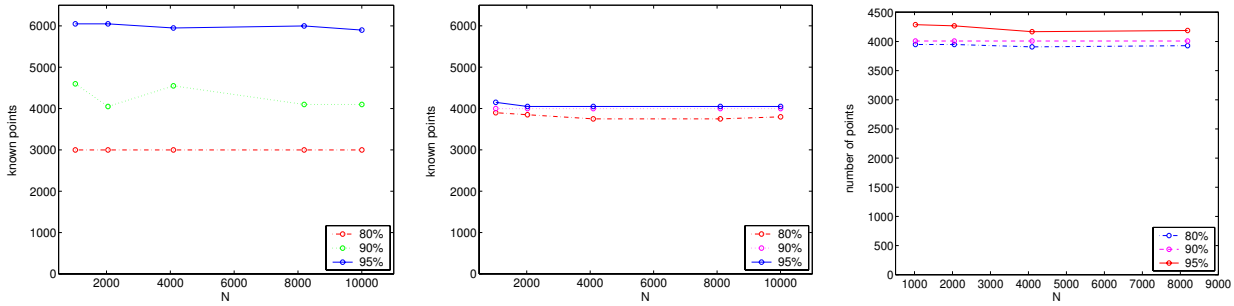
Beside its immediate value, the algorithm serves to demonstrate that various data mining problem can be solved in a sensor-network setting through reduction to basic operators of the kind of a majority vote. These operators can later be solved by efficient local algorithms. we believe in-network data mining can be

Figure 3: Convergence time ($m = 10$, $k = 2$, $n/N = 1000$)



(a) BRITE (b) grid (c) de Bruijn

Figure 4: Locality measurement ($m = 10$, $k = 2$, $n/N = 1000$)



(a) BRITE (b) grid (c) de Bruijn

come one of the key techniques by which the output of these systems is accessed.

The paper also pose some interesting research questions: Can the more strict formulations of k-means clustering also be solved locally? If not, what are the characteristics of a data mining problem which make it suitable for a local algorithm? Can the dependency of our algorithm on local failure detection be avoided? What other paradigms can be used to solve such problems in large-scale distributed system? How can approximation (deterministic or probabilistic) be used in order to improve the performance of algorithms intended for large-scale distributed systems? We hope to be able to answer some of these problems in future research.

References

[1] Vijay Arya, Naveen Garg, Rohit Khandekar, Kamesh Munagala, and Vinayaka Pandit. Local search heuristic for k-median and facility location problems. In *ACM Symposium on Theory of Computing*, pages 21–29, 2001.

[2] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Compact distributed data structures for adaptive network routing. *Proc. 21st ACM STOC*, May 1989.

[3] Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction (extended abstract). In *Proceedings of the 32nd annual symposium on Foundations of computer science (FoCS)*, pages 268 – 277, 1991.

[4] Y. Birk, L. Liss, A. Schuster, and R. Wolff. A local algorithm for ad hoc majority voting via charge fusion. In *Proceedings of the 18th annual conference on distributed computing*, 2004.

[5] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for the facility location and k-

median problems. In *IEEE Symposium on Foundations of Computer Science*, pages 378–388, 1999.

[6] Inderjit S. Dhillon and Dharmendra S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Large-Scale Parallel Data Mining*, pages 245–260, 1999.

[7] L.R. Ford and D.R. Fulkerson. *Flows in Networks*. Princton University Press, 1962.

[8] George Forman and Bin Zhang. Distributed data clustering can be efficient and exact. *SIGKDD Explor. Newsl.*, 2(2):34–38, 2000.

[9] D. Foti, D. Lipari, C. Pizzuti, and D. Talia. Scalable Parallel Clustering for Data Mining on Multicomputers. In *3rd Workshop on High Performance Data Mining. In conjunction with International Parallel and Distributed Processing Symposium 2000 (IPDPS'00)*, Cancun, Mexico, May 2000.

[10] Guha and Khuller. Greedy strikes back: Improved facility location algorithms. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1998.

[11] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on inforamtion Theory*, 46(2):388 – 404, 2000.

[12] J.M. Jaffe and F.H. Moss. A responsive routing algorithm for computer networks. *IEEE Transactions on Communications*, pages 1758–1762, July 1982.

[13] Kamal Jain and Vijay V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *IEEE Symposium on Foundations of Computer Science*, pages 2–13, 1999.

[14] F. Kaashoek and D. Karger. Koorde: A simple degree-optimal distributed hash table, February 2003.

[15] Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan. A microeconomic view of data mining. *Data Mining and Knowledge Discovery*, 1998.

[16] Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 1–10. Society for Industrial and Applied Mathematics, 1998.

[17] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be computed locally! In *Proceedings of the 23rd Symposium on Principles of Distributed Computing (PODC)*, July 2004.

[18] S. Kutten and B. Patt-Shamir. Time-adaptive self-stabilization. *Proc. of the 16th Annual ACM Symp. on Principles of Distributed Computing*, pages 149–158, August 1997.

[19] S. Kutten and D. Peleg. Fault-local distributed mending. *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing*, August 1995.

[20] N. Linial. Locality in distributed graph algorithms. *SIAM J. Comp.*, 21:193–201, 1992.

[21] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. Brite: An approach to universal topology generation. In *Proceedings of Ninth IEEE MASCOTS '01 (Tools track)*, pages 346–353, Cincinatti, August 2001.

[22] M. Naor and L. Stockmeyer. What can be computed locally? *25th ACM Symposium on Theory of Computing*, pages 184–193, 1993.

[23] R. Wolff and A. Schuster. Association rule mining in peer-to-peer systems. In *Proc. of the IEEE Conference on Data Mining ICDM*, Melbourne, Florida, 2003.