Efficient Information Compression in Sensor Networks

Song Lin *

Vana Kalogeraki

Dimitrios Gunopulos

Stefano Lonardi

Department of Computer Science and Engineering,
University of California, Riverside, CA, 92521, USA
E-mail: {slin, vana, dg, stelo}@cs.ucr.edu
*Corresponding author

Abstract: In the emerging area of wireless sensor networks, one of the most typical challenges is to retrieve historical information from the sensor nodes. Due to the resource limitation of sensor nodes (processing, memory, bandwidth, and energy), the collected information of sensor nodes has to be compressed quickly and precisely for transmission. In this paper, we propose a new technique -- the ALVQ (Adoptive Learning Vector Quantization) algorithm to compress this historical information. The ALVQ algorithm constructs a codebook to capture the prominent features of the data and with these features all the other data can be piece-wise encoded for compression. In addition, we extend our ALVQ algorithm to compress multi-dimensional information by transforming the multi-dimensional data into one-dimensional data array. Finally, we consider the problem of transmitting data in a sensor network while maximizing the precision. We show how we apply our algorithm so that a set of sensors can dynamically share a wireless communication channel.

Keywords: data compression, sensor networks, bandwidth allocation, multi-dimensional information compression.

1 INTRODUCTION

The ever changing developments in electrical embedded systems have enabled the widespread deployment of sensor networks consisting of small sensor nodes with sensing, computation, and communication capabilities. The sensor nodes can monitor various characteristics of the environment such as temperature, humidity, pressure, light, sound, chemicals, noise levels, radioactivity, movement, etc. The applications of sensor networks have been seen in a large variety of areas. For example, the sensor networks can help biologists automatically recognize and track different species of birds. The environmental scientists can utilize sensor networks to monitor and record the development of environmental conditions. In the intelligent building, sensors are deployed in offices and hallways to measure temperature, noise, light, and interact with the building control system. In the battlefields, soldiers equipped with sensors can be easily tracked and organized by the commander.

Suppose there is a sensor network with a base station (sink) and N sensors. An interesting application is to let the sink collect the historical information from each sensor to perform some query processing or statistical analysis. Several constraints of sensor networks make the information retrieval implementation difficult. The first constraint is the resource (i.e. memory, bandwidth and power, etc) limitation of sensor node. Thus it is impractical to transmit the original data feed from each sensor to the base station. The second constraint is communication channel sharing. When transmitting data to the sink, several sensors have to share the communication channel. In this paper, we address the historical information retrieval problem by addressing the following sub-problems: a) At each sensor, how to

compress its historical information given a bandwidth allocation? b) At each sensor, how to compress the multi-dimensional sensor information? c) In the sensor network, how to allocate bandwidth to sensors in order to maximize and balance their compression qualities?

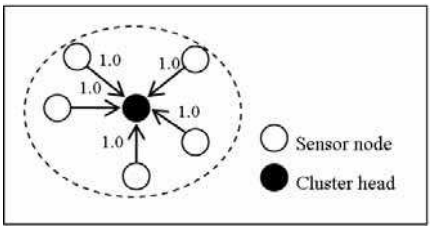

Figure 1 *The LEACH model*

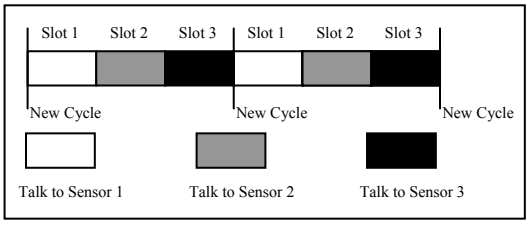

Figure 2 *A sample communication channel*

An efficient data compression technique SBR (Self Based Regression) [6] has been proposed recently and has been shown to work better than other well-known techniques in sensor network settings. A problem of SBR is that its codebook is not precise enough for compression. In addition the codebook updating in SBR wastes a lot communication bandwidth. As to the network topology, a large amount of energy-saving algorithms [7][11][12] group nearby sensors into clusters. All the sensors in the same cluster share the cluster head's communication channel evenly (as in Figure 1), that is, they utilize the same data transmission bandwidth if they want to talk to the cluster head. For example, in TDMA scheduling, a different time slot is assigned to each sensor (as in Figure 2), while in FDMA scheduling, a different frequency range slot is allocated to each sensor. For our compression problem, as the information is distributed differently at different sensors, the compression qualities show large variances among sensors given the same bandwidth allocation. Since the bandwidth is an important factor for increasing compression qualities, if we could assign more bandwidth to the low quality sensors and assign less bandwidth to the sensors whose compression qualities are high enough, the overall compression qualities of all sensors would be maximized and balanced. In this paper we address this problem and present a Dynamic Bandwidth Assignment algorithm to solve it.

1.1 Our contributions

Our contributions are summarized as follows:

1. We apply the LVQ (Learning Vector Quantization) algorithm to construct the codebook for data compression. Our results show that the LVQ learning process can further improve the codebook for high compression precision.

2. We introduce the concept of two-level regression for higher precision compression. The two-level regression is applied to the codebook update in order to save more bandwidth while keeping the codebook updated with high precision.

3. We extend our ALVQ algorithm to compress multi-dimensional sensor information. We propose an efficient algorithm to transform multi-dimensional information into one-dimensional data array and compress the transformed information with high precision.

4. We consider the problem of dynamic allocation of bandwidth in wireless sensor networks. We present a new algorithm, DBA (Dynamic Bandwidth Assignment), which works in combination with our compression algorithm, and dynamically allocates bandwidth among channel sharing sensors. The algorithm uses the recent history to predict future bandwidth requirements and balance the expected loss for the different sensors.

The rest of this paper is organized as follows. Section 2 states the background and definition of our problem. In Section 3 we present the related work and we describe our ALVQ algorithm in Section 4. We extend our ALVQ algorithm to compress multi-dimensional sensor information in Section 5 and the dynamic bandwidth allocation problem is addressed in section 6. In Section 7 we provide our experimental results. Finally, we conclude our remarks and future work in Section 8.

2 DATA MODEL AND PROBLEM DEFINITION

In sensor networks, each sensor has the capability of measuring, communicating and computing. It continuously collects measurements from its local environment, processes the data and transmits the results through multiple sensor hops to the base station through other sensors. Among all the resource limitations of the sensors (CPU, memory, bandwidth, energy, etc), the communication and power consumptions turn out to be the main bottleneck of the sensor networks' capability. Since a major source of energy consumption in a sensor node is the data transmission process, the design of data reduction techniques that effectively reduce the amount of data transmitted in the network is essential in order to meet the transmission bandwidth constraints and increase the network lifetime.

A sensor node S is equipped with a measuring system which generates a data record $r = (t, val1, val2,\ldots)$ every ε seconds, where t is the timestamp on which the record was generated, and $(val1, val2,\ldots)$ are the measurements at that time instance. The sensor has its local data buffer $\boldsymbol{B}$ which stores these records. When $\boldsymbol{B}$ is full, the sensor compresses the data and transfers the compression representative to the sink. In this paper, we address the following three problems:

Problem I: Given a one dimensional time series data array $\boldsymbol{X}$ collected by a sensor, the goal is to find a proper encoder

function F making $\boldsymbol{Y} = F(\boldsymbol{X})$ and a decoder function G, so that

a) $| \boldsymbol{X} | / | \boldsymbol{Y} | \geq R$ and b) $\| \boldsymbol{X} - G(\boldsymbol{Y}) \|$ is minimized.

In a) R is the compressing rate determined by to application specifications. In b) the distance between the retrieval values from the compressed information and the original values is minimized.

Problem II: Given the multi-dimensional geographical information, the goal is to transform it into one dimensional data array and compress it with high compression quality.

Problem III: Given a sensor network cluster $\boldsymbol{C}$ with k sensors, given the historical compression and transmission statistics in the cluster, the goal is to dynamically allocate different bandwidth to different sensors, such that the overall compression qualities of all k sensors in the cluster are maximized and balanced.

3 DICTIONARY LOOKUP SCHEMES

In high rate lossy compressions, Dictionary Lookup schemes [4] are widely used in graphics, pattern recognition, etc. In such schemes, a codebook (or base signal) captures the prominent patterns of the data. For each data piece to be compressed, we "look up" the codebook, find the best approximation pattern and then use the approximation parameters to represent the original data piece.

The characteristics of sensor data make the Dictionary Lookup Scheme very appealing for compression. Firstly, the data in sensor networks is collected from the environment and therefore is likely to show similar patterns over time. Secondly, some sensor nodes collect different measurements at the same time. These measurements show intrinsic correlation between each other, as is the case between pressure and humidity in weather monitoring systems.

Therefore, Dictionary Lookup Scheme is a good choice for historical information compression in sensor networks.

3.1 Piece-wise approximation

In sensor networks, many physical quantities are correlated, like air temperature, pressure etc. Therefore we can use piece-wise linear regression to capture those properties.

3.1.1 Piece-wise linear regression

Given two time series data pieces, $\boldsymbol{X}$ and $\boldsymbol{Y}$, we use $\boldsymbol{X}$ to approximate $\boldsymbol{Y}$ by piece-wise linear regression, that is $\boldsymbol{Y'} = a*\boldsymbol{X} + b$ so that the regression error $\| \boldsymbol{Y} - \boldsymbol{Y'} \|$ is minimized. In sensor networks, many quantities show strong linear relationships between each other and the quantities themselves show similar patterns over time. Therefore, if we choose the most prominent patterns, we can piece-wise approximate other patterns with high precision.

In Figure 3, there are two time series $\boldsymbol{X}$ and $\boldsymbol{Y}$. If we let $\boldsymbol{Y'} = 3\boldsymbol{X} - 8$, then $\boldsymbol{Y'}$ is a good approximation of $\boldsymbol{Y}$.

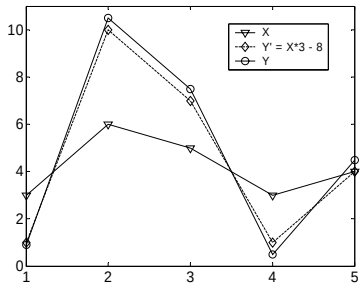

Figure 3 *Piece-wise approximation*

3.1.2 Two-level piece-wise linear regression

Given three vectors of same size, $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$, we could use linear combination of $\boldsymbol{X}$ and $\boldsymbol{Y}$ to approximate $\boldsymbol{Z}$:

1. We piece-wise approximate $\boldsymbol{Z}$ by $\boldsymbol{X}$, that is $\boldsymbol{Z'} = a*\boldsymbol{X} + b$ so that the regression error $\| \boldsymbol{Z} - \boldsymbol{Z'} \|$ is minimized.
2. We construct a new vector $\boldsymbol{D}$ to store the difference of Vector $\boldsymbol{Z}$ and the approximation vector $\boldsymbol{Z'}$, $\boldsymbol{D} = \boldsymbol{Z} - \boldsymbol{Z'}$.
3. Employ $\boldsymbol{Y}$ to approximate the difference vector $\boldsymbol{D}$ we obtained above, that is $\boldsymbol{D'} = c*\boldsymbol{Y} + d$, so that the regression error $\| \boldsymbol{D} - \boldsymbol{D'} \|$ is minimized.
4. The compression parameters a, b, c, d are transmitted as the compression representative of $\boldsymbol{Z}$.

It should be noted that *2-level Piece-wise Linear Regression* is at least as good as standard *Piece-wise Linear Regression*. Usually it is much more precise than its 1-level counterpart. It could be utilized in applications where higher compression precision is required.

3.2 Previous work: LVQ (Learning Vector Quantization)

LVQ algorithm [8] is used in many applications such as image and voice compression, voice recognition, etc.

Assume there is a training data set with n vectors in k classes. Each vector has the class label indicating which class it belongs to. We want to construct a set of representative codeword vectors (called codebook) to classify the data in real world. Each new vector to be classified finds the nearest codeword vector in the codebook and is assigned the same class as that codebook vector.

Unfortunately designing a codebook that best represents the set of training vectors is NP-hard [13]. We therefore resort to a suboptimal codebook design scheme − LVQ (Learning Vector Quantization). First, we randomly select a training vector from each class. We denote theses vectors as $m_{1,\ldots}m_k$ and take them as raw codebook. For every other training vector x we perform the following learning process:

for $i = 1, \ldots k$,

$m_i = m_i + a(t)[x - m_i]$ *if x and* m_i *are in the same class;*

$m_i = m_i - a(t)[x - m_i]$ *if x and* m_i *are in different classes;*

Here $0<a(t)<1$, and $a(t)$ may be constant or decrease monotonically with time.

LVQ continuously adjusts its codebook with all the training data, so that the codebook is trained nearer to the optimal representative of each class.

3.3 Previous work: SBR (Self Based Regression)

Because the historical information in sensor networks shows similar patterns over time and different measurements show natural correlation between each other, the SBR (Self Based Regression) algorithm was recently proposed in [6] to exploit these characteristics of the data. The authors show it outperforms other standard approximation techniques such as DCT, Wavelets and Histograms in precision. The SBR algorithm extracts the prominent feature from the training data to construct the codebook and the base station keeps a codebook for each sensor. When the data buffer at some sensor node is full, the collected data is compressed by piece-wise regression and the update codebook data piece is calculated. These results are then transmitted to the base station where the approximation of sensor data is retrieved.

Compared with SBR, our ALVQ algorithm increases the compression precision by improve the codebook accuracy. In addition, ALVQ compresses the update of codebook too, which saves bandwidth for data transmission and increases the accuracy of the approximation.

4 THE ALVQ FRAMEWORK

We now present the *Adoptive LVQ* (ALVQ) algorithm for compressing historical information in sensor networks.

The ALVQ algorithm receives as input the latest n (size of data buffer in sensor node) data values, a bandwidth constraint *TotalBand* (number of values to transmit, including any codebook update values), the maximum size of the codebook M_{code} and the current codebook C_{base} of size $|C_{base}| < M_{code}$. The details of ALVQ algorithm are described below,

The ALVQ framework

1. Create the raw codebook from training data set at each sensor
2. Utilize Learning Vector Quantization algorithm to polish the codebook
3. Transmit the codebook to the base station
4. Let the sensor collect data and fill the local buffer
5. When the buffer is full, compute the codebook update and compress the collected data
6. Each sensor transmit the compressed codebook update and the compressed data to the base station
7. Flush the buffer and go to step 4

Our ALVQ algorithm works in the following ways: First, in the codebook construction, ALVQ performs a LVQ (Learning Vector Quantization) learning process on the codebook, which adjusts the codebook, to be nearer to the optimal codebook. Second, for codebook updates, ALVQ compresses the codebook updates and transfers the compressed data to the base station. Using 2-level piece-wise regression, ALVQ can compress the update with high precision while saving more bandwidth for data transmission in order to increase the quality of the approximation.

4.1 Codebook construction from training dataset

A training dataset is needed to construct the "raw" codebook. The raw codebook can be generated as follows. We divide the training data into several Data Pieces (*DP*s) each with same size W ($W = n^{1/2}$). Each *DP* could then be approximated by another *DP* using piece-wise regression. According to the memory limitations of the sensor nodes, the size of codebook is restricted to some limitation *Mcode*. We take the first (*Mcode*/W) *DP*s that can best approximate the other *DP*s by piece-wise regression as the "raw" codebook. This is implemented by repeatedly finding in the training dataset the top *DP* that can improve the approximation precision most if inserted into the codebook.

When the "raw" codebook is full, we perform the following LVQ learning process to polish the codebook:

For each *DP* $\boldsymbol{X}$ in the training dataset, we find the best *CDP* (Codebook Data Piece) in the codebook that can approximate it with the smallest error. We denote this *DP* in the codebook as CDP_i, and update CDP_i:

$$CDP_i = CDP_i + \alpha\,[(\boldsymbol{X}-\boldsymbol{b})/\boldsymbol{a} - CDP_i],$$

where $\boldsymbol{a}$, $\boldsymbol{b}$ are the regression parameters; and $0 < \alpha < 1$ is the training parameter.

After all the *DPs* in the training dataset have been tested, the codebook is adjusted and transmitted to the base station.

4.2 Compressing time series data in real world

After the training process is finished, the sensor node collects measured data continuously and adds it to its buffer. When the buffer is full, it divides the buffer data into several intervals each having the same size W ($n^{1/2}$). First, it maps each interval to the best *CDP* (Codebook Data Piece) in the codebook using piece-wise regression. Then it finds the interval with the largest regression error and divides it into half. If the data interval size is smaller than W, it can shift in the *CDP* until the best approximation is achieved. We keep dividing the intervals with the highest error until the maximum number of intervals is achieved (depending on the bandwidth and the buffer size). Then the approximation parameters are transmitted to the base station as compression representative. For each interval, the regression parameters are *start* (start point of the approximated interval), *length* (length of the interval), *shift* (the part of the codebook that is used for approximation), a, b, *error* (regression parameters and error).

4.3 Codebook update

Because the environmental data feature may change over time, as a new data stream is collected at the sensor node, old data patterns in the codebook may become out of date and inappropriate for the new data regression. Thus we need to insert new frequently occurring patterns into the codebook and remove out-of-date patterns. Please note that

it is not always desirable to update the codebook with too many new patterns. Since the transmission bandwidth to the base station is upper bounded by *TotalBand*, the more new *CDPs* we use to update codebook, the less bandwidth there is left that can be utilized for data transmission.

The algorithm works as follows:

1. The sensor first creates a new "raw" codebook on the new data that the sensor measures.
2. Then the sensor applies LVQ learning process to adjust the "raw" codebook to make the approximation better.
3. After that ALVQ compares the new codebook with the old codebook and determine the appropriate number of *CDPs* to be ultimately inserted into the old codebook. We call these *CDPs* as the codebook update.
4. Having found the exact set of *CDP*s to be inserted into the old codebook, a two-level regression subroutine is called to compress these *CDP*s and the compression representatives are transmitted to the base station. For two-level regression, each *CDP* has 6 parameters. Therefore, there are totally $N_{CDP}\times 6$ representative values.
5. Since the codebook size is upper-bounded by M_{code}, out-of-date *CDP*s are replaced by newly inserted ones. Those out-of-date *CDP*s are found by Least Frequently Used (*LFU*) policy. And their positions in the codebook are transmitted to the base station too.
6. Each transmission to the base station includes exactly *TotalBand* values:
   --*The compressed N_{CDP} newly inserted CDPs* ($N_{CDP}\times 6$ values)
   --*The positions of evicted CDPs in the old codebook* (N_{CDP} values)
   --*The compression representative of original sensor data* ((*TotalBand* − $N_{CDP}\times 7$) values)
7. When the transmission is finished, the base station first retrieves the newly inserted *CDP*s to update the codebook. Then utilize the new codebook and the compression representatives to approximate the original data at the sensor node.

4.4 Comparison between ALVQ and SBR

SBR (Self Base regression) [6] uses a very similar framework for data compression. Compared with SBR, the ALVQ algorithm is more efficient: There are two main differences between them:

1. In the codebook construction phrase, SBR chooses some data pieces from the training dataset as the codebook. ALVQ also chooses some data pieces, but then does a LVQ learning process on the initial codebook, which adjusts the codebook by all the data.
2. In codebook update phrase, SBR transmits the codebook update data pieces as they are. ALVQ use old codebook to compress the codebook update, which saves bandwidth for data transmission and increases the quality of the approximation.

4.5 Computing complexity of ALVQ

According to [6], the time complexity of SBR is $O(n^{1.5})$ where n is the size of the data buffer in the sensor node. In ALVQ, for each *DP* in the buffer, our learning process finds the best codebook data piece *CDP* and adjust *CDP* in $O(W)$ time. In addition, the compression of codebook update takes $O(M_{base}\cdot W)$. Therefore, the running time complexity of ALVQ is $O(n^{1.5}+ W\cdot W+ M_{base}\cdot W) = O(n^{1.5})$ which is the same as SBR and is acceptable for real sensors.

5 COMPRESSING MULTI-DIMENSIONAL INFORMATION

So far we have presented an efficient algorithm to compress the historical information of one-dimensional measurements (like temperature, humidity, pressure, etc) which can be efficiently approximated with codebook. In this section, we extend our ALVQ algorithm to compress multi-dimensional information (geographical information).

5.1 Applying codebook lookup scheme to compress geographical information

Codebook lookup scheme is also applicable to geographical information compressions. This is due to the restrictions and regulations of the moving patterns. For example, the trains move on the trial ways, the cars move on the highways or local roads, ships and aircrafts cruise with some predefined routes. With the codebook represents these restrictions and regulations of mobile sensors, we can approximate the geographical information with high precision.

The idea to solve the multi-dimensional information compression problem is to transform the multi-dimensional data into one-dimensional data array and then apply ALVQ algorithm to compress it. We can perform the dimension reduction approach by scanning the original multi-dimensional information *record by record* (**Record Scan**) or *dimension by dimension* (**Dimension Scan**). By **Record Scan** we scan and copy all the dimensional information for each record at a time, and we scan the records sequentially one by one. By **Dimension Scan**, we scan all the records' information for one dimension at a time, and we scan different dimensional information one by one. For example, there are m records in buffer, $\boldsymbol{X} = \{\{v_{11}, v_{12}, ..., v_{1n}\}, \{v_{21}, v_{22}, ..., v_{2n}\}, ...,\{v_{m1}, v_{m2}, ..., v_{mn}\}\}$, to be transformed into a one-dimensional data array. With **Record Scan**, we get a new vector $\boldsymbol{Y} = \{v_{11}, v_{12}, ..., v_{1n}, v_{21}, v_{22}, ..., v_{2n}, ..., v_{m1}, v_{m2}, ..., v_{mn}\}$, while with Dimension Scan transformation, the resulted vector is $\boldsymbol{Y} = \{v_{11}, v_{21}, ..., v_{m1}, v_{12}, v_{22}, ..., v_{m2}, ..., v_{1n}, v_{2n}, ..., v_{mn}\}$. In order to understand **Record Scan** and **Dimension Scan** better, we can consider the m (n-dimensional) records as an $m\times n$ matrix. Each row of the matrix is a record and each column of the matrix represents all the records' information at a specific dimension. By **Record Scan** we scan the $m\times n$ matrix row by row and with **Dimension Scan** we scan the matrix column by column.

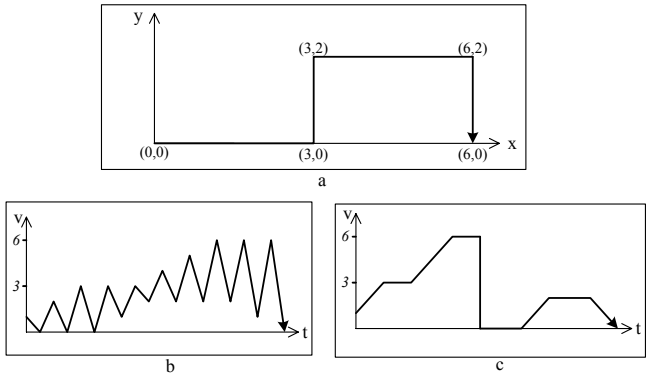

Figure 4 *(a) The original multi-dimensional information X. b) The resulted vector Y after Record Scan transformation. c) The resulted vector Y after Dimension Scan transformation.*

Although both **Record Scan** and **Dimension Scan** are able to transform multi-dimensional information into one dimensional data array, **Record Scan** transformation has some drawbacks for codebook creation and data compression. This is because **Record Scan** treats the information at different dimensions equally (recall that different dimensional information is mixed together in the resulted array in **Record Scan**). In reality, this is usually inefficient in detecting and utilizing frequent patterns. The reason is that the regulations at different dimensions are usually different, and **Record Scan** cannot efficiently grasp the different patterns in different dimensions. For example, a mobile sensor S moves in a two-dimensional space and generates position reading every second. As it is shown in Figure 4a, it starts from original point (0, 0) and moves towards east for 3 seconds, then goes towards north for 2 seconds. After that, it goes east again for 3 seconds and finally it goes south for 2 seconds. Suppose S moves in a constant speed of 1 inch/second. The generated data records are $\boldsymbol{X}$ = {(1,0), (2,0), (3,0), (3,1), (3,2), (4,2), (5,2), (6,2), (6,1), (6,0)}. After **Dimension Scan**, we get a new vector $\boldsymbol{Y}$ = {1,2,3,3,3,4,5,6,6,6,0,0,0,1,2,2,2,2,1,0} (as is shown in Figure 4c). It is easy to conclude from Figure 4c that if we create a codebook with codeword {1,2,3} and {0,0}, we can approximate the original data perfectly (i.e. we use codeword {1,2,3} to compress data piece {1,2,3}, {4,5,6}, {0,1,2}, {2,1,0} and we use codeword {0,0} to approximate data piece {3,3}, {6,6}, {0,0}, {2,2}). Therefore, with codebook size of 5 in **Dimension Scan** we can approximate $\boldsymbol{X}$ without any error. In Figure 4b, however, we cannot find a codebook with size 5 that can approximate the one-dimensional data array (computed by **Record Scan**) without error using ALVQ algorithm.

5.2 The multi-dimensional ALVQ algorithm

In order to exploit the different patterns in different dimensions for compression, we utilize **Dimension Scan** to transform multi-dimensional information into one dimensional data array and apply ALVQ technique to compress it.

Before presenting our algorithm, let us consider some properties of codebook compression. An interesting observation is that if two data sequences are similar, their codebooks are also similar. In addition, if the data sequences are linear correlated (one sequence can be piece-wise linearly approximated by another), the codebook data pieces of them are also similar after normalization (with *mean* 0 and *variance* 1). Even though different dimensions usually show their individual patterns differently, some sub-patterns at different dimensions may be similar or linear correlated. We can take advantage of these similarities and transmit only one copy of the similar sub-patterns to make the codebook more compact and save network utilization further.

The basic idea is that we transform the multi-dimensional data $\boldsymbol{X}$ into one-dimensional data array $\boldsymbol{Y}$ by scanning the dimensional information of all data records dimension by dimension. Suppose there are m data records with each including geographical information in n dimensions. Our codebook generation and data compression algorithm is described below,

Given a codebook size C_{base}, and the original data $\boldsymbol{X}$ (an $m \times n$ matrix), we divide the matrix $\boldsymbol{X}$ into n data arrays by taking each column of the matrix as an array and apply *ALVQ* technique to create codebook and compress these data arrays one by one. The codebook size for each array is initially set as the same ($C_{base}[\mathrm{i}] = \tau = C_{base}/n$).

After codebook generation and data compression for each data array, we get a $\tau \times n$ codebook data piece (*CDP*) matrix, with $C_{i1}, C_{i2}, \ldots, C_{i\tau}$ representing the codebook data pieces for the data compression at the ith dimension. We normalize all these codebook data pieces ($\forall 0<i<n$, $0<j<\tau$ mean(C_{ij}) = 0, var(C_{ij}) = 1), and the data arrays at different dimensions are compressed again using the normalized codebook. Please notice that the *ALVQ* algorithm utilize piece-wise linear regression to compress the sensor data, which will not impact compression quality when we perform linear transformation on the codebook (i.e. we perform a normalization on the codebook).

With the normalized codebook data piece (*CDP*) matrix, we search for similar *CDPs* in it. If two *CDPs* C_{ij}, C_{mn} with $\| C_{ij} - C_{mn}\| < \varepsilon$ (ε is a small threshold), we say C_{ij} and C_{mn} belong to a *Similar CDP Set* (*SCS*). After finding all *Similar CDP Sets*, we compact the codebook by keeping one copy of a *CDP* for each *Similar CDP Set*, and add a reference record wherever the similar *CDP* resides. For example, C_{ij} and C_{mn} are in a *SCS*, we keep the *CDP* C_{ij} as it was, and replace C_{mn} with a reference record (R, i, j), indicating we utilize C_{ij} as the *CDP* here.

After compressing the codebooks at different dimension, we compare the approximation qualities at different dimensions ($Q_1, Q_2, \ldots, Q_n$). If the difference of the approximation qualities for two dimensions is higher than some threshold ($|Q_i - Q_j| > T$), we adjust the codebook size at different dimension to balance the approximation qualities of them. First, we compute the average compression quality among dimensions, $Q_A = \sum_{i=1 \cdots k} Q_i/k$. Suppose the codebook size at the ith dimension is $C_{base}[i]$, we then update the

codebook size $C'_{base}[i] = C_{base}[i] - \beta(Q_i - Q_A)$, where β is an adjust parameter. With dynamic adjustment of codebook size at different dimensions, the compression qualities for different dimensions are balanced well.

6 DYNAMIC BANDWIDTH ASSIGNMENT IN SENSOR NETWORKS

Now let us consider the problem in a general case where we want to gather the historical information from all the sensors in the sensor network. Recently a large number of energy-saving algorithms have been proposed for efficient routing in sensor networks. Algorithms [5][7][11][12][14][15] are based on the collect and send scheme, treat all the sensors equally in the data transmission scheduling. In our collect-compress-transmit scenario, as the compression qualities show variances at different sensors, to assign more bandwidth priorities to low-compression-quality sensors is more appealing.

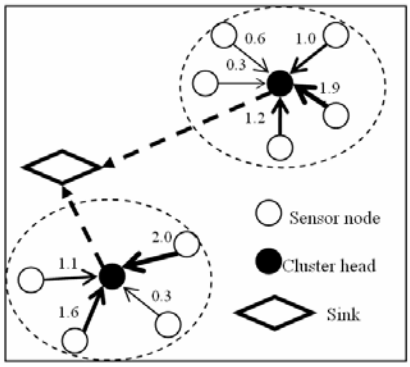

Figure 5 *Dynamic Bandwidth Assignment model*

6.1 Network topology: the LEACH framework

LEACH (Low-Energy Adaptive Clustering Hierarchy) [7] and its variants [11][12] are among the most popular hierarchical routing algorithms for sensor networks. The idea is to form clusters of the sensor nodes based on the received signal strength and use local cluster heads as routers to the sink. The cluster heads are elected from sensor nodes randomly over time in order to balance the energy dissipation of nodes.

As described in Figure 1, in *LEACH*, all the sensor nodes send packages directly to its local cluster head and the cluster head transmits these packages to the sink through multiple hops in the sensor network. Since the communication between the sensor node and the cluster head is wireless, the communication channel of the cluster head is the bottleneck of the total data transmission of all the sensors in the cluster. In *LEACH*, all the sensors in the cluster share the communication channel of the cluster head evenly (for example, TDMA schedule), namely, they share the same communication bandwidth of transmission data to the cluster head.

As different sensors usually collect different data, it is highly possible that the compression qualities at different sensors are different even given same compression rate. Since *LEACH* assigns same transmission bandwidth for all the sensors in the cluster, the compression rates for these sensor nodes are the same. Therefore, the compression qualities of different sensors cannot be maximized and balanced well.

In applications where similar compression qualities are required for all the sensors, different compression rate should be assigned to different sensors in order to maximize the overall compression qualities. As all the sensor nodes share the communication channel of the cluster head, we can take better advantage of it. We can let the cluster head assign its communication channel unevenly and dynamically to different sensors. For those sensors with low compression qualities, cluster head assigns more bandwidth to them; for those sensors with high compression qualities, less bandwidth are assigned. Therefore, with different transmission bandwidth, similar compression qualities of sensors are achieved.

This can be done by changing the channel schedule. For example, in TDMA mechanism, sensors that need more bandwidth can use the communication channel longer.

The *DBA* algorithm

1. Embed the sensors into the environment
2. Let the sensors set up clusters as in *LEACH*
3. The cluster head assigns transmission bandwidth evenly to all sensors in the cluster
4. Let the sensor collect and compress the measured data
5. The sensor transmits the compressed data to the cluster head with the bandwidth assigned to it
6. The cluster head transmits data to sink through multi-hops in sensor networks
7. The cluster head reassigns bandwidth to each sensor in the cluster
8. Go to Step 4

6.2 Dynamic Bandwidth Assignment (DBA)

Here we introduce the Dynamic Bandwidth Assignment Algorithm (Figure 5) for sensor information transmission and compression qualities balancing in sensor networks.

Our algorithm works as follows: After setting up of a cluster (with k sensors), the cluster head collects the compression quality of each sensor Q_1, Q_2 ... Q_k, and the bandwidth assigned to them B_1, B_2 ... B_k in last transmission between these sensors and the cluster head.

Then the average compression quality for all these sensors are computed $Q_A = \sum_{i=1...k} Q_i/k$. For data transmitted later, the cluster head assigns bandwidth to sensor i as $B_i - \alpha(Q_i - Q_A)$, where α is the bandwidth adjusting parameter.

With the dynamic bandwidth assignment to different sensors, those sensors with low approximation quality can get more bandwidth and those with high quality will get less

bandwidth. Therefore, the compression qualities for all the sensor nodes are balanced well.

7 EXPERIMENTAL EVALUATION

7.1 Description of the dataset

We provide a thorough experimental evaluation of our ALVQ algorithm, multi-dimensional ALVQ algorithm and DBA technique on synthetic data set and real world data set. More specifically, we use the following datasets:

Washington State Climate: This is a real dataset of atmospheric data collected at 32 different sites by the Department of Atmospheric Sciences at the University of Washington. The readings, which are recorded at weather logging sites in Washington, include air temperature, barometric pressure, wind speed, relative humidity, cumulative rain and others. More specifically, each of the 32 sites maintains the temperature readings between June 2003 and June 2004. We connect these sites by a network (like in Figure 3) according their positions, and use the DBA technique to minimize the compression error given a bandwidth constraint. We use the **Washington State Climate** data set to evaluate our ALVQ and DBA algorithm.

INFATI: This is a real dataset derived from the INFATI Project [8] carried out by Aalborg University. The readings are the GPS positions of 24 different cars moving in the city of Aalborg, Denmark in 2001. The readings include carid, timestamp, x-coordinate, y-coordinate, etc. Our dataset includes approximately 250k readings recorded between January and March 2001. The **INFATI** dataset is used to evaluate our multi-dimensional ALVQ compression algorithm.

Synthetic: There are totally 300k synthetic data records generated for 8 sensors. To simulate the real sensor measurements at different spots, we generate data records with different variation for different sensors. In addition, some sharp changes are embedded to simulate the exception or abnormal events in the environment. In our experiments, the **Synthetic** data set is used to compare the performance of our DBA algorithm with LEACH model.

7.2 One dimensional data compression within a sensor

We compare the performance of ALVQ and SBR [6] on the **Washington State Climate** data set using the same network constraints and compression parameters. The main objective of our experimental work is to quantify the advantage of ALVQ, namely, quantify the advantage of employing LVQ learning process in the construction of the codebook, and the transmission of com-pressed updates. We only compare with SBR because (1) The two techniques (SBR and ALVQ) are using the same framework and (2) the experimental results of [6] show that SBR outperforms other techniques in the sensor networks' setting.

7.2.1 Varying the Compression Rate

With 3.6k data items in the buffer and codebook 2KB, we vary the compression rate from 4% to 20%. Figure 6 shows that the compression precision decreases gradually as the compression rate increases for both SBR and ALVQ. In addition, ALVQ improves the overall precision over SBR by an average of 15% for all the compression rates, which is due to the higher accuracy of the codebook in ALVQ.

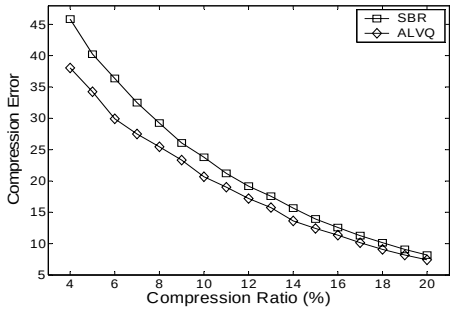

Figure 6 *Varying Compression Rate*

7.2.2 Varying the transmission data size

As time goes on, more data is received and transmitted to the base station (sink). We fix the SBR compression rate as 5%, codebook size as 2KB and try to find the minimum size of transmitted data by ALVQ to achieve the same compression error. Figure 7 shows that with the same error constraint, ALVQ transfers less data than SBR, which means ALVQ achieves a higher compression rate and uses less network communications than SBR.

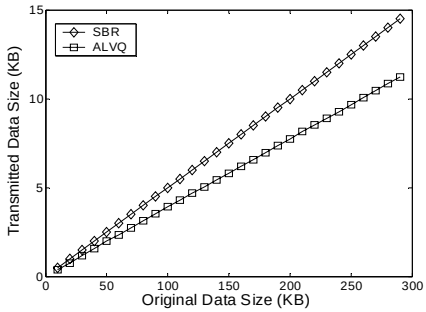

Figure 7 *Transmitted Data Size*

7.2.3 Varying the codebook size

We vary the codebook size from 0.1KB to 2 KB while fixing the compression rate to 5% and data file size to 10KB. Figure 8 shows that the higher the codebook size, the more precise compression is achieved for both SBR and ALVQ. This is because data can be approximated more accurately if we have more patterns in the codebook. On

average, our ALVQ algorithm improved the compression precision of SBR by 15% for different codebook sizes.

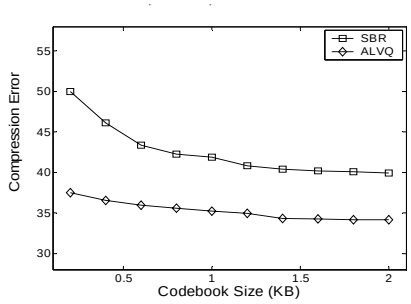

Figure 8 *Varying Codebook Size*

7.3 Multi-dimensional data compression within a sensor

As mentioned in Section 5, there are two approaches for multi-dimensional information compression – **Record Scan** and **Dimension Scan**. These methods transform multi-dimensional information into one-dimensional data array and apply ALVQ algorithm to compress the data array. In this subsection, we compare the compression performances of **Record Scan** and **Dimension Scan** on multi-dimensional sensor information. We use **INFATI** [8] as the real world data set, which are the GPS positions of cars moving in the city of Aalborg, Denmark in 2001.

7.3.1 Varying the Compression Rate

With 10k data items in the buffer and codebook 2KB, we vary the compression rate from 4% to 20%. Figure 9 shows that the compression precision decreases gradually as the compression rate increases for both **Record Scan** and **Dimension Scan**. Compared to **Record Scan**, **Dimension Scan** can compress multi-dimensional information with much less compression errors for all the compression rates, which is analyzed and explained in Section 5.

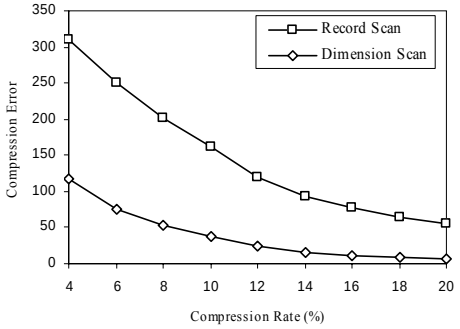

Figure 9 *Varying Compression Rate for Multi-dimensional ALVQ*

7.3.2 Varying the codebook size

We vary the codebook size from 0.4KB to 2 KB while fixing the compression rate to 10% and data file size to 10KB. Figure 10 shows that the higher the codebook size, the more precise compression is achieved for both **Record Scan** and **Dimension Scan**. This is because data can be approximated more accurately if we have more patterns in the codebook. In addition, we found that the codebook size shows more influence on **Recode Scan** than **Dimension Scan**. This is due to the fact that Record Scan cannot effectively capture the frequent patterns in different dimensions, which makes its compression accuracy more dependent on codebook size.

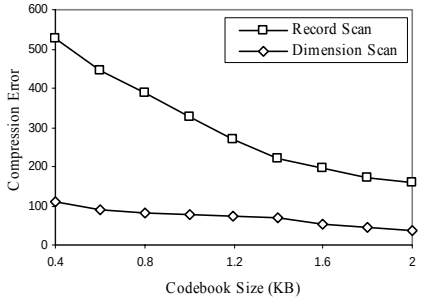

Figure 10 *Varying Codebook Size for Multi-dimensional ALVQ*

7.3.3 Varying the Correlations of Different Dimensional Information

With codebook size 2KB and fixed compression rate of 10%, we evaluate the compression performances of **Record Scan** and **Dimension Scan** with varied correlations of different dimensional information. Initially we set the information at *X* and *Y* dimension with the same values (this corresponds to the strongest correlations of information at dimension *X* and *Y*). Then we vary the correlation levels of data in dimension *X* and *Y* (with correlation level 0 as the most non-correlated and 1 as most correlated) by adding a noise vector to the information at dimension *Y*, Data(*Y*) = Data(*Y*) + *Noise*. With different magnitude of *Noise* we can vary the correlation levels of the data at dimension *X* and *Y*. Figure 11 shows that the compression quality of Dimension Scan is always better than that of Record Scan with different correlation levels. It can also be concluded that, as the information at different dimensions become less correlated (i.e. the correlation level becomes smaller), the advantage of Dimension Scan over Record Scan is more obvious.

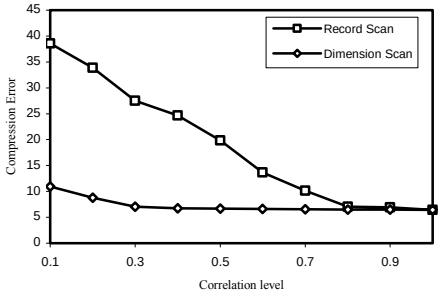

Figure 11 *Varying Correlations of Dimensional Information*

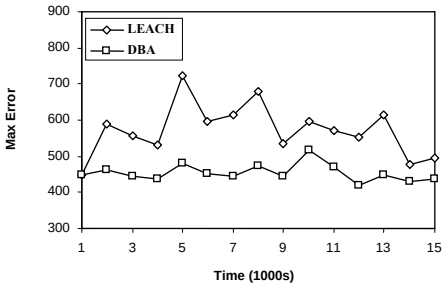

Figure 12 *Maximum Errors on Synthetic Dataset*

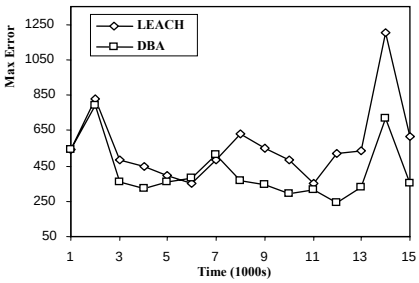

Figure 14 *Maximum Errors on Real World Dataset*

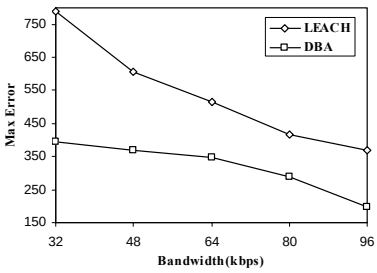

Figure 16 *Maximum Errors with Varied Bandwidth*

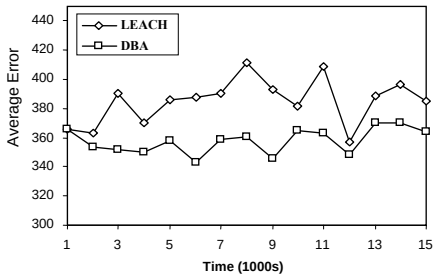

Figure 13 *Average Errors on Synthetic Dataset*

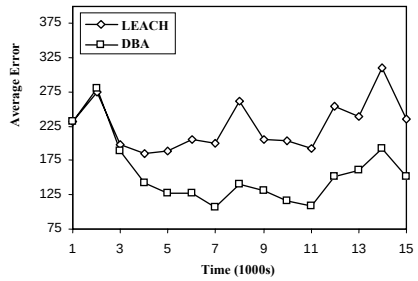

Figure 15 *Average Errors on Real World Dataset*

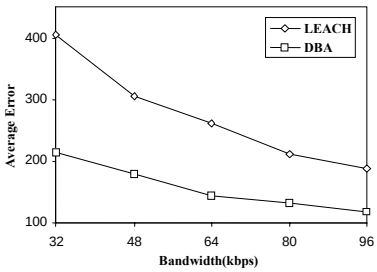

Figure 17 *Average Errors with Varied Bandwidth*

7.4 Dynamic Bandwidth Assignment in the Sensor Network

We compare the performances of DBA and LEACH under the same network settings. The main objective of our work is to quantify the advantage of DBA algorithm over LEACH, namely, quantifying how well DBA algorithm can maximize and balance the compression qualities among different sensors.

7.4.1 Data compression in the sensor network

We evaluate the performances of DBA and LEACH on both **Synthetic** and **Washington State Climate** data sets. For synthetic data set, it is assumed that the data record is generated every second and the sensors transfer their compressed information every 1000 seconds. The bandwidth between sensors is set as 64 Kbps which is a typical sensor node setting. For real world data set (**Washington State Climate**), each measuring site is taken as a sensor and the 32 sites are assigned to 4 clusters (with 8 sensors each) according to their positions. We try to compare the maximum and average compression errors within a cluster under DBA and LEACH model. As is shown in Figure 12, 13, 14 and 15, the DBA algorithm is always consistent with small error as time goes on, while the LEACH approach which just divides the bandwidth evenly to all the sensors is sometimes good (if all the sensors need the same bandwidth), but sometimes very bad. This is because LEACH can not adjust bandwidth to achieve global optimization as in DBA for biased data distribution among sensors.

7.4.2 Varying the bandwidth of channel

With the real world **Washington State Climate** dataset having 32 sensors, we vary the communication bandwidth between sensors from 32 kbps to 96 kbps. Figure 16 and 17 show that DBA outperforms LEACH in maximum and average compression errors under all circumstances. As the bandwidth become smaller, the advantage of DBA over LEACH is more obvious although the compression errors for both model increase.

8 CONCLUSIONS

We present a new data compression technique, designed for historical information compression in sensor networks. Our method splits the collected data into variable length and encodes each of them by an artificially constructed codebook. The values of the codebook are extracted from the training dataset and maintained dynamically as data changes. The LVQ learning process is employed to polish the codebook in the codebook construction step and codebook's updates are compressed to save bandwidth for sensor data transmission. In addition, we propose Dimension Scan approach that extends our ALVQ algorithm to compress multi-dimensional information. Finally we propose a DBA algorithm to dynamically adjust communication bandwidth in order to maximize and balance compression qualities of different sensors.

In our experiments on synthetic datasets and real datasets, we show that both our ALVQ technique and DBA algorithm improve the overall precisions of the approximation in the sensor networks.

ACKNOWLEDGEMENT

This work was supported by NSF grant 0330481.

REFERENCES

[1] J. Chen, D.J. Dewitt, F. Tian, and Y. Wang. "NiagaraCQ: A Scalable Continuous Query System for Internet Databases." *Proceedings of ACM SIGMOD Conference*, Dallas, TX, 2000.

[2] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. "Multi-Dimensional Regression Analysis of Time-Series Data Streams." *Proceedings of VLDB*, Hongkong, China, 2002.

[3] R. Cheng, D. V. Kalashnikov, and S. Prab-hakar. "Evaluating Probabilistic Queries over Imprecise Data." *Proceedings of ACM SIGMOD Conference*, San Diego, CA, 2003.

[4] V. Cherkassky and F. Mulier. "Learning from data: Concepts, Theory and Methods." *John Wiley & Sons*, 1998.

[5] C. Liu, K. Wu, J. Pei, "A Dynamic Clustering and Scheduling Approach to Energy Saving in Data Collection from Wireless Sensor Networks." *Proceedings of SECON*, Santa Clara, CA, 2005

[6] A. Deligiannakis, Y. Kotidis and N. Rouss-opoulos. "Compressing Historical Information in Sensor Networks." *Proceedings of ACM SIGMOD*, Paris, France, 2004.

[7] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless sensor networks." *Proceeding of the Hawaii International Conference System Sciences*, Hawaii, January 2000.

[8] C. S. Jensen, H. Lahrmann, S. Pakalnis, and J. Runge, "*The INFATI Data*", In TIME-CENTER Technical Report, 2004.

[9] T. Kohonen. "Self-Organizing Maps." *Springer-Verlag*, 1995

[10] T. Kohonen, "Improved versions of learning vector quantization." *Proceedings of the International Joint Conference on Neural Networks*, San Diego, CA, 1990.

[11] A. Manjeshwar and D. P. Agrawal, "TEEN : A Protocol for Enhanced Efficiency in Wireless Sensor Networks." *Proceedings of the 1st International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*, San Francisco, CA, 2001.

[12] A. Manjeshwar and D. P. Agrawal, "APTEEN: A Hybrid Protocol for Efficient Routing and Comprehensive Information Retrieval in Wireless Sensor Networks." *Proceedings of the 2nd International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile computing*, Ft. Lauderdale, FL, 2002.

[13] M. Ruhl, H. Hartenstein, "Optimal Fractal Coding is NP-Hard." *Data Compression Conference*, Snowbird, Utah, 1997.

[14] M. Younis, M. Youssef and K. Arisha, "Energy-Aware Routing in Cluster-Based Sensor Networks." *MASCOTS*, TX, October 2002.

[15] F. Ye, G. Zhong, J. Cheng, S. Lu and L. Zhang, "PEAS: A Robust Energy Conserving Protocol for Long-lived Sensor Networks." *Proceedings of the 10th IEEE International Conference on Network Protocols*, Paris, France, November 2002.