The VLDB Journal (2007) 16: 439–461
DOI 10.1007/s00778-005-0173-5

REGULAR PAPER

Antonios Deligiannakis · Yannis Kotidis · Nick Roussopoulos

Dissemination of compressed historical information in sensor networks

Received: 1 December 2004 / Accepted: 2 August 2005 / Published online: 28 July 2006

Abstract Sensor nodes are small devices that "measure" their environment and communicate feeds of low-level data values to a base station for further processing and archiving. Dissemination of these multi-valued feeds is challenging because of the limited resources (processing, bandwidth, energy) available in the nodes of the network. In this paper, we first describe the SBR algorithm for compressing multi-valued feeds containing historical data from each sensor. The key to our technique is the *base signal*, a series of values extracted from the real measurements that is used to provide piece-wise approximation of the measurements. While our basic technique exploits correlations among measurements taken on a single node, we further show how it can be adapted to exploit correlations among multiple nodes in a localized setting. Sensor nodes may form clusters and, within a cluster, a group leader identifies and coalesces similar measurements taken by different nodes. This *localized* mode of operation further improves the accuracy of the approximation, typically by a factor from 5 to 15. We provide detailed experiments of our algorithms and make direct comparisons against standard approximation techniques like Wavelets, Histograms and the Discrete Cosine Transform, on a variety of error metrics and for real data sets from different domains.

Keywords Sensor Networks · Compression

1 Introduction

Recent advances in wireless technologies and microelectronics have made feasible, from both a technological as well as an economical point of view, the deployment of densely distributed sensor networks [3, 56]. These networks are used in a variety of monitoring applications such as military surveillance, habitat monitoring, location tracking and inventory management. Each sensor node is typically equipped with several sensing elements, such as microphones, accelerometers and temperature sensors that allow it to gather low-level measurements of its surroundings. Once enough data is collected, it is processed locally and transmitted to a *base station* for further analysis. A base station may represent any node of the network with increased storage, battery and processing capabilities.

Large-scale sensor networks require tight data handling and data dissemination techniques. Transmitting a *full-resolution* data feed from each sensor back to the base station is often prohibitive due to (i) limited bandwidth that may not be sufficient to sustain a continuous feed from all sensors and (ii) increased power consumption due to the wireless multi-hop communication. In order to minimize the volume of the transmitted data, we can apply two well known ideas: *aggregation* and *approximation*. Aggregation works by summarizing the measurements in the form of simple statistics like average, maximum, minimum, etc. that are then transmitted to the base station over regular intervals. Aggregation is an effective means to reduce the volume of data, but is rather crude for applications that need detailed historical data, like military surveillance. Furthermore, when data feeds exhibit a large degree of redundancy, approximation is a less intrusive form of data reduction in which the underlying data feed is replaced by an approximate signal tailored to the application needs. The trade-off is then between the size of the approximate signal and its precision compared to the real-time information monitored by the sensors.

In this paper we study the problem of disseminating detailed historical information in sensor networks. This is a departure from previous work that only focused on maintaining simple aggregate statistics of the observed data. There are several applications that can benefit from our techniques. A prominent example involves habitat and environmental monitoring applications [2, 39]. Deploying dense networks consisting of hundreds or thousands of inexpensive sensors

A. Deligiannakis (✉) · N. Roussopoulos
University of Maryland
E-mail: {adeli,nick}@cs.umd.edu

Y. Kotidis
AT&T Labs-Research
E-mail: kotidis@research.att.com

will allow us to gather detailed, localized measurements that would have been hard, or even impossible to obtain otherwise [3, 56]. Long-term historical records are invaluable for detecting changing behavioral patterns or building models on the observed eco-system. As is noted in [39], sensor network deployment represent a substantially more economical and less intrusive method for conducting long-term studies than traditional personnel-rich methods.

Our techniques can also help in obtaining historical data for training purposes in model-driven data acquisition [18, 34]. Data models trained by the sensor readings can help reduce the cost of data acquisition in large scale networks during ad-hoc queries, identify sensors that are providing faulty data, and can extrapolate the values of sensor readings at locations where sensors are no longer operational. A hidden cost in building models from sensor readings is that of collecting all necessary data for training purposes. In [18] the authors used detailed sensor readings from six days to train their models. Extracting so much data will be overwhelming for large-scale networks. Our techniques fit nicely in this space by allowing us to obtain and disseminate training data in a more energy-efficient manner.[1] Our data reduction framework can also leverage the computation of temporal and top-k queries discussed in [60].

The techniques presented in this paper build on the Self-Based Regression (SBR) data reduction algorithm introduced in [14]. SBR identifies piece-wise linear correlations among measurements generated at a single sensor node and builds a data dictionary, called the *base signal*, which it then uses in order to compress the collected data. During data transmission, the collected data is carefully partitioned into intervals of varying length that can be efficiently approximated as linear projections of some part of the *base signal*.

While SBR exploits correlations among measurements taken on a single node, in this paper we show how it can also be adapted to exploit correlations among multiple nodes in a localized setting. Our solution is based on an adaptation of the HEED protocol [58] that is used to cluster the sensor nodes into groups and elect within each group a group leader that instruments the execution of each SBR instance in the nodes of its group and also handles the final transmission of the compressed data to the base station. This form of localized processing allows the nodes to exploit spatial correlations and results in a reduction of the error of the approximation by a factor of up to 15, in our experiments, compared to the case when nodes individually compress and transmit their data. We further discuss modifications of the SBR algorithm that allow us to minimize different error metrics than the commonly used sum squared error.

The localized process is illustrated in Fig. 1. Under the localized mode of operation the sensor nodes transmit their compressed data feeds to the group leader, which in turn applies the SBR algorithm to the incoming feeds (including its own measurements). The group leader further coordinates the allocation of bandwidth among the members of its group in order to minimize the desirable error metric. This localized framework allows the group leader to explore correlations between measurements of different sensors and achieve a significantly more accurate approximation for the same overall *compression ratio*. Such correlations may be due to either the small variations of the monitored physical quantities because of the proximity of the sensors (for instance, when obtaining temperature readings), or the nature of the measurements (i.e., voice levels tend to fade gradually with the distance from the source). We also propose an alternative framework for the operation of sensors within localized groups, in which sensors with limited capabilities can compress their data using the base signal constructed by the group leader.

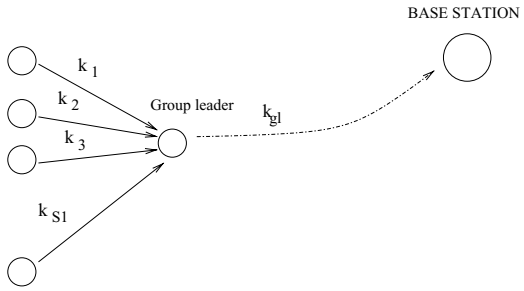

Fig. 1 Dissemination using localized groups: nodes form groups and elect a group leader. Using a localized algorithm each node in the group obtains a compression ration k_i so that the target overall compression ratio is achieved, while the error of the approximation is minimized

While our techniques are motivated by applications of sensor networks, the same algorithms can also be used in other domains for the lossy compression of multiple time series. In addition to weather data that is common in sensor network applications, we also provide experimental evidence using phone-call and stock data sets. In these domains where significantly more powerful processors can be used than in sensor network applications, some of the optimizations proposed in this paper may be relaxed in order to achieve even better approximation.

The rest of the paper is organized as follows. Section 2 presents related work. In Sect. 3 we state our problem and sketch the basics of our techniques, while in Sect. 4 we describe our framework and the SBR algorithm in detail. In Sect. 5 we describe the localized mode of operation. Section 6 contains our experiments. Section 7 presents concluding remarks and future directions.

2 Related work

In recent years there has been a flurry of research in the area of sensor networks. Some of the most important issues addressed include network self-configuration [6, 34], discovery [19, 26] and computation of energy-efficient data

[1] Our lossy compression scheme can also be used to filter out outliers or abnormal readings during the transmission process, by utilizing variants of a regression subroutine model where only a fraction (i.e., 95%) of the data points is used to perform the approximation. However, the details of this process are beyond the scope of this paper.

routing paths [8, 27, 36, 37, 50, 52]. In the database community there is ongoing effort in understanding how embedded database systems can be used to provide energy-based query optimization [38, 57]. In-network data aggregation is investigated in [15, 17, 20, 29, 37, 48, 57]. The main idea is to build an aggregation tree, which partial results will follow. Non-leaf nodes of the tree aggregate the values of their children before transmitting the aggregate result to their parents, thus substantially reducing the number of messages in the network. A robust in-network aggregation framework that handles packet loss and node failures has been presented in [13]. Decentralized algorithms for aggregate computation have also been proposed [4, 31]. Recent proposals for combining data modeling with data acquisition can also help in reducing the cost of aggregation [18, 34].

Sensor nodes are small devices that "measure" their environment and communicate streams of low-level values to a base station for further processing and archiving. These streams are then used to construct a higher-level model of the environment. This process makes historical data equally important to current values [2, 12, 60]. In this paper we propose approximation as a less intrusive form of data reduction that is more suited for applications in which a long-term historical record of measurements from each sensor is required. Unlike aggregate queries that continuously transmit a few values to the base station, in our framework the sensor nodes wait till enough data has been gathered and only then is the data compressed and transmitted over the network. This allows the nodes to power-down their radios for long intervals, thus substantially reducing the energy drain of their batteries.

Recently, there has been increasing interest in understanding the principles of continuous queries in data streams [9, 28, 41, 54, 59]. The work of [43] studies the trade-off between precision and performance when querying replicated, cached data. In [42] the users register continuous queries with strict precision constraints at a central *stream processor*, which, in turn installs filters at the remote data sources. These filters adapt to changes in the streams to minimize update rates. The work in [15] investigates the optimal assignment of error filters in order to minimize the bandwidth consumption of continuous aggregate queries in sensor networks. An online algorithm for minimizing the update cost while the query can be answered within an error bound is presented in [32]. The authors of [11] study a probabilistic query evaluation method that places appropriate confidence in the query answer to quantify the *uncertainty* of the recorded data values.

Approximate processing techniques have been widely studied. The AQUA project explored sampling-based techniques for building *synopses* and using them to provide approximate answers at a fraction of the time that a real answer would require [23, 24]. Histograms have been extensively used by query optimizers to estimate the selectivity of queries, and recently in tools for providing fast approximate answers to queries [30, 44, 45, 47, 53]. Wavelets are a mathematical tool for the hierarchical decomposition of functions, with applications in image and signal processing [51]. More recently, Wavelets have been applied successfully in answering range-sum aggregate queries over data cubes [55], in selectivity estimation [40] and in approximate query processing [7, 16, 22, 25]. The Discrete Cosine Transform (DCT) [1] constitutes the basis of the *mpeg* encoding algorithm and has also been used to construct compressed multidimensional histograms [35]. Linear regression has been recently used in [10] for on-line multidimensional analysis of data streams.

In [14] we introduced the SBR algorithm, a method for compressing measurements generated at a single sensor node through a data dictionary, called the *base signal*. In this paper we extend the SBR algorithm to also operate on a localized organization of sensor nodes. The use of a dictionary is a standard technique in data compression algorithms (for example in gzip). As an abstraction, the base signal of the SBR algorithm is a dynamic data dictionary that is used to compress present and future values. A fundamental difference with compression techniques such as gzip is that our compression is lossy, which allows us to achieve significantly higher compression ratios. Furthermore, the details of our approximation (construction of the base signal, approximation using regression etc.) differ at a fundamental level from standard compression techniques. Many popular transforms also use some form of basis that is either fixed (i.e., Wavelets, DCT) or data dependent (i.e., SVD). We also explore the use of such bases in our framework and present the necessary modifications. However, the base signal constructed by our algorithms seems to always outperform these bases, for the specific encoding we use in our approximation.

3 Preliminaries

In this section we first present a description of the characteristics of sensor networks and their applications. We then describe the operation of sensor nodes in our data reduction framework and present the optimization problems that we address in this paper.

3.1 Characteristics of sensor networks

Recent technological advances have made possible the development of low-cost sensor nodes with heavily integrated sensing, processing and communication capabilities. Information about the environment is gathered using a series of sensing elements connected to an analog-to-digital converter. Examples include microphones for acoustic sensing, accelerometers and temperature sensors. Once enough data is collected, it is processed locally and periodically forwarded to a base station, using a multi-hop routing protocol [49].

The processing subsystem on the nodes depends on the nature of the application. Applications such as military

reconnaissance that require significant processing to be performed at the nodes use sensor nodes with significant processing power. As an example, an improved model of the commonly used StrongARM 1100 processor (μAMPS [49] and HiDRA nodes) reaches a frequency of 400 MHz and can support up to 64 MB of memory.

As the processing and storage capabilities of sensor nodes tend to follow Moore's Law, their communication and power subsystems become the major bottleneck of their design. For example, over the last years, the energy capacity of the batteries used in such nodes has exhibited a mere 2–3% annual growth.[2] The main source of energy consumption in a node is the data transmission process. There are several reasons for this:

1. The energy drain during transmission is much larger than the consumption during processing [19]. As an example, on a Berkeley MICA Mote sending one bit of data costs as much energy as 1,000 CPU instructions [38].
2. Transmission ranges between nodes are fairly short. The transmitted data may thus require to traverse multiple hops to reach the base station. This retransmission process at each intermediate node is very costly. Furthermore, because nodes often use broadcast protocols over radio frequencies [37], transmitted messages are not only received by the intended node, but by all nodes in the vicinity of the sender, thus increasing the overall power consumption.

Even on applications where battery lifetime is not a concern (i.e., military surveillance sensing nodes attached to moving vehicles with practically infinite power supply), the available bandwidth may not sustain a continuous feed of measurements for all sensors deployed in the terrain. The design of data reduction protocols that effectively reduce the amount of data transmitted in the network is thus essential when the goal is to meet the application's bandwidth constraints or to increase the network's lifetime.

3.2 Data model and processing

In order not to deplete their power supply and to conserve bandwidth, the sensors do not continuously transmit every new measurement they obtain but rather wait till enough data is collected and then forward it to the base station [49]. This form of batch processing allows them to power-down their radio transmitter and prolong their lifetime in a way analogous to [37].

Within a sensor, the recorded data is depicted in a two dimensional array A where each row i stores sampled values of a distinct quantity. Informally, each row i is a time series $\vec{Y}_i$ of samples from quantity i collected by the sensor. The array has N rows, N being the number of recorded quantities and M columns, where M depends on the available memory.[3]

As more measurements are obtained, the sensor's memory buffers become full. At this point the latest $N \times M$ values are processed and each row i (of length M) is approximated by a much smaller set of B_i values (i.e., $B_i \ll M$). The resulting "compressed" representation, of total size equal to $B = \sum_{i=1}^{N} B_i$, is then transmitted to the base station. The base station maintains the data in this compact representation by appending the latest "chunk" to a log file. A separate file exists for each sensor that is in contact with the base station. The entire process is illustrated in Fig. 2.

Each sensor allocates a small amount of memory of size M_{base} for what we call the *base signal*. This is a compact ordered collection of values that we extract from the recorded values and are used as a base reference in the approximate representation that is transmitted to the base station (details will be given in the next section). The data values that the sensor transmits to the base station are encoded using the in-memory values of the base signal at the time of the transmission. The base signal may be updated at each transmission to ensure that it will be able to capture newly observed data features (where the term "feature" is used in this paper to denote data intervals which contain data value flunctuations that are frequently encountered in the collected data), and that the obtained approximation will be of good quality. When such updates occur, they are transmitted along with the data values and appended in a special log file that is unique for each sensor. This allows the base station to reconstruct (approximately) the series $\vec{Y}_i$ at any given point in the past.

3.3 Our optimization problem

We can think of the base signal as a dictionary of features used to describe the data values. The richer the pool of features we store in the base signal the better the approximation. On the other hand, these features have to be (i) kept in the memory of the sensor to be used as a reference by the data reduction algorithm and (ii) sent to the base station in order for it to be able to reconstruct the values. Thus, for a target bandwidth constraint (number of values that can be transmitted) the more insert and update operations on the base signal that we perform, the less bandwidth is left available for approximating the data values. Moreover, the time to perform the data approximation increases, in our algorithms, linearly with the size of the base signal.

In the next section we present an efficient algorithm that decides (i) how large the base signal needs to be at each transmission, (ii) what new features to be included in it, (iii) which older features are not relevant any more, and (iv) how to best approximate the data measurements using these features. The only user input needed by the algorithm is the

[2] http://nesl.ee.ucla.edu/courses/ee202a/2002f/lectures/L07.ppt

[3] We here assume that all quantities are sampled with the same frequency. This simplifies notation, however, our framework also applies when each quantity is recorded on a different schedule.

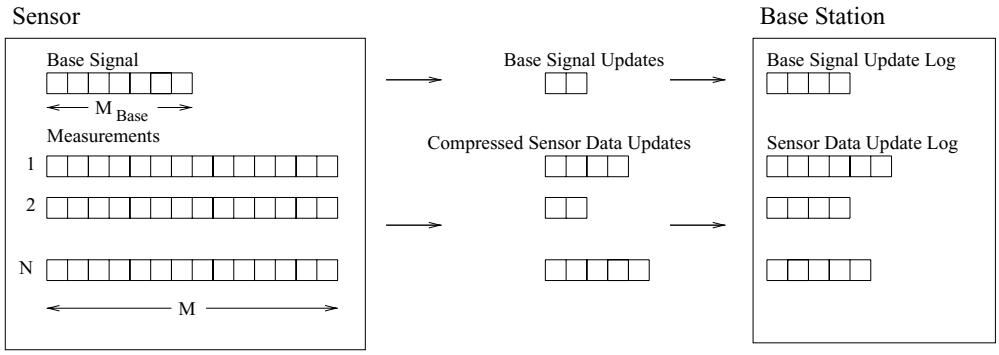

Fig. 2 Transfer of approximate data values and of the base signal from each sensor to the base station

target bandwidth constraint and the maximum buffer size of the base signal values.

4 The SBR framework

We now describe our framework in more detail. We start with a motivational example that demonstrates the intuition behind our techniques. Section 4.2 presents the primitive operations required by our framework while the *SBR* algorithm is presented in Sect. 4.3. Table 1 contains a brief description of the parameters used in our algorithms.

4.1 Motivational example

Many real signals are correlated. We expect this to be particularly true for measurements taken by a sensor, especially if they are physical quantities like temperature, dew-point, pressure etc. The same is often true in other domains. For example, in Fig. 3 we plot the average Industrial and Insurance indexes from the New York stock market for 128 consecutive days.[4] Both signals show similar trends (i.e., they go up and down together). Figure 4 depicts a XY scatter plot

[4] Data at http://www.marketdata.nasdaq.com/mr4b.html

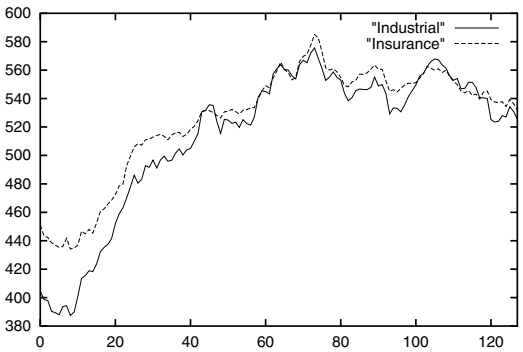

Fig. 3 Example of two correlated signals (*Stock Market*)

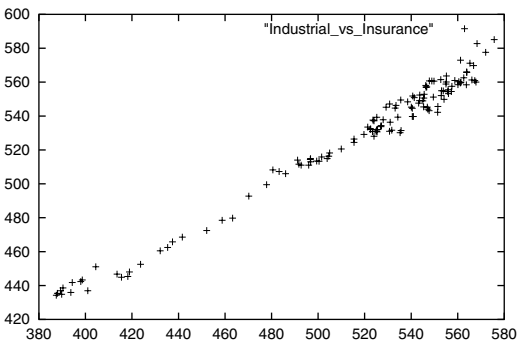

Fig. 4 XY scatter plot of Industrial (*X axis*) vs Insurance (*Y axis*)

Table 1 Configuration, input and derived parameters of our algorithms

Configuration Parameters	
N	Number of input signals
M	Measurements per input signal
A	The $N \times M$ array of measurements
Input Parameters	
$TotalBand$	Total bandwidth per transmission
M_{base}	Buffer size for base signal values
Derived/Calculated Parameters	
$n = N \times M$	Size of in-memory data
$W = \sqrt{n}$	Size of each base interval
B	Compressed Data Size
$maxIns$	Maximum number of base intervals inserted during the current transmission
Ins	Number of base intervals actually inserted during the current transmission

of the same values. This is created by pairing values of the Industrial (X-coordinate) and Insurance (Y-coordinate) indexes of the same day and plotting these points in a two-dimensional plane. The strong correlation among these values makes most points lie on a straight line. This observation motivates our work. Assuming that the Industrial index (call it $\vec{X}$) is given to us in a time-series of 128 values, we can approximate the other time-series (Insurance: $\vec{Y}$) as:

$$\vec{Y}' = a * \vec{X} + b$$

The coefficients a and b are determined by the condition that the sum of the square residuals, or equivalently the L_2 error norm $||\vec{Y}' - \vec{Y}||_2$, is minimized. This is nothing more than

standard linear regression. However, unlike previous methods, we will not attempt to approximate each time-series independently using regression. In Fig. 3 we see that the series themselves are not linear (i.e., they would be poorly approximated with a linear model). Instead, we will use regression to approximate piece-wise correlations of each series to a base signal that we will choose accordingly. In the example of Fig. 4 the base signal can be the Industrial index ($\vec{X}$) and the approximation of the Insurance index will be just two values (a, b). In practice the base signal may be much smaller than the complete time series, since it only needs to contain the "important" trends of the target signal $\vec{Y}$. For instance, in case $\vec{Y}$ is periodic, a sample of the period would suffice. Our algorithm breaks the latest measurements obtained by the sensor into small intervals (of varying sizes) and looks for intervals of the same length in the base signal that are linearly correlated. At the same time, the base signal values are evaluated and may get updated with features from the newly collected measurements when necessary.

4.2 Primitives of our implementation

Piece-wise approximation of measurements

We here assume that the base signal $\vec{X}$ is given to us; we later describe how to construct the base signal. We will approximate the latest $N \times M$ measurements in $\vec{Y}_1, \ldots, \vec{Y}_N$ using $B \geq 4 \times N$ values (thus using at least four values per time series). Later in this section we describe how to determine the value of B given the desired size *TotalBand* of the transmitted representation (including any updates to the base signal).

To simplify notation, we model the collected data as a single series $\vec{Y}$ that is simply the concatenation of the N series $\vec{Y}_i$. Our technique relies on breaking $\vec{Y}$ into $B/4$ intervals and "mapping" each one to an interval of the base signal of equal length.[5] The algorithm works recursively. It starts with a single interval for each row of the collected data. In each iteration, the interval with the largest error in the approximation is selected and divided in two halves, until the "budget" of $B/4$ intervals is exhausted. An interval I is a data structure with six entries:

- *start*, *length*: these define the scope of the interval; i.e., I represents values of $Y[i]$, with i in [*start*, *start*+*length*).
- *shift*: it defines the part of the base signal that is used to approximate the values of I; the interval I is mapped to segment [*shift*, *shift* + *length*) in $\vec{X}$.
- a, b, err: the first two are the regression parameters, while err is the sum squared error (*sse*) of the approximation.

Subroutine `Regression()` shown in Algorithm 1 is at the core of our method. This function pairs a segment of the base signal between values [$start\_x$, $start\_x + length$) with values of $\vec{Y}$ between [$start\_y$, $start\_y + length$), as in Fig. 4, and computes the regression parameters a, b as well as the (sse) error of the approximation $\vec{Y}' = a\vec{X} + b$ in this range. Each value $Y[i]$ with index i in [$start\_y$, $start\_y + length$) is approximated as $aX[start\_x + i - start\_y] + b$.

It should be noted that the `Regression()` subroutine calculates the optimal a, b values that minimize the sum squared error of the approximation. If the desired error metric is different, then the formulas need to be appropriately modified. In Sect. 4.6 we present the necessary modifications for two interesting optimization problems: minimizing the sum squared relative error, and minimizing the maximum absolute error of the approximation. The modified algorithms run in $O(length)$ time and require $O(1)$ and $O(length)$ space, respectively.

Subroutine `BestMap()` of Algorithm 2 looks for the best way to approximate an interval I. It shifts I over $\vec{X}$ and calculates the regression parameters and the approximation error for the *shift* parameter that produces the smallest error. This algorithm contains two deviations from our previous discussion. First, it also considers approximating each interval I using standard linear regression, and uses a negative value for the $I.shift$ parameter to denote this. Second, it performs the shifting process over the base signal only for intervals with a maximum length of $2 \times W$, where

Algorithm 1 `Regression` Subroutine

Require: $\vec{X}, \vec{Y}, start\_x, start\_y, length$
1: {Compute Regression Parameters}
2: $sum\_x = \sum_{0 \leq i < length} X[i + start\_x]$
3: $sum\_y = \sum_{0 \leq i < length} Y[i + start\_y]$
4: $sum\_xy = \sum_{0 \leq i < length} X[i + start\_x]Y[i + start\_y]$
5: $sum\_x2 = \sum_{0 \leq i < length} X[i + start\_x]^2$
6: $a = \frac{length \times sum\_x\_y - sum\_x \times sum\_y}{length \times sum\_x2 - sum\_x \times sum\_x}$
7: $b = \frac{sum\_y - a \times sum\_x}{length}$
{Compute sse of approximate signal $\vec{Y}' = a\vec{X} + b$ in range $[start\_y, start\_y + length)$}
8: $err = \sum_{i=0}^{length-1} (Y[i + start\_y] - (aX[i + start\_x] + b))^2$
9: return (a,b,err)

Algorithm 2 `BestMap` Subroutine

Require: $\vec{X}, \vec{Y}$, Interval I, W
1: $I.shift = -1$
2: Perform standard linear regression on I and set the values of $I.a$, $I.b$ and $I.err$
3: **if** $I.length \leq 2 \times W$ **then**
4: {Shift I over $\vec{X}$ and find the segment for which the regression error is minimized}
5: **for** $shift$ in $0..length(\vec{X}) - I.length - 1$ **do**
6: (a,b,err) = Regression($\vec{X}, \vec{Y}, shift, I.start, I.length$)
7: **if** err is minimum error so far **then**
8: Update values of $I.a$, $I.b$, $I.err$ and $I.shift$
9: **end if**
10: **end for**
11: **end if**

[5] This mapping requires four values per interval, thus the division by 4.

W is a parameter that denotes the length of the intervals that constitute the base signal.[6] The last modification is performed both to reduce the time complexity of the algorithm to O($I.length + W \times M_{base}$), and because of the reduced likelihood that large intervals will be accurately mapped to multiple consecutive intervals of the base signal.

The core approximation algorithm `GetIntervals()` is given in Algorithm 3. The approximation obtained is returned as a list of $B/4$ intervals in i_list. This list is maintained sorted (priority queue) based on the sse of each interval. $\vec{X}$ is the current base signal.

Theorem 1 *The* `GetIntervals()` *algorithm runs in* $O(NMlog(\frac{B}{N}) + B \times M_{base} \times W)$ *time.*

Proof The `GetIntervals()` algorithm repeatedly breaks the interval with the largest error into two halves and calls the `BestMap()` algorithm for each interval. Retrieving the interval with the maximum error can be done in $O(1)$ time since the i_list is sorted. Since the running time of the `BestMap()` algorithm increases with the size of the mapped interval, the worst case complexity of the `GetIntervals()` algorithm arises when the algorithm continuously retrieves and breaks the interval with the largest size. Since exactly $B/4 - 1$ breaks will be performed, and N signals of size M exist, the algorithm will perform in the worst case N breaks of intervals with size M, $2 \times N$ intervals of size $\frac{M}{2}, \ldots, 2^{\log \frac{B}{4N}-1} \times N$ intervals of size $\frac{M}{\log \frac{B}{4N}-1}$. Thus, the overall running time is:

$$\begin{aligned}
&\sum_{i=1}^{\log \frac{B}{4N}} \left(N \times 2^{i-1} \times O\left(\frac{M}{2^{i-1}} + W \times M_{base} \right) \right) \\
&= O\left(N \times \sum_{i=1}^{\log \frac{B}{4N}} (M + W \times M_{base} \times 2^{i-1}) \right) \\
&= O\left(MN \log \frac{B}{4N} + N \times W \times M_{base} \times \frac{B}{4N} \right) \\
&= O\left(MN \log \frac{B}{N} + B \times W \times M_{base} \right).
\end{aligned}$$

□

For each interval in i_list a record with four values ($I.start$, $I.shift$, $I.a$, $I.b$) is transmitted to the base station. The base station will sort the intervals based on $I.start$ and, thus, there is no need to transmit their length. It is interesting to note that the `GetIntervals()` algorithm decides dynamically how many intervals it will use to approximate each of the N rows of the collected data, allocating more intervals to signals that are harder to approximate accurately.

Selecting data features for inclusion in the base signal

We focus on the time when the sensor's memory is filled with $N \times M$ values, as depicted in Fig. 2. We assume that the buffer allocated to the base signal is of size M_{base}. This buffer is organized as a list of intervals (called *base intervals*) of the same length W. For simplicity, we assume that both M and M_{base} are multiples of W. We note here that in Algorithm 3 the base signal is presented as a series of M_{base} values, which is simply the concatenation of the base intervals in the buffers.

The `GetBase()` algorithm (Algorithm 4) is called during the initialization and update procedure of the base signal. The algorithm receives as inputs the N signals, each of

[6] This will become more clear later in our discussion.

Algorithm 3 `GetIntervals` Algorithm

Require: $\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, B, W$
i_list = ()
$\vec{Y} = concat(\vec{Y_1}, \ldots, \vec{Y_N})$ {Virtual assignment}
{Create an interval for each row $\vec{Y_i}$ (M values each)}
for i in 1..N **do**
  $(I.start, I.length) = ((i-1) \times M, M)$
  BestMap($\vec{X}, \vec{Y}, I, W$)
  i_list.push(I);
end for
$num\_intervals = N$
while $num\_intervals{+}{+} < B$ / 4 **do**
  {i_list is sorted on decreasing order of $I.err$}
  $I = i\_list$.pop()
  {Break I in 2 pieces}
  $(I_{left}.start, I_{left}.length) = (I.start, I.length/2)$
  BestMap($\vec{X}, \vec{Y}, I_{left}, W$)
  $(I_{right}.start, I_{right}.length) =$
    $(I.start+I.length/2, I.length/2)$
  BestMap($\vec{X}, \vec{Y}, I_{right}, W$)
  i_list.push(I_{left})
  i_list.push(I_{right})
end while
return i_list

Algorithm 4 `GetBase()` Algorithm

Require: $\vec{Y_1}, \ldots, \vec{Y_N}, W, M, maxIns$
Create $K = \frac{N \times M}{W}$ CBIs of width W
For each CBI $Cand_i$, set its benefit to 0
Maintain unsorted list Q with CBIs
Maintain list base_list with selected stored intervals
$LinearErr(Cand_j)$ is the error of approximating $Cand_j$ using standard linear regression
for i in 1..K **do**
  for j in 1..K **do**
    {Calculate error of approximating the j-th CBI by using as base the i-th CBI}
    $error$=Regression($Cand_i$,$Cand_j$,0,0,W)
    if $error \leq LinearErr(Cand_j)$ **then**
      $Cand_i.benefit$+=$LinearErr(Cand_j)$-$error$
    end if
  end for
  Q.insert($Cand_i$)
end for
for i in 1..$maxIns$ **do**
  $C = Q$.popBestInterval()
  base_list.insert(C)
  for j in 1..$|Q|$ **do**
    adjust($Q[j].benefit, C$)
  end for
end for
return base_list

size M, the size W of each base interval, and the maximum number of intervals *maxIns* that can be inserted in our base signal, where

$$maxIns = \frac{\min\{M_{base}, TotalBand\}}{W}$$

Each input signal $\vec{Y}_i$ is broken into $\frac{M}{W}$ non-overlapping intervals of size W. This provides a "dictionary" of $\frac{N*M}{W}$ *candidate base intervals* (CBIs). The algorithm will choose $maxIns$ CBIs out of this dictionary to be inserted into a *candidate update base signal*. We will describe in Sect. 4.3 how to determine how many of these CBIs will ultimately be inserted into the base signal.

Each CBI $Cand_i$ can be used to approximate any other CBI $Cand_j$, which is in-fact part of some $\vec{Y}_k$, using regression. We consider such an approximation to be beneficial, only if the error of the approximation is smaller than the error of approximating $Cand_j$ using standard linear regression. In Algorithm 4 we denote the latter error as $LinearErr(Cand_j)$. The benefit of using $Cand_i$ to approximate $Cand_j$ is simply the reduction in error that we get compared to $LinearErr(Cand_j)$.

The CBIs are stored in an unordered list Q. At each step of the algorithm, the CBI in Q with the largest benefit is selected for inclusion in the candidate update base signal stored in base_list. After each selection, the benefits of the remaining CBIs in Q have to be properly updated. As we mentioned, the benefit of using $Cand_i$ to approximate $Cand_j$ is originally equal to the reduction in error that we get compared to $LinearErr(Cand_j)$. However, at an intermediate step of the algorithm, some CBIs have already been selected for inclusion in the candidate update base signal. By using these stored CBIs, many of the remaining CBIs can now be better approximated than by using standard linear regression. Thus, the benefit of using $Cand_i$ to approximate $Cand_j$ has to be adjusted to depict the reduction in error that we get when compared to the best approximation for $Cand_j$ that we have so far, by using the current candidate update base signal. Intuitively, this adjustment prohibits the inclusion in the base signal of CBIs that help approximate well similar parts of the data.

An example is presented in Fig. 5. In this small example we consider just 3 CBIs, out of which we need to pick which two to select. In the left part of the figure, we present the benefits of each of the 3 CBIs. Recall that at each step of the algorithm, the benefit of using the i-th CBI for the approximation of the j-th CBI is defined as the reduction in error that we achieve compared to the best approximation that we may achieve for the j-th CBI using either standard linear regression, or a mapping to an already selected CBI. In our example, the first CBI has the largest total benefit, and is thus selected. In the right part of the figure, the adjusted benefits of the remaining CBIs are presented. For example, the benefit of using the second CBI in order to approximate the third CBI is reduced to $0.55 - 0.50 = 0.05$, due to the improved approximation of the third CBI that we can achieve using the recently selected first CBI. Notice that now, the third CBI will be selected, even though initially it had a lower benefit than the second CBI.

CBI	Approximated CBI 1	2	3	Total Benefit
1	1	0.95	0.50	2.45
2	0.8	1	0.55	2.35
3	0.6	0.65	1	2.25

Initial Benefits of CBIs

CBI	Approximated CBI 2	3	Total Benefit
2	0.05	0.05	0.10
3	0	0.5	0.50

Adjusted Benefits of Non-Stored CBIs

Fig. 5 Example of the GetBase() Algorithm

In the `GetBase()` algorithm, for each of the $K = \frac{N \times M}{W}$ CBIs, we first estimate its benefit for approximating all the other CBIs. Each such approximation requires $O(W)$ time, thus resulting in a total complexity of $O(\frac{N^2M^2}{W})$. Then, for each of the *maxIns* selected CBIs, detecting the one with the largest benefit requires $O(K)$ time (we do not sort the CBIs). After each selection, adjusting the benefits of the remaining CBIs requires time $O(K^2)$. Thus, the overall running time complexity of the algorithm is $O(\frac{N^2M^2}{W} + maxIns \times \frac{N^2M^2}{W^2})$, while its space requirements is $O(\frac{N^2M^2}{W^2})$.

For $n = N \times M$ being the size of the data, a value of $W = \sqrt{n}$ used by the SBR algorithm (described in the Sect. 4.3) results in a running time of $O(n^{1.5})$ for GetBase() and space of O(n), since $maxIns \times W \leq TotalBand \leq n$. In case of severe memory constraints, we can easily modify the `GetBase()` algorithm to only store for each CBI the smallest error of approximating it using at each step the current base signal. The only modification will be to replace Lines 19-21 of the `GetBase()` algorithm with a double for-loop similar to the one of Lines 6-15, and alter the calculation of each CBI's benefit to take into account the error of the best approximation that we have for each CBI so far. This modified algorithm requires $O(\sqrt{n})$ space and has a running time of $O(maxIns \times n^{1.5})$.

4.3 The SBR algorithm

We now present the *Self-Based Regression* (SBR) algorithm that performs the approximation of the data values. The algorithm receives as input the latest $n = N \times M$ data values, a bandwidth constraint *TotalBand* (number of values to transmit, *including any base signal values*), the maximum size of the base signal M_{base} and the current base signal $\vec{X}$ of size $|\vec{X}| \leq M_{base}$.[7] From these parameters the user/application has to provide only *TotalBand* and M_{base}. The SBR algorithm must then make the following decisions:

1. Decide how many, and which base intervals to insert into the base signal. Recall that any such base interval has to be transmitted to the base station.
2. If the above procedure causes the size of the base signal to exceed M_{base}, then some base intervals need to be

[7] At the first transmission the current base signal will be empty.

Algorithm 5 `SBR` Algorithm

Require: $\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, M, TotalBand, M_{base}$
1: $maxIns = \frac{\min\{M_{base}, TotalBand\}}{W}$
2: $W = \sqrt{N \times M}$
3: $base\_list = GetBase(\vec{Y_1}, \ldots, \vec{Y_N}, W, M, maxIns)$
4: {$Errors[i]$ is the approximation error after inserting the first i CBIs of base_list in the base signal}
5: Initialize $Errors[i]$ = UNDEFINED $\forall i \in [0..maxIns)$
6: Ins = Search($\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, W, M, TotalBand, base\_list, Errors, 0, maxIns$)
7: Form $\vec{X}_{new}$ by appending the Ins first intervals of the base_list to $\vec{X}$
8: $B = TotalBand - Ins \times (W + 1)$
9: $GetIntervals(\vec{X}_{new}, \vec{Y_1}, \ldots, \vec{Y_N}, B, W)$
10: **if** $|\vec{X}_{new}| > M_{base}$ **then**
11: Evict $Repl = \frac{|\vec{X}_{new}| - M_{base}}{W}$ intervals of $\vec{X}_{new}$ that also belonged to $\vec{X}$ using a LFU replacement policy
12: Replace evicted intervals with the last $Repl$ intervals of $\vec{X}_{new}$
13: **end if**
14: $\vec{X} = \vec{X}_{new}$
15: Transmit the inserted base intervals, their offsets in the base signal and the regression intervals

Algorithm 6 `CalculateError` SubRoutine

Require: $\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, B, W, Errors, pos$
1: **if** Errors[pos] == UNDEFINED **then**
2: list' = $GetIntervals(\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, B - pos \times W, W)$
3: Errors[pos] = sum of errors in list'
4: **end if**

Algorithm 7 `Search` SubRoutine

Require: $\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, W, B, base\_list, Errors, start, end$
1: **if** end == start **then**
2: return start
3: **end if**
4: middle = (start + end) / 2
5: $CalculateError(\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, B, W, middle)$
6: $CalculateError(\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, B, W, start)$
7: **if** $Errors[middle] > Errors[start]$ **then**
8: $CalculateError(\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, B, W, end)$
9: **if** $Errors[end] > Errors[start]$ **then**
10: return Search($\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, W, M, B, base\_list, Errors, start, middle$)
11: **else**
12: return Search($\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, W, M, B, base\_list, Errors, middle, end$)
13: **end if**
14: **else**
15: $CalculateError(\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, B, W, middle + 1)$
16: **if** $Errors[middle + 1] < Errors[middle]$ **then**
17: return Search($\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, W, M, B, base\_list, Errors, middle + 1, end$)
18: **else**
19: return Search($\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, W, M, B, base\_list, Errors, start, middle$)
20: **end if**
21: **end if**

evicted from the base signal, in order to keep its maximum size at M_{base}.
3. Decide how to best approximate the data values given the updated base signal.

We here have to emphasize that it is not always desirable to insert a large number of base intervals into the base signal. Since any inserted base interval needs to be communicated to the base station, the larger the number of such intervals, the smaller the number of intervals that can be used to approximate the N signals by the `GetIntervals()` algorithm, since the overall bandwidth consumption is upper-bounded by the *TotalBand* parameter.

The SBR algorithm is presented in Algorithm 5. It initially calls the `GetBase()` subroutine to select a set of *maxIns* CBIs, where $maxIns = \frac{\min\{M_{base}, TotalBand\}}{W}$. It then performs a binary search on this list, to determine the number of CBIs that will ultimately be inserted into the base signal. This search terminates when the algorithm determines a number of intervals *Ins*, such that the error of the approximation when inserting the first *Ins* intervals of the aforementioned list in the base signal is lower than inserting either the first $Ins - 1$ intervals, or the first $Ins + 1$ intervals into the base signal. This is achieved through the call to function `Search()` at Line 6, which is presented in Algorithm 7. The approximation of the N signals is then performed by using the concatenation of the previous base signal with these Ins intervals. After this step, if the size of the base signal now exceeds M_{base}, then enough base intervals of the old base signal are evicted from the base signal using a Least Frequently Used (LFU) replacement policy. Any newly inserted base interval will thus either occupy an empty position of the base signal, or replace another base interval. Each transmission includes exactly *TotalBand* values:

1. The *Ins* newly inserted base intervals, and their position in the base signal in which they were ultimately inserted ($Ins \times (W + 1)$ values in total).
2. $\frac{TotalBand - Ins \times (W+1)}{4}$ intervals of four values each (start, shift plus the two regression parameters).

The running time complexity of the SBR algorithm is $O(n^{1.5} + (n\log(\frac{TotalBand}{N}) + TotalBand \times \sqrt{n} \times M_{base}) \times \log(maxIns))$, where $maxIns = \frac{\min\{M_{base}, TotalBand\}}{\sqrt{n}}$. Thus, the entire algorithm has a modest $O(n^{1.5})$ dependency on the data size, while its running time scales linearly with the size of the transmitted data *TotalBand* and the (maximum) size of the base signal M_{base}.

4.4 Design issues in SBR

Several decisions in the design of the `SBR` algorithm were made in order to limit its running time complexity. We initially decided to simply look for and exploit linear piecewise dependencies between each data interval and some part of the base signal. Obviously, we could alternatively have used a more complex model (i.e., a polynomial function), hoping that the increased expressiveness of the model could lead to a more accurate compressed representation of the

data. However, one of the main advantages of using a linear model is that its parameters can be easily calculated in time linear to the size of the approximated interval. Any model which would require more than linear time for the calculation of its parameters would incur a similar increase in the complexity of the SBR algorithm. Moreover, the use of a linear model is intuitive. If two data series exhibit a similar behavior, then if we appropriately scale and then shift one of them we expect it to match the other one quite well. This is exactly what a linear regression model achieves.

Similarly, in our `GetIntervals()` algorithm (Algorithm 3) the interval with the largest error is broken in half. This step is obviously suboptimal, since breaking the interval into two subparts of uneven length might result in a better approximation. However, such a modification would entail two main disadvantages. Firstly, the running time complexity of finding the best breakpoint for an interval of length $I.length$ will require $O(I.length \times (I.length + W \times M_{base}))$ time. Moreover, the optimal breakpoint may lie near the endpoints of the interval, which implies that the length of one of the newly generated subinterval may be as high as $I.length - 2$. This, in turn, results in higher running times, since larger intervals are broken and mapped into the base signal. Thus, the overall increase in the algorithm's running time complexity would be prohibitive in sensor network applications, even though it might be acceptable in other, less constrained applications. For example, in our Phone data set experiment in Sect. 6.1.1 (Table 4), this modification reduces the error of SBR in the first transmission by 14.7%, while increasing its running time 111 times.

A similar justification also motivated the design of our `GetBase()` algorithm. An alternative choice would have been to consider $(M - W + 1)$ CBIs of length W for each monitored quantity (signal) (instead of just $\frac{M}{W}$ CBIs), where the i-th CBI would cover the values in the range $[i..i + W]$. Using an increased number of CBIs would help in cases when two CBIs corresponding to two different signals are strongly correlated (i.e., follow a similar behavior), but with a small time delay. The selection of the CBIs to include in the candidate update base signal then has to be modified, since two (or more) selected CBIs from the same signal may correspond to ranges of data values that overlap. Thus, the selection for inclusion should be based on the *per storage benefit* of selecting a CBI, where the CBI's storage cost is defined as the number of its data values that are not already included in the candidate update base signal, at the current step of the algorithm. The SBR algorithm then requires the following important changes:

1. The used CBIs in the binary search process may not have the same length (due to the described possible overlap).
2. We need to make sure that overlapping CBIs are properly stored (i.e., based on the index of their covered data values) in the candidate update base signal, independently of the timing of their selection from the `GetBase()` algorithm. This will force consecutive values from CBIs of each signal to also lie consecutively in the candidate update base signal, and increases the chance that larger data intervals will be accurately mapped into the base signal.

However, these changes would result in an increase of the running times requirements of the `GetBase()` (and, thus, the SBR algorithm) to $O(n^{2.5})$, while its space requirements also increase to $O(n^2)$, which would obviously make the algorithm impractical for sensor network applications.

4.5 Understanding the complexity of SBR

We note that the SBR algorithm is only executed periodically, thus, its running-time complexity has to be evaluated with respect to the size of the data and the frequency that the algorithm is being executed in a real application. Using our implementation of the algorithm on a 300MHz processor, it takes about 30 seconds to process $n = 20{,}480$ data values (10 time series of 2048 values each) for a 10% compression ratio (see Sect. 6). Even if one measurement is being taken every second, the above running time corresponds to measurements collected over 34 minutes. This means that the time required by the SBR algorithm for approximating the data is just the 1/68 of the time it took the sensor to collect it, thus making it easy for the SBR algorithm to run in parallel with the collection process. If a shorter running time of SBR is desired, one can simply either execute the algorithm with a smaller value of n,[8] or decide not to update the base signal, which is by far the most expensive part of the SBR algorithm, in each invocation. The latter method is not expected to affect the quality of the approximation significantly, since our experiments have demonstrated that after the first transmissions few base intervals are inserted in the base signal, because the current base signal is already of good quality at that point. Notice that if the base signal is not updated, then only the `GetIntervals()` algorithm is invoked, resulting in an overall running time complexity of: $O(n\log(\frac{TotalBand}{N}) + TotalBand \times \sqrt{n} \times M_{base})$. In this case the algorithm exhibits a linear dependency on the size of the processed data n.

4.6 Handling other error metrics

We now present the necessary modifications to the Regression algorithm of Sect. 4.2 when the desired error metric involves minimizing the sum squared relative error, or the maximum absolute error of the approximation.

The Regression algorithm approximates the data value $Y[i + start\_y]$ as $a \times X[i + start\_x] + b$. The relative error induced by this approximation is:

$$\frac{|Y[i + start\_y] - a \times X[i + start\_x] - b|}{max\{c, |Y[i + start\_y]|\}}$$

where the c value serves as a *sanity* bound, and helps avoid very large relative error values when the $Y[i + start\_y]$

[8] For example, when reducing the value of n to 10,240 data values, the corresponding running time of SBR is just 14.4 seconds.

value is either zero, or close to zero. The Regression algorithm that minimizes the sum squared relative error of the approximation is presented in Algorithm 8.

Calculating the a, b parameters that minimize the maximum absolute error of the approximation is somewhat harder to accomplish. The solution is based on the well known Chebyshev approximation problem, which can be solved with a randomized linear programming algorithm in O(*length*) randomized expected time and O(*length*) space.

4.7 Providing strict error bounds

The SBR algorithm, as presented above, seeks to minimize a user-defined error metric (i.e., sum squared error) given a target bandwidth constraint. An interesting extension is when the application requires strict error bounds. The typical goal in such cases is to minimize the maximum error of the approximation and provide this maximum error along with the approximate signal. In this case, a `Regression()` subroutine for minimizing the maximum error of the approximation (see Sect. 4.6) should be used.

Another interesting case occurs when the application provides a target size *TargetBand* and an error target with which it will be satisfied. In this case, the application will be satisfied with any approximation of size less or equal to *TargetBand* that satisfies the error target (if such an approximation exists). In this case the recursive procedure of the `GetIntervals()` algorithm may be stopped if the error target is achieved before the size of the transmitted data reaches *TargetBand*.

It is important to emphasize that in these application scenarios, whenever the base signal is not updated, it is easy for the sensor to also report to the base station the error of the approximation for multiple synopses with size less than *TotalBand*. Note that this is feasible because at each step of the GetIntervals() algorithm we are fully aware of the approximation error for each data interval. Reporting the errors for multiple synopses sizes can be helpful, since the application can then properly select the size of the transmitted data given its requirements on the accuracy of the compressed representation.

4.8 Node operation when the bandwidth changes

In our discussion so far, each sensor is aware of the total size *TotalBand* for its compressed, transmitted data. However, there might be situations when the value of *TotalBand* may change. For example, the error of the compressed data may be so small that the base station may decide that the error would be acceptable if it decided to limit the desired value of *TotalBand*. Moreover, in cases when the network dynamics change due to either node/link failures or changes in the available bandwidth due to cross-traffic, it might also be desirable to modify the desired size of the transmitted data. Obviously, if the node is notified for the new value of *TotalBand* before it initiates the execution of SBR, no modifications are necessary to the algorithm. On the other hand, when the new value of *TotalBand* is received *during* the execution of the algorithm, it would be desirable if several steps of the algorithm can be reused, in order to avoid executing it from the beginning. Depending on whether the sensor decided to update its base signal (or simply executed the `GetIntervals()` algorithm), the following optimizations can be made:

1. If only the `GetIntervals()` algorithm is executed, then if the size of the compressed data (calculated by the number of created data intervals) is less than *TotalBand*, then the algorithm simply continues its operation until it reaches this compressed data size. If, however, the size of the compressed data has exceeded the value *TotalBand*/4, then we can either re-execute the `GetIntervals()` algorithm (in which case we do not reuse any of the algorithm's steps), or keep a list of the choices (Line 12 in Algorithm 3) that the algorithm has made, in order to backtrack to a stage where the new space constraint is met. For this list of choices we simply need to store the starting point of the split interval at each step. This starting point will obviously be the same as the starting point of the first subinterval created by `GetIntervals()`, while the length of this first subinterval reveals the starting point (and the length) of the second subinterval.
2. If the base signal is updated, then obviously the results of the `GetBase()` algorithm, which is computationally the most expensive part of SBR, can be reused. If the change in the bandwidth constraint is not significant and the SBR algorithm has already decided how many CBIs to insert into the base signal, then we can simply execute the `GetIntervals()` algorithm on the updated base signal using the new value of *TotalBand* (and subtracting the size of the newly inserted CBIs). If the change in the bandwidth constraint is significant, then the errors calculated by SBR in Line 6 of the algorithm may differ significantly from the true errors, given the new constraint.

Algorithm 8 Regression Subroutine that Minimizes the Sum of the Squared Relative Errors

Require: $\vec{X}, \vec{Y}, start\_x, start\_y, length, sanity$

1: $sum\_x = \sum_{0 \le i < length} \frac{X[i+start\_x]}{max\{sanity, |Y[i+start\_y]|\}}$
2: $sum\_y = \sum_{0 \le i < length} \frac{Y[i+start\_y]}{max\{sanity, |Y[i+start\_y]|\}}$
3: $sum\_xy = \sum_{0 \le i < length} \frac{X[i+start\_x]Y[i+start\_y]}{max\{sanity, |Y[i+start\_y]|\}}$
4: $sum\_x2 = \sum_{0 \le i < length} \frac{X[i+start\_x]^2}{max\{sanity, |Y[i+start\_y]|\}}$
5: $sum\_z = \sum_{0 \le i < length} \frac{1}{max\{sanity, |Y[i+start\_y]|\}}$
6: {Compute Regression Parameters}
7: $a = \frac{sum\_z \times sum\_x\_y - sum\_x \times sum\_y}{sum\_z \times sum\_x2 - sum\_x \times sum\_x}$
8: $b = \frac{sum\_y - a \times sum\_x}{sum\_z}$
{Compute sum squared relative error of signal $\vec{Y'} = a\vec{X} + b$ in range $[start\_y \ldots start\_y + length)$}
9: $err = \sum_{i=0}^{length-1} (\frac{Y[i+start\_y] - (a \times X[i+start\_x] + b)}{max\{sanity, |Y[i+start\_y]|\}})^2$
10: return (a, b, err)

Thus, in this case only the results of GetBase() can be reused.

5 Localized groups

5.1 Framework description

It would often be advantageous to collect the input of several sensors on a single node, and perform the approximation for all values simultaneously. We expect that several quantities (like temperature, pressure, etc.) observed by neighboring nodes in the network will exhibit similar trends. Thus, many intervals of the base signal in one sensor could approximate well intervals of signals from its neighbors. A non-localized algorithm would not be able to detect this, and would include in the base signals of individual nodes intervals of similar features. Moreover, a localized algorithm that operates on the collected measurements of multiple sensors can make better decisions involving the distribution of the approximated intervals over them, thus further reducing the error of the approximation.

Assume a group of S sensors that operate individually. Each sensor i transmits an approximation of the $N \times M$ values it collects of size $k_i \times (N \times M)$. The ratio $k_i < 1$ of the size of the transmitted data over the size of the collected data is called the "compression ratio". For ease of exposition, we first assume that for all sensors in the group the number of monitored quantities, their sampling rates and their compression ratios k_i are the same. We will deviate from this latter assumption later in this section. If the base station is, on average, H hops away then the volume of data sent is (for $k_i = k$)

$$data_{non-localized} = k \times (N \times M) \times S \times H$$

The alternative organization that we explore in this section is depicted in Fig. 1. The sensors nodes are organized as a group and one of them is assigned to act as the group leader. The rest of the sensors in the group (S-1 in total) will send it an approximation of their measurements using a compression ratio k_i. For now, let us assume that all values of k_i are the same (i.e., equal to k_1). Without loss of generality, we also assume that the group leader is one hop away from each sensor in the group (i.e., the group is of radius 1), as shown in the figure. While this later assumption is made to simplify our presentation, the modifications to our formulas for groups of larger radii are straightforward.

The group leader will approximate all values in the group ($S \times N \times M$ in total) with a compression ratio k_{gl}. We note that this process does not require for the group leader to decompress the values from the other nodes in the group. Our algorithms can be rewritten to operate directly on the transmitted approximate values that essentially describe a piecewise approximation of each series Y_i. The approximation of each data interval (from the other sensors) is easily computed from the respective base interval of that sensor that is used to approximate this interval using linear regression. It is important to emphasize though that:

1. The group leader needs to maintain and update the base intervals transmitted to it by the other sensors in its group.
2. The base signal of the group leader is constructed using the group leader's collected data and the (approximate) reconstructed data values received by the other sensors of the group.

Using an organization of the sensors like the one depicted in Fig. 1, the total volume of data transmitted will now be:

$$data_{sensors \rightarrow group\ leader} + data_{group\ leader \rightarrow base\ station} = (k_1 \times (S-1) + k_{gl} \times S \times H) \times (N \times M)$$

The goal of our localized processing algorithm is to construct a more accurate approximate data representation of the data, while using at most as much bandwidth as in a non-localized organization. To achieve the latter requirement, our localized algorithm will control the values of k_{gl} and k_1 based on the overall data reduction factor k that the non-localized organization would use, by enforcing that:

$$k_1 \times (S-1) + k_{gl} \times S \times H \leq k \times S \times H$$
$$\Longrightarrow k_1 \leq (k - k_{gl}) \times \frac{S}{S-1} H \quad (1)$$

If, due to correlations among the measurements of the sensors in the group, we can achieve a higher reduction in the data transmitted from the group leader to the base station (i.e., $k_{gl} < k$), then we can use more bandwidth for sending the initial measurements to the group leader (i.e., $k_1 > k$). As an example, assume that we target an overall compression ratio $k = 10\%$ for a group of five sensors ($S = 5$) and that $H = 20$. For k_{gl} equal to e.g. 7% we get $k_1 = 3 * 5/4 * 20 = 75\%$ that is much higher than the target $k = 10\%$. Notice that the maximum value of k_1 is linear to the number of wireless hops. Thus, the further away the base station is, the more leverage we get. In fact if

$$H > \frac{S}{S-1} \frac{1}{k - k_{gl}}$$

i.e., 27 hops for our example, we can afford sending the exact, uncompressed data values to the group leader.

5.2 Tuning the compression ratios

Previously, we assumed that each sensor node in the group (other than the group leader) uses the same compression ratio k_1 to summarize its values. There are many reasons why this "uniform" allocation of bandwidth among the sensors will be sub-optimal. For instance, all sensor nodes will not necessarily monitor the same quantities. Some sensor nodes within a group may obtain meteorological measurements, while other nodes may obtain measurements involving noise and chemical levels, number of detected objects etc. It is expected that some measured quantities will be harder to accurately approximate than others. The same may also occur even among nodes collecting the same type of data, simply

because of some spatio-temporal characteristics of the monitored quantities.

In what follows we describe an algorithm for tuning the compression ratio used by each sensor to communicate with the group leader. A high level view of the algorithm is as follows. Each sensor node in the group will execute the SBR algorithm for a selected target bandwidth smaller than the one required by the application and will inform the group leader of the error of the approximation obtained. The group leader will then adjust the individual compression ratios used by the nodes in its group, allowing for more bandwidth on nodes whose values are harder to approximate. During this process the nodes will execute the SBR algorithm only once. When additional bandwidth is given to a node, this node will call a re-entrant version of the `GetIntervals()` subroutine to further partition the previously created data intervals.

We assume as input to the algorithm the desired total bandwidth consumption B, the number of nodes in the group S, the distance of the group leader from the base station H. Without loss of generality, we assume that all nodes have the same amount of data $N \times M$; this only simplifies notation in the presentation. Bandwidth B includes the cost of retransmissions. Thus, B translates to an overall (average) compression ratio of

$$k = \frac{B}{S \times H \times N \times M}$$

in the non-localized setting.

The group leader first determines the compression ratio k_{gl} that it will use for its own transmission. In our experimental evaluation we demonstrate how the value of k_{gl} can be properly selected, given the bandwidth consumption B. Each node in the group other than the group leader is initially assigned the same compression ratio $k_i = (k - k_{gl}) \times \frac{S}{S-1} H$ (see Sect. 5.1). This implies a bandwidth $B_i = k_i \times N \times M$ for the transmission of node i to the group leader.

The key part of the algorithm involves the adjustment of the B_i bandwidth units to the $S - 1$ nodes of the group (besides the group leader) for the transmission of their values to the group leader. The adjustment process distributes a portion of the total bandwidth to the nodes that exhibit the largest errors in their approximation, while also taking into account the bandwidth consumed by each node in its previous transmission. To accomplish this, each sensor node i executes the SBR algorithm and reports to the group leader the resulting error using a smaller value for B_i than the one used during its last transmission. In particular, it scales B_i by a factor λ: $B_i(t) = \lambda \times B_i(t-1)$ (e.g. $\lambda = 0.80$). In the initial transmission, we set $B_i(t-1) = k_i \times N \times M$. Notice that this step does not involve transmitting any compressed data. Node i simply executes the algorithm, reports the error and awaits for further instructions.

Since each node in the group has reported errors for a reduced bandwidth $B_i(t)$, there is a residual bandwidth of $B_{residual} = (1 - \lambda) \sum B_i(t-1)$ that can be redistributed among the $S - 1$ nodes. The group leader considers the reported error of each node and continuously allocates to the node with the largest error additional bandwidth $B_{residual} \times \alpha^{-1}$, where α is an integer parameter that controls the number of performed iterations. In our experiments we use $\alpha = S - 1$. Essentially, this process continuously allocates additional bandwidth to nodes that need it the most and results in the reduction of the approximation error of these nodes. Each node will ultimately transmit their compressed data to the group leader only after this bandwidth partitioning process has been completed, using the B_i value achieved at the end of the process. An alternative bandwidth dissemination option would have been to use a two-step dissemination process, where the additional bandwidth assigned to each node during the second step would be proportional to its data reconstruction error. This two-step bandwidth allocation process leads to smaller execution times of the overall compression process, but may result in sub-optimal bandwidth distribution in cases where nodes can significantly reduce their approximation error with a small increase in their bandwidth consumption.

A key observation is that nodes that receive additional bandwidth do not need to run the SBR algorithm again. These nodes have already created a base signal and have partitioned their collected data into intervals approximated by linear regression with some part of this base signal. When additional bandwidth is given, the node may simply call a modified `GetIntervals()` subroutine that continuous partitioning the previously created data intervals (instead of starting from the original data), until the new bandwidth limit is reached. It then notifies the group leader for the new error obtained. This incremental evaluation of the algorithm is essential for the efficient execution of the compression process.

When $k_i < 1$, our localized schema assumes that the group leader maintains an up-to-date replica of the base signal of sensor node i in the group, of size M_{base}, in order to be able to reverse the transmitted encodings. In the experiments we see that real data need very small base signals which suggest that this assumption is reasonable.

In Fig. 6 we demonstrate the end-to-end process. First, the base station informs the group leader for the bandwidth consumption limit B. The group leader then allocates the same bandwidth B_i to all nodes in its group and each node reports its error after running the SBR algorithm, Fig. 6(b,c). In Fig. 6(d), the group leader assigns extra bandwidth to the node with the largest error (node 2 in this example). In turn, node 2 reports the new error E_2' obtained with the extra bandwidth by a continuing execution of subroutine `GetIntervals()` (Fig. 6(e)). These two steps are repeated α times, assigning each time additional bandwidth to the node with the largest error. In Fig. 6(f), all nodes transmit their values to the group leader. Finally, the group leader executes the SBR algorithm and transmits the compressed data to the base station.

5.3 Selecting the group leader

In our discussion so far we have not referenced the procedure with which the sensor nodes (i) determine how many

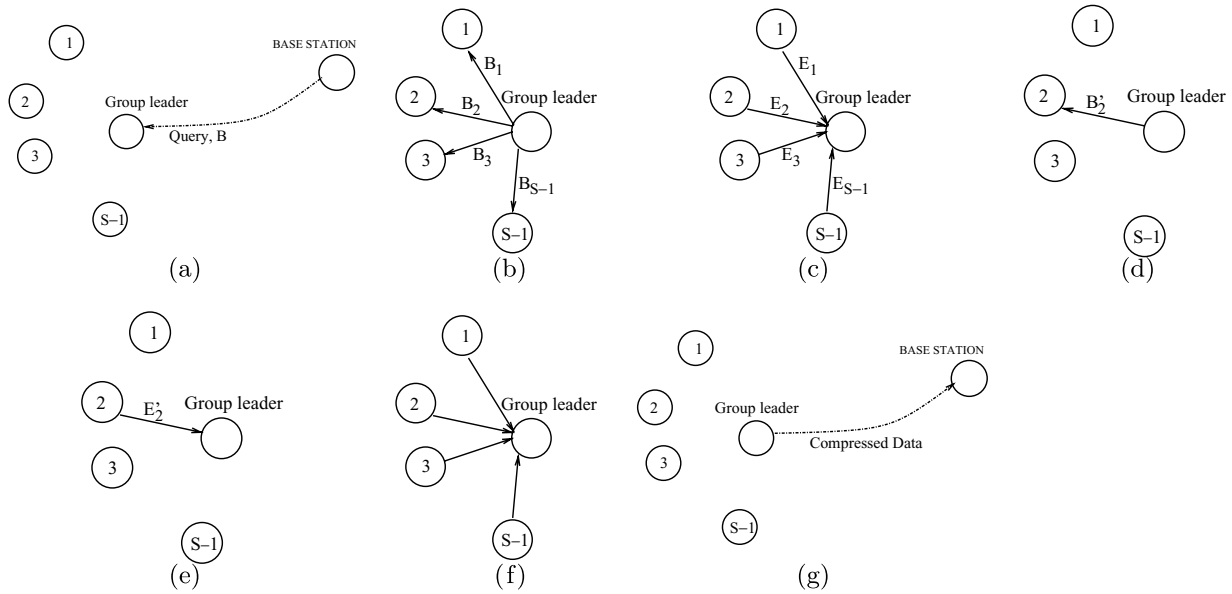

Fig. 6 **a** Transmission of query and desired Bandwidth consumption B. **b** Group Leader assigns initial bandwidth limit to nodes in group. **c** Nodes report error with given bandwidth limit. **d** and **e** Handshake procedure: Group Leader assigns additional bandwidth to node with largest error, and receives from this node a new reported error. **f** At the end of the handshake process, nodes transmit their compressed data. **g** Group leader compresses data from all nodes in group and transmits them towards the base station

localized groups to form, (ii) detect their localized group, and (iii) select the group leader of each group. Our solution to this problem is motivated by the recently proposed HEED protocol, described in [58], which describes a distributed and energy-efficient way of clustering sensor nodes in a way that seeks to maximize the lifetime of the network. This clustering phase is performed within a provably small number (O(1)) of phases (iterations), thus ensuring high scalability and low reorganization cost, While the cost model that we will use in our localized scheme differs from the approach envisioned in [58], the techniques proposed in [58] are still applicable, with some modifications.

Consider that the base station issues a query over the data collected by the sensor nodes in an area of the network over the last M epochs, along with a desired average compression ratio k. The query is disseminated through the network in search of the nodes that meet the query's selection criteria. The resulting set S of nodes, that will transmit their collected data in response to the query initiate a process for selecting the group leaders amongst all nodes in S. The number of group leaders is not known beforehand. We will describe this process shortly. After the set GL of group leaders has been determined, each node in $S - GL$ needs to inform one of the nodes in GL that it will be a member of its group. Most sensor nodes can adjust their transmission power based on the distance of the node with which they wish to communicate ([58]). It is, thus, optimal in terms of energy consumption for a node to select as group leader the node in GL which lies the closest to it, since the energy drain during transmission will be minimized with this approach.

Now, let us consider how the set of group leaders is selected. The operation of the HEED protocol consists of multiple steps (iterations), during which sensor nodes initially become *tentative* group leaders and, later, some of these tentative group leaders will ultimately form the final list of group leaders. Consider a scenario where all the sensor nodes possess the same processing and memory characteristics. If this is not the case, then only the most powerful of the nodes will attempt to become group leaders.

Since the group leaders process larger amounts of data, which, in turn, results in a larger energy drain, it would be preferable if the group leaders are amongst the nodes with the largest remaining energy. Consider a single sensor node and let E_{init} denote its initial (maximum) energy, while E_{curr} denotes the current energy of the node. At each iteration of the clustering protocol, if this sensor does not lie within the radius of a group leader, it will elect to become a group leader with an initial probability $CH_{prob} = C_{prob} \times \frac{E_{curr}}{E_{init}}$, where C_{prob} has a suggested value of 0.05 in [58]. This initial probability CH_{prob} is then doubled in each iteration for all these "uncovered" nodes. A new group leader is considered to be *tentative* if its CH_{prob} is lower than 1 and *final*, otherwise. Each new group leader transmits a message to the sensors within its transmission range containing: (i) The estimated cost, in terms of energy, for all the nodes in its radius to transmit their uncompressed measurements to it (calculated using the transmission power needed by each of these nodes to reach this group leader), and (ii) The estimated distance (in the number of hops) of the group leader from the base station.

Nodes within the radius of a set S_{GL} of group leaders elect the group leader with the minimum advertised cost (described above) to join. In case of a tie, the group leader that is the closest to the base station is selected (this could

be the node itself). When the value of CH_{prob} reaches the value 1, the selection of the group leader becomes final. At the final step of the algorithm, all "uncovered" nodes will elect to perform the compression of their measurements individually.

There is one more detail that we have not discussed so far. When the group leader of a sensor node N_i changes, the sensor needs to construct a new base signal for its new group leader GL_i, since the new group leader is not aware of the base intervals that the node transmitted to its previous group leader. Even if the sensor N_i had used the same group leader GL_i in the past, it is not guaranteed that GL_i will have kept in its buffers the candidate base intervals of N_i, since this depends on its memory and storage capabilities and the time that has elapsed between these two events. Of course, if the node N_i (group leader GL_i) have enough memory to maintain the base intervals transmitted to GL_i (received from N_i), then N_i may use these base intervals for the approximation of its measurements. Finally, we need to emphasize that transferring the base signal and the individual base signals of *all* the nodes in the group from the previous group leader to the new group leader is not often a good decision, even if the nodes in the group have not changed, as this procedure may consume a significant fraction of the available bandwidth. Thus, it may be preferable from a node to voluntarily *surrender* being a group leader only when its energy drops significantly, compared to the energy of the other nodes in its group. In this case, this group leader transmits a message and initiates the process described above for selecting a new group leader only among the nodes in its group (and not in a global fashion at each data transmission).

5.4 Alternative group operation

The execution of the SBR algorithm may be infeasible for sensor nodes with very limited capabilities. For such sensors, or generally in the case of hybrid sensor networks with nodes that possess widely different capabilities, the operation of the localized group may be slightly altered in order to help the low-end sensors to compress their data using the base signal of the group leader. The alternative group operation that we explore is one where the group leader initially constructs a base signal based solely on the data that it has collected. The group leader then transmits this base signal to the nodes in its group using a single broadcast message. The other nodes in the group will then use this base signal and compress their data by simply calling the `GetIntervals()` algorithm. Note that the handshaking process described above, where the bandwidth allocated to each node is determined after several steps, will still be followed. The group leader simply has to take into account the size of its transmitted base signal when deciding on the amount of bandwidth that the sensor nodes in its group will use to transmit their measurements to the group leader. Also, notice that in this case the group leader has already performed an approximation of its own data, and has computed the error of this approximation. Therefore, we can also perform the following two modifications to the localized group operation algorithm:

1. At each step of the handshaking process, the group leader may decide to allocate additional bandwidth to itself, instead of strictly allocating bandwidth to the other nodes within its group. Similarly, if a 2-step approach is followed for the bandwidth dissemination (see Sect. 5.2), then the node will also allocate part of the bandwidth to the size of its own compressed data representation.
2. Since the group leader has allocated bandwidth to each node within its group based on the approximation error of each node, and due to the common base signal being used by all nodes, there is no need for the group leader to re-compress the data that it receives from the nodes within its group.

If $B_{BaseSignal} = Ins \times (W + 1)$ denotes the bandwidth needed to transmit the updates to the base signal (determined by the group leader) and k_1 denotes the average compression ratio used by the nodes in the group (including the group leader), then the overall bandwidth consumption will be:

$$B_{BaseSignal} \times (H + 1) + k_1 \times N \times M \times (S \times H + S - 1)$$

Thus, since we would like the localized organization not to exceed the bandwidth consumption B specified by the query, the value of k_1 must be set to:

$$k_1 \leq \frac{B - B_{BaseSignal}}{N \times M \times (S \times H + S - 1)}$$

This alternative localized processing algorithm results in significant energy savings compared to the organization model proposed in Sect. 5.2, both for the group leader, since it compresses only its own data, but also for the remaining nodes within the group, since they only need to execute the `GetIntervals()` algorithm. Moreover, no changes are needed if the group leader changes (if the nodes comprising the group remain the same), since all the nodes within the group are aware of the used base signal. On the other hand, the accuracy of the compressed data will depend on the quality of the base signal constructed by the group leader and whether this base signal contains patterns observed frequently by the other nodes within the group.

6 Experiments

In this section, we provide an experimental evaluation of our techniques. In Sect. 6.1 we compare the SBR algorithm against standard approximation techniques (DCT, Wavelets, Histograms). In Sect. 6.2 we compare the `GetBase()` algorithm against alternative base signal constructions, while in Sect. 6.3 we present an analysis of the SBR algorithm. In Sect. 6.4 we evaluate the localized mode of operation and draw direct comparisons against the non-localized setting. For these experiments we used the following real data sets:

Table 2 Properties of used data sets

Data set	Min	Max	Avg	Min(Var(Y_i))	Max(Var(Y_i))	Avg(Var(Y_i))
`Phone Call Data`	3	38,472.0	4,173.9	986,510.9	119,272,823.3	21,313,902.7
`Weather Data`	0	514.0	35.1	14.1	5,179.1	895.9
`Stock Data`	16	213.9	93.2	1.3	75.4	17.8

1. `Phone Call Data:` Includes the number of long distance calls originating from 15 states (AZ, CA, CO, CT, FL, GA, IL, IN, MD, MN, MO, NJ, NY, TX, WA). For each state we provide the number of calls per minute for a period of 19 days (data provided by AT&T Labs).
2. `Weather Data:` Includes the air temperature, dew-point temperature, wind speed, wind peak, solar irradiance and relative humidity weather measurements for the station in the university of Washington, and for year 2002.[9]
3. `Stock Data:` Includes information on all trades performed in a minute basis over April 3 and April 4 of year 2000. The approximated measure in our experiments is the trade value of the stock.

Table 2 compiles a few simple statistics about these data sets.

6.1 Comparison to alternative techniques

For this experiment we used all three data sets described above. From the *Stock* data, we extracted the trade values of the following ten ($N = 10$) stocks: Microsoft, Oracle, Intel, Dell, Yahoo, Nokia, Cisco, WorldCom, Ariba and Legato Systems. For each stock we created a random sample of 20,480 of its trade values, and then split each sample in ten files of 2,048 values each. The first of these ten files of each stock was used for the initial creation of our base signal, while the remaining files were used to simulate nine update operations. For the *Weather* data set, we selected the first 40,960 records and then split the data measurements of each signal into ten files of 4,096 values each. For the *Phone Call* data set, the aggregates for each state ($N = 15$) were broken into ten files of 2,560 values each.

In our experiments we compared the accuracy of SBR against the approximations obtained by using the *Wavelet* decomposition [7], equi-depth Histograms [45] and the DCT. The Fourier transform was also considered, but produced consistently larger errors than DCT and is thus omitted. For a fair comparison we set the space used by all methods to the exact same amount.

For all methods we considered both treating each bunch of updates as a group of N series $\vec{Y}_i$ each of length M and, alternatively, concatenating the signals into a single series Y of length $N \times M$. For Wavelets, we found out that this produced in most cases significantly more accurate results than by dividing the space equally among the N signals (by a factor of 5 in many cases) because some signals needed more wavelet coefficients than others to be approximated well. For Wavelets, we also considered a 2-dimensional decomposition of the $N \times M$ values, which produced worse results than the 1-dimensional decomposition. We here present the best results achieved by each method.

6.1.1 Varying the compression ratio

We varied the compression ratio (size of the transmitted data *TotalBand* over the data size n) from 5% to 50%. In this experiment we set M_{base} to 2,048 values for the *Phone Call* and the *Stocks* data sets and to 3,456 values for the *Weather* data set. In Tables 3 and 4 we present the results. In all data sets SBR produces significantly more accurate results than the other approximations. The difference is larger for the *Phone Call* data set which contained the largest values. As the size of transmitted data increases, the error in our method decreases more sharply, and is up to 8.5 times smaller than the error of Wavelets. The DCT and the Histogram approximations produced much larger errors is most cases.

We repeated the experiment for the *Phone Call* data set, computing this time the sum-squared relative error. The results are also shown in Table 4. For this error metric, our method often provided errors that were more than an order of magnitude smaller than the errors achieved by Wavelets and DCT, while the improvements were even larger when compared to histograms. We note here that for this comparison we used Haar Wavelets that are optimal only under the sum-squared-error. The work of [21] describes algorithms for minimizing, among other metrics, the relative error of a Wavelet-based approximation. Except for cases of very skewed data sets, these algorithms reduce the mean relative error up to 3 times over regular Wavelets. These improvements were seen for very coarse approximations (i.e., for a compression ratio of 5% or less) where our method already has an advantage of 42-1 over regular Wavelets. For more space, these techniques are a lot closer to regular Wavelets.

The increased accuracy of SBR over techniques like Wavelets and DCT is not surprising. Similarly to SBR, both the Wavelets and DCT utilize a basis in order to compress the data. However, while this basis is data-independent in the cases of Wavelets and DCT, SBR extracts the values contained in the base signal from the actual data. Finally, in order to explain the increased accuracy achieved by SBR, one has to consider what happens when a data set contains multiple correlated data intervals that are hard to approximate. In this case, the studied competitive techniques will

[9] http://www-k12.atmos.washington.edu/k12/grayskies

Table 3 Total SSE varying the compression ratio for weather and stock data sets

Compression ratio %	Weather data (245,760 total values)				Stock data (204,800 total values)			
	SBR	Wavelets	DCT	Histograms	SBR	Wavelets	DCT	Histograms
5	317,238	519,303	8,703,192	7,661,293	18,376	25,856	47,722	58,531
10	103,633	200,501	4,923,294	3,375,518	6,800	11,993	42,927	49,849
15	54,219	125,449	3,515,698	2,219,533	3,423	7,179	39,725	46,305
20	30,946	87,118	2,643,229	1,471,421	1,913	4,769	37,075	44,266
25	18,600	63,105	2,198,455	946,735	1,126	3,345	34,464	42,470
30	11,558	46,833	1,598,451	594,644	700	2,410	31,882	40,290
35	7,161	35,275	1,366,211	410,208	442	1,769	29,506	38,982
40	4,603	26,824	1,112,117	288,127	291	1,316	27,035	37,454
45	2,964	20,502	905,422	236,947	190	991	24,608	36,351
50	1,861	15,762	768,568	160,079	125	748	22,428	34,997

Table 4 Errors Varying the Compression Ratio for Phone Call Data Set

Compression Ratio %	Average Sum Squared Error				Total Sum Squared Relative Error			
	SBR	Wavelets	DCT	Histograms	SBR	Wavelets	DCT	Histograms
5	9,631	29,938	15,714	172,964	922	38,477	9,019	126,346
10	5,071	12,349	10,173	55,126	503	19,186	3,002	56,749
15	3,192	7,998	6,767	29,703	325	12,885	1,400	58,636
20	2,170	5,821	5,661	20,494	222	10,954	1,192	38,123
25	1,527	4,468	4,791	16,103	158	6,915	823	37,846
30	1,091	3,537	4,157	12,898	116	3,865	721	34,654
35	786	2,853	3,692	10,898	85	2,404	665	31,497
40	568	2,331	3,324	9,459	62	2,092	583	27,220
45	398	1,918	2,939	8,228	45	1,826	445	26,031
50	281	1,587	2,622	7,055	32	1,674	410	24,285

spend the same amount of effort (space) approximating each of these data intervals. On the other hand, SBR has the option of including just one of these intervals in the base signal to help accurately compress the other intervals, while devoting very small space for them.

6.1.2 Modifying the data correlations

The SBR algorithms exploits intrinsic correlations between the signals. We now explore its behavior when these correlations are reduced. At first, we tried mixing data from our three data sets. We created a data set that contains phone call data from three states (AZ, CA and FL), three types of meteorological measurements (air temperature, pressure and solar irradiance), and data from three stocks (Microsoft, Intel and Oracle). For each of these data series we created ten files of 2,048 values each. We then varied the compression ratio of all algorithms from 5% to 30% and set M_{base} to 2,048 values. In Table 5 we present the average sum squared and total sum squared relative errors for all methods. The relative performance of the SBR algorithm is even better compared to the other methods: the SBR algorithm produced up to 27 times smaller average sum squared errors than the closest competitor, while the improvement reached up to 1,034 times for the total sum squared relative error.

While the results may seem surprising because the correlation between the data sets was decreased, they are not counter-intuitive. All the approximation methods exploit some form of correlation or redundancy to reduce the footprint of the data. Table 5 simply shows that SBR is more robust, than Wavelets and Histograms for example, when such correlations are reduced. The design of the algorithm allows it to find correlations even in such cases, between intervals from different signals and different time periods. The algorithm also has a fall-back plan of using plain regression when such correlations are not strong. In such cases, fewer space is allocated for the base signal and most of the transmitted values are used for approximating more, shorter intervals.

We now also explore the sensitivity of all methods to the correlation of the time series by generating synthetic sensor streams, where the correlation amongst their values is varied. We started with a set of four streams of 20,480 values that we broke into 10 pieces for simulating updates ($N = 4$, $M = 2{,}048$). The first stream was produced using the identity function, the second using the step (Haar) function. The third stream was from a cosine function while the fourth one was depicting normally distributed data (bell-shaped curve). All streams where normalized within range [-20..20]. We then started perturbing the values by adding white Gaussian noise at random places. During this process we executed each data reduction algorithm for a compression ratio of 10% and computed the average sum squared error and the total sum squarred relative error of the approximation.

Table 5 Errors varying the compression ratio for the mixed data set

Compression ratio %	Average sum squared error				Total sum squared relative error			
	SBR	Wavelets	DCT	Histograms	SBR	Wavelets	DCT	Histograms
5	2,900	8,094	12,677	199,150	113	20,974	29,625	182,027
10	918	3,020	7,146	46,805	37	11,054	8,653	43,701
15	364	1,582	4,757	23,711	17	5,481	4,825	26,068
20	139	894	3,814	14,157	9	5,310	3,339	14,780
25	46	516	3,120	10,486	5	5,172	6,115	11,118
30	11	297	2,680	6,894	3	5,109	1,579	9,591

In Figs. 7 and 8 we plot the results (for SBR, we set M_{base} to 256 values and $W = 64$). The x-axis in both Figures is the *fractal dimension* of the data set, as defined in [5]. As explained in [5] the fractal dimension is a single positive number that describes the degree of freedom amongst the four data-streams. A larger number denotes more random/uncorrelated values amongst the data streams due to the addition of white noise. As expected, all methods substantially degrade their quality of approximation when the fractal dimension increases. Moreover, we can see that the improvements of SBR over the competitive techniques are very large when strong correlations occur (fractal dimension close to 1), while when the correlation becomes very small due to the added white noise, Wavelets and DCT produce competitive errors to SBR.

6.2 Alternative base signal constructions

We present two alternative algorithms to `GetBase()`. The first, denoted as `GetBaseSVD()`, is based on the Singular Value Decomposition. The second algorithm, denoted as `GetBaseDCT()`, uses the basis of the Discrete Cosine Transform (DCT), which is a collection of cosine functions. Finally, a third alternative for SBR is to do standard linear regression without using a specially constructed base signal. For the later case, no bandwidth is consumed for sending base signal values and we do not need the *I.shift* pointer. Thus, we can send exactly *TotalBand*/3 intervals for a bandwidth limit *TotalBand*. Similarly, the DCT base consists of cosine functions and its values are constructed on the fly and are thus neither stored in memory, nor are they transmitted to the base station.

Construction using SVD

SVD involves computing the eigenvectors and eigenvalues of a given $N \times n$ matrix R. It can be proven (see [46]) that any real $N \times n$ matrix can be written as:

$$R = U \times \Lambda \times V^t$$

where U is a column-orthonormal $N \times r$ matrix, r is the *rank* of matrix R, Λ is a diagonal $r \times r$ matrix of the eigenvalues λ_i of R and V is a column-orthonormal $n \times r$ matrix. By definition $U^t \times U = V^t \times V = I$, where I is the identity matrix. The columns of V are the eigenvectors of matrix $R^t \times R$. Similarly, the eigenvalues of $R^t \times R$ are the squares of λ_is:

$$R^t \times R = V \times \Lambda^2 \times V^t$$

For $R = A$ (our collected measurements), $R^t \times R$ captures the similarities among the columns of A (each collected sample). SVD can be used for approximating $R^t \times R$

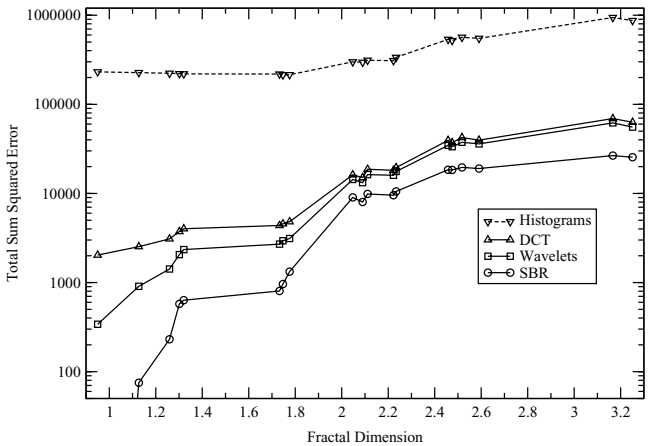

Fig. 7 Total SSE error vs Fractal Dimension

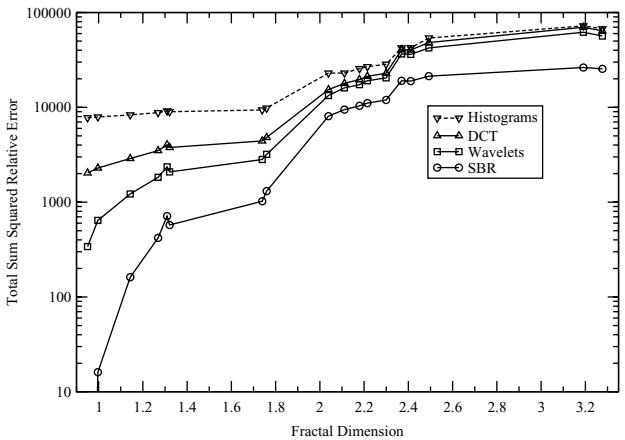

Fig. 8 Total SSRE error vs Fractal Dimension

Table 6 Comparison to alternative base signals

	Error over GetBase()		
Data Set	GetBaseSVD	Linear Regression	GetBaseDCT
Weather	10.55	4.47	6.44
Phone	1.13	1.32	1.19
Stock	2.08	2.77	2.99

by keeping the first few eigenvectors (columns of matrix V). Informally, each eigenvector captures linear trends among the rows of A (the $\vec{Y}_i$s).[10] We here propose the use of SVD as a competitor to the `GetBase()` algorithm for generating a base signal from the data. We sketch the new algorithm (`GetBaseSVD()`) below.

1. For each row of A, list all non-overlapping intervals of length W. This gives us $\frac{M}{W}$ intervals per row and $K = \frac{N \times M}{W}$ intervals overall.
2. Build a $K \times W$ matrix R whose rows are the intervals of the previous step.
3. Compute the SVD of $R = U \times \Lambda \times V^t$. Return the first $Store$ columns of V.

By definition, V is an $r \times W$ matrix ($r = rank(R)$) of the eigenvectors of $R^t \times R$. The eigenvectors are ordered from left to right in V. The first column of V contains the eigenvector (of length W) that corresponds to the largest eigenvalue of $R^t \times R$. The algorithm returns the top-$maxIns$ eigenvectors of total size $maxIns \times W$. These constitute the base signal from `GetBaseSVD()`.

Construction using DCT

The base signal can be constructed from the basis-vectors of standard mathematical transforms. We here present a base signal construction motivated by the Discrete Cosine Transform (DCT). Assuming that we are to use base intervals of length W, we enumerate all frequencies f such that $0 \leq f \leq W$. For each frequency f, we define a base interval with values $cos(\frac{(2i+1)\pi}{2W} f)$, where $0 \leq i < W$. We call this algorithm `GetBaseDCT()`. We notice that we do not need to store these intervals implicitly, as they can be computed on the fly.

In Table 6 we compare the approximations obtained by using the base signals computed by our `GetBase()` algorithm with the base signal from the alternative constructions. We need to emphasize here that for this experiment we modified the `BestMap()` function not to use linear regression as an alternative to using the base signal (so that the differences among `GetBase()`, `GetBaseSVD()`, `GetBaseDCT()` and linear regression are not diffused). Using the BestMap() function as presented in Section 4.2 would further improve the results of our method. The compression ratio was set to 10%. We notice that our `GetBase()` performs a lot better in the *Weather* data set, up to 10 times better than the alternative algorithms. For the *Phone Call* and the *Stock* data the differences are smaller but still significant.

[10] [33] presents an application of this observation in a different context.

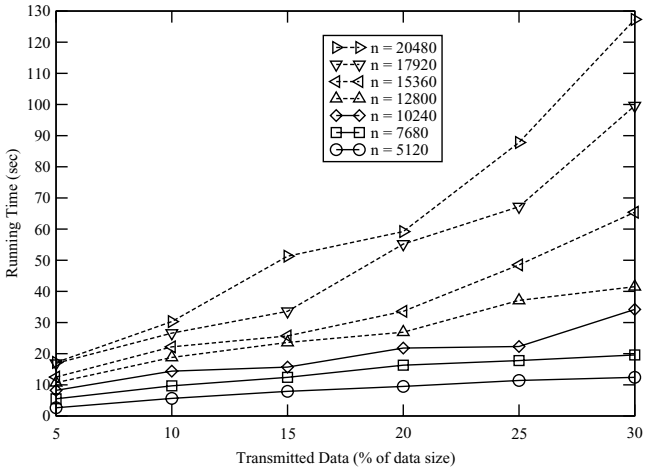

Fig. 9 Average Running Time vs TotalBand

6.3 Analysis of SBR

We now analyze several characteristics of the SBR algorithm, including its running time, the number of base intervals it selects for inclusion in the base signal and the quality of its decisions.

In Fig. 9 we plot the average time of each transmission operation for the Stock data set, when the size of the transmitted data is varied from 5% to 30% of the data size, the size of n varies from 5,120 to 20,480,[11] and the size of the base signal is 1,024. Since we have not yet ported our code to the StrongARM platform, we executed this experiment on a Irix machine using a 300 MHz processor. As expected (see Sect. 4.3) the running time scales linearly with the size of the transmitted data. Notice that SBR is significantly faster when greater reduction is obtained. For many practical applications, we expect to use a compression ratio of 10% (or even less), where running time varies from 5.6 to 30 seconds depending on the value of n.

The SBR algorithm dynamically decides the number of base signal values to use for an upper bound M_{base}. We now compare SBR against a straight-forward implementation that populates all the available space for the base signal. In Fig. 10 we plot the error of only the initial transmission as the size of the base signal is varied, manually, from 1 to 30 intervals for the *Phone, Stock* and *Weather* data sets. For this initial transmission we populated the entire space of the base signal using the `GetBase()` algorithm. For each data set we also show the selection that the SBR algorithm made, when deciding how many base intervals to populate. For presentation purposes the errors for each data set have

[11] By varying the value of M. We always used data from 10 stocks.

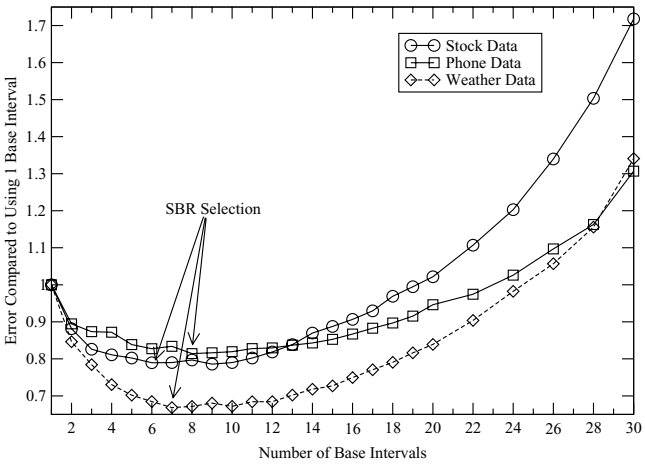

Fig. 10 SSE error vs base signal size

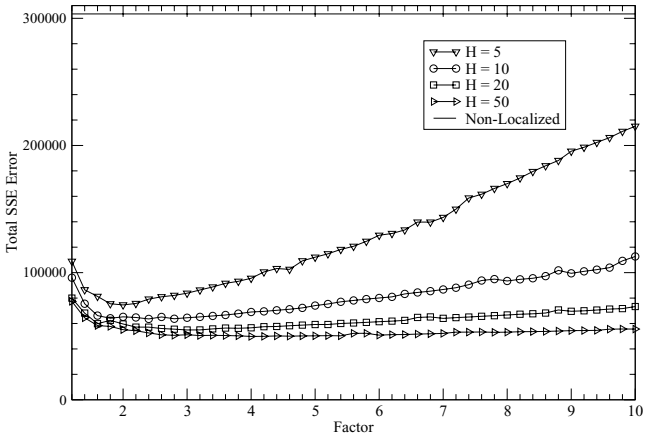

Fig. 11 Selecting the ratio $\frac{k_1}{k_{gl}}$ vs H

been divided by the error of the approximation when using just one interval. We set the size of each stock, phone and weather data file to 3072, 2048 and 5120 values respectively, in order for all data sets to have exactly the same size, and the *TotalBand* value to 5012, which results to a compression ratio ($TotalBand/n$) of about 16%.

The fixed value of the compression ratio implies that an increase in the size of the base signal results in a decrease in the number of intervals used to approximate the data values in order to keep the total space constant. After some point, the benefit of storing more intervals for the base signal is outweighted by the increase in the error that we get due to the reduced number of intervals used for the approximation. It is interesting to see that the optimal case occurs for a base size of between 7 (for the *Weather* data set) and 9 base intervals (for the *Stock* data set), which correspond to just 2.9% to 3.75% of the data size at the first transmission. The SBR algorithm made the optimal choice for the *Phone* and *Weather* data sets and produced a near-optimal solution for the *Stock* data set (it selected to insert 6 base intervals, instead of 9). We remind that the M_{base} base signal values need to be kept in the memory of the sensor in order to perform the approximation. Our results suggest that a very small fraction of memory needs to be sacrificed for these values.

For the same data setup, we report in Table 7 the number of inserted base intervals during the 10 transmissions. As we can see, most base intervals are inserted during the first two transmissions. We notice that there are many transmissions during which no new base intervals are inserted, and that the different data sets seem to contain a widely different number of features, with the Weather data set containing the most features, and the Stock data set containing the fewest.

The small number of intervals inserted in the base signal after the initial transmissions allows us to consider executing the SBR algorithm only periodically, or when the quality of the approximation degrades, in the case of constrained environments. For the other transmissions, the approximation may be performed by simply using the significantly faster `GetIntervals()` algorithm.

Table 7 Number of inserted base intervals per transmission

	Transmission									
Data Set	1	2	3	4	5	6	7	8	9	10
Weather	6	6	1	0	3	0	2	3	0	1
Phone	3	6	0	1	0	0	2	0	0	1
Stock	3	0	0	2	1	0	0	0	2	0

6.4 Localized groups

We now seek to evaluate the benefits of localized group processing described in Sect. 5. We first investigate which are the best values of the compression ratios k_1 and k_{gl}, as a function of the target compression ratio k.[12] We used a group of six sensors, each collecting 4096 values from a different type of weather measurements (i.e., one sensor monitoring air temperature, one sensor monitoring pressure etc). We set the compression ratio k of the non-localized approach to 10% and adjusted the ratio $\frac{k_1}{k_{gl}}$ so that the localized approach consumes the same amount of bandwidth. For a given value of k and for a fixed ratio of $\frac{k_1}{k_{gl}}$, the values of k_1 and k_{gl} follow easily from Eq. (1).

In all cases the number of transmissions (update operations) was set to 6, while $\lambda = 0.6$, and $M_{base} = 2048$. Figure 11 presents the total sum squared error for all six transmissions for the localized and the non-localized approach and for different values of H. We also conducted a similar experiment, where the value of H was set to 10 and we varied the ratio $\frac{k_1}{k_{gl}}$ for different values of k and present the total SSE error in Figure 12. Table 8 also presents the

[12] k_1 is the initial value for the compression ratios k_i of the sensor nodes in the group. While our algorithm dynamically adjusts the values of k_i to reduce the error of the approximation, the ratio $\frac{avg(k_i)}{k_{gl}}$ remains constant.

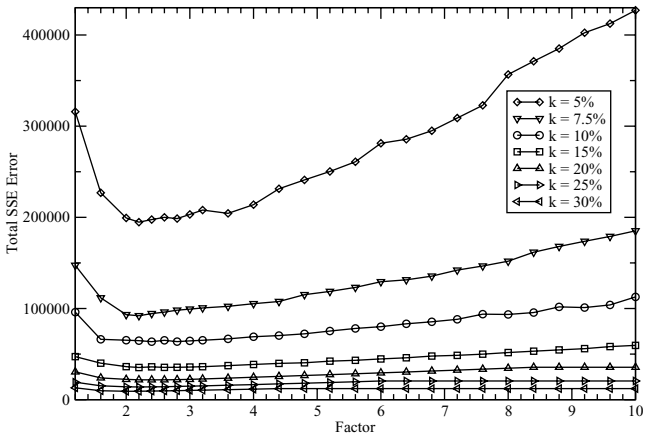

Fig. 12 Selecting the ratio $\frac{k_1}{k_{gl}}$ vs k

errors for the non-localized approach for the second experiment and compares them to the errors of the localized algorithm, when the ratio $\frac{k_1}{k_{gl}}$ is set to 3. Two major observations can be drawn from Figs. 11 and 12 and Table 8:

1. The localized approach can result in a significant reduction in the approximation's error when compared to the non-localized approach. This error reduction is often by a factor more than 5 (and up to 15 in Table 8), and increases with the distance H of the base station from the group leader or with smaller values of k.
2. There is a large range, whose width increases with the value of H and k, of values (i.e., between 2 and 4) for the ratio $\frac{k_1}{k_{gl}}$ for which near optimal results are obtained for the localized algorithm. For very small or large values of this ratio the quality of the approximation degrades, but is often still significantly better than the localized case. We thus suggest always setting the ratio $\frac{k_1}{k_{gl}}$ to the value 3.

To understand why the range of values of the ratio $\frac{k_1}{k_{gl}}$ where near-optimal results are obtained, is increased with H, we need to consider the relative change $\frac{\delta k_{gl}}{k_{gl}}$ when we increase the ratio $\frac{k_1}{k_{gl}}$ from r to $r + \delta r$ (i.e., $\delta r = 1$) for the same value of k. We can then show that:

$$\frac{\delta k_{gl}}{k_{gl}} = \frac{\delta r(S-1)}{(r+\delta r)(S-1)+SH}$$

Table 8 Error comparison of localized vs non-localized algorithm

k %	Localized	Non-localized
5	203,213	2,923,141
7.5	99,342	812,026
10	64,507	303,460
15	35,784	84,907
20	22,278	38,954
25	14,876	23,706
30	10,169	16,901

Table 9 Compression ratios for sensors ($k = 10\%$, $H = 10$)

	Sensor				
Update	1	2	3	4	5
1	14.3	24.0	33.7	33.7	14.3
2	8.5	33.6	29.9	29.9	18.1
3	5.1	39.4	46.8	17.9	10.8
4	3.0	42.9	47.3	20.3	6.4
5	11.4	35.4	38.0	21.8	13.5
6	6.8	40.5	42.0	22.6	8.0

Thus, the relative reduction in the value of k_{gl} decreases with the value of H. This justifies the reduced effect that the $\frac{k_1}{k_{gl}}$ ratio has on the approximation accuracy as H increases.

In Table 9 we present the compression ratios k_i that were assigned to each of the 5 sensor nodes in the group (besides the group leader) for each update operation in the second experiment (where $H = 10$ and we varied k) for a value of $k = 10\%$. This table clearly demonstrates the need for dynamically adjusting the bandwidth for each node. Nodes 1 and 5 seem to always collect measurements that are easy to approximate, while the measurements of node 3 are consistently hard to approximate and, thus, its compression ratio value is higher. On the other hand, notice that initially more bandwidth was assigned to node 4 than in node 2, a trend that was reversed in later invocations of the algorithm. Any algorithm that statically allocates the bandwidth to each sensor would not be able to exploit such changes in the characteristics of the collected data and would result in reduced accuracy in the obtained approximation.

7 Conclusions

We presented a new data compression technique, designed for historical data collected in sensor networks, which however can also be applied in compressing multiple time series in general. Our method splits the recorded series into intervals of variable length and encodes each of them using an artificially constructed *base signal*. The values of the base signal are extracted from the real measurements and maintained dynamically as data changes. Our method easily adapts to different error metrics by simply changing the Regression subroutine used. It can also be modified to provide strict error bounds or a combination of error and space bounds.

In our experiments we used real data sets from a variety of fields (weather, stock and phone call data). Using the sum-squared error and the sum-squared relative error of the approximation, our method significantly outperformed in accuracy approximations obtained by using Wavelets, DCT and Histograms. We also explored the benefits of organizing the nodes in localized groups and found the reduction in the obtained approximation error to often be significant.

A key to our method is the use of the base signal for encoding piece-wise linear correlations among the data values.

We emphasize here that our method does not only apply to linear data sets; in fact none of the data we used are linear in nature. Linearity is exploited when encoding the correlations of the data values and the base signal. An interesting question is to what extent non-linear encodings over the base signal values would benefit the approximations obtained without sacrificing complexity. We plan to investigate this path in the future.

References

1. Ahmed, N., Natarakan, T., Rao, K.R.: Discrete cosine transform. IEEE Transactions on Computers, C-23 (1974)
2. Ailamaki, A., Faloutsos, C., Fischbeck, P.S., Small, M.J., VanBriesen, J.: An environmental sensor network to determine drinking water quality and security. SIGMOD Record **32**(4), 47–52 (2003)
3. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A Survey on Sensor Networks. IEEE Communications Magazine, **40**(8), 102–114 (2002)
4. Bawa, M., Garcia,-Molina, H., Gionis, A., Motwani, R.: Estimating Aggregates on a Peer-to-Peer Network. Technical report, Stanford, 2003.
5. Belussi, A., Faloutsos, C.: Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. In: Proceedings of the VLDB Conference (1995)
6. Cerpa, A., Estrin, D.: ASCENT: Adaptive self-configuring sensor network topologies. In: Proceedings of INFOCOM(2002)
7. Chakrabarti, K., Garofalakis, M., Rastogi, R., Shim, K.: Approximate query processing using wavelets. In: Proceedings of the VLDB Conference (2000)
8. Chang, J.H., Tassiulas, L.: Energy conserving routing in wireless ad-hoc networks. In: Proceedings of INFOCOM (2000)
9. Chen, J., Dewitt, D.J., Tian, F., Wang, Y.: NiagaraCQ: A scalable continuous query system for internet databases. In: Proceedings of the ACM SIGMOD Conference (2000)
10. Chen, Y., Dong, G., Han, J., Wah, B.W., Wang, J.: Multi-dimensional regression analysis of time-series data streams. In: Proceedings of the VLDB Conference (2002)
11. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Evaluating Probabilistic Queries over Imprecise Data. In: Proceedings of the ACM SIGMOD Conference (2003)
12. Cherniack, M., Franklin, M.J., Zdonik, S.B.: Data management for pervasive computing. In: Proceedings of the VLDB Conference (2001)
13. Considine, J., Li, F., Kollios, G., Byers, J.: Approximate aggregation techniques for sensor databases. In: Proceedings of the ICDE Conference (2004)
14. Deligiannakis, A., Kotidis, Y., Roussopoulos, N.: Compressing historical information in sensor networks. In: Proceedings of the ACM SIGMOD Conference (2004)
15. Deligiannakis, A., Kotidis, Y., Roussopoulos, N.: Hierarchical in-network data aggregation with quality guarantees. In: Proceedings of the EDBT Conference (2004)
16. Deligiannakis, A., Roussopoulos, N.: Extended wavelets for multiple measures. In: Proceedings of the ACM SIGMOD Conference (2003)
17. Demers, A., Gehrke, J., Rajaraman, R., Trigoni, N., Yao, Y.: The cougar project: A Work In Progress Report. SIGMOD Record **32**(4), 53–59 (2003)
18. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J.M., Hong, W.: Model-Driven Data Acquisition in Sensor Networks. In: Proceedings of the VLDB Conference (2004)
19. Estrin, D., Govindan, R., Heidermann, J., Kumar, S.: Next century challenges: scalable coordination in sensor networks. In: Proceedings of MobiCOM (1999)
20. Ganesan, D., Estrin, D., Heidermann, J.: DIMENSIONS: Why do we need a new Data Handling architecture for Sensor Networks? In: Proceedings of HotNets-I (2002)
21. Garofalakis, M., Gibbons, P.B.: Wavelet synopses with error guarantees. In: Proceedings of the ACM SIGMOD Conference (2002)
22. Garofalakis, M., Gibbons, P.B.: Probabilistic Wavelet Synopses. ACM Transactions on Database Systems **29**(1), 43–90 (2004)
23. Gibbons, P.B.: Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports. In: Proceedings of the VLDB Conference (2001)
24. Gibbons, P.B., Matias, Y.: New sampling-based summary statistics for improving Approximate Query Answers. In: Proceedings of the ACM SIGMOD Conference (1998)
25. Gilbert, A.C., Kotidis, Y., Muthukrishnan, S., Strauss, M.: One-Pass Wavelet Decompositions of Data Streams. IEEE Transactions on Knowledge and Data Engineering **15**(3), 541–554 (2003)
26. Heidermann, J., Silva, F., Intanagonwiwat, C., Govindan, R., Estrin, D., Ganesan, D.: Building efficient wireless sensor networks with low-level naming. In: Proceedings of SOSP (2001)
27. Heinzelman, W., Chandrakasan, A., Balakrishnan, H.: Energy-efficient communication protocol for wireless microsensor networks. In: Proceedings of the Hawaii Conference on System Sciences (2000)
28. Hellerstein, J.M., Franklin, M.J., Chandrasekaran, S., Deshpande, A., Hildrum, K., Madden, S., Raman, V., Shah, M.A.: Adaptive Query Processing: Technology in Evolution. IEEE Data Engineering Bulletin **23**(2) (2000)
29. Intanagonwiwat, C., Estrin, D., Govindan, R., Heidermann, J.: Impact of network density on data aggregation in wireless sensor networks. In: Proceedings of ICDCS (2002)
30. Ioannidis, Y.E., Poosala, V.: Histogram-based approximation of set-valued query answers. In: Proceedings of the VLDB Conference (2000)
31. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proceedings of FOCS (2003)
32. Khanna, S., Tan, W.C.: On Computing functions with uncertainty. In: Proceedings of ACM PODS Conference (2001)
33. Korn, F., Labrinidis, A., Kotidis, Y., Faloutsos, C.: Quantifiable Data Mining Using Ratio Rules. VLDB Journal **8**(3–4), 254–266 (2000)
34. Kotidis, Y.: Snapshot Queries: Towards data-centric sensor networks. In: Proceedings of the ICDE Conference (2005)
35. Lee, J., Kim, D., Chung, C.: Multi-dimensional selectivity estimation using compressed histogram information. In: Proceedings of the ACM SIGMOD Conference (1999)
36. Lindsey, S., Raghavendra, C.S.: Pegasis: Power-Efficient Gathering in Sensor Information Systems. In: Proceedings of the IEEE Aerospace Conference (2002)
37. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: TAG: A tiny aggregation service for ad hoc sensor networks. In: Proceedings of the OSDI Conference (2002)
38. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: The design of an acquisitional query processor for sensor networks. In: Proceedings of the ACM SIGMOD Conference (2003)
39. Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D., Anderson, J.: Wireless Sensor Networks for Habitat Monitoring. ACM WSNA Workshop (2002)
40. Matias, Y., Vitter, J.S., Wang, M.: Wavelet-based histograms for selectivity estimation. In: Proceedings of the ACM SIGMOD Conference (1998)
41. Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G., Olston, C., Rosenstein, J., Varma, R.: Query processing, resource management, and approximation in a data stream management system. In: Proceedings of CIDR (2003)
42. Olston, C., Jiang, J., Widom, J.: Adaptive filters for continuous queries over distributed data streams. In: Proceedings of the ACM SIGMOD Conference (2003)
43. Olston, C., Widom, J.: Offering a precision-performance tradeoff for aggregation queries over replicated data. In: Proceedings of the VLDB Conference (2000)

44. Poosala, V., Ioannidis, Y.E.: Selectivity estimation without the attribute value independence assumption. In: Proceedings of the VLDB Conference (1997)
45. Poosala, V., Ioannidis, Y.E., Haas, P.J., Shekita, E.J.: Improved Histograms for Selectivity Estimation of Range Predicates. In: Proceedings of the ACM SIGMOD Conference (1996)
46. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C. Cambridge University Press, 2nd edition edition (1992)
47. Qiao, L., Agrawal, D., Abbadi, A.E.: RHist: Adaptive Summarization over Continuous Data Streams. In: Proceedings of CIKM (2002)
48. Sharaf, A., Beaver, J., Labrinidis, A., Chrysanthis, P.: Balancing Energy Efficiency and Quality of Aggregate Data in Sensor Networks. VLDB Journal **13**(4), 384–403 (2004)
49. Shih, E., Cho, S.-H., Ickes, N.: Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. In: Proceedings of MOBICOM (2001)
50. Singh, S., Woo, M., Raghavendra, C.S.: Power-Aware Routing in Mobile Ad Hoc Networks. In: ACM/IEEE International Conference on Mobile Computing and Networking (1998)
51. Stollnitz, E.J., DeRose, T.D., Salesin, D.H.: Wavelets for Computer Graphics–Theory and Applications. Morgan Kaufmann Publishers, Inc., San Francisco, CA (1996)
52. Tan, H.O., Korpeoglu, I.: Power Efficient Data Gathering and Aggregation in Wireless Sensor Networks. SIGMOD Record **32**(4), (2003)
53. Thaper, N., Guha, S., Indyk, P., Koudas, N.: Dynamic Multidimensional Histograms. In: Proceedings of the ACM SIGMOD Conference (2002)
54. Viglas, S.D., Naughton, J.F.: Rate-based query optimization for streaming information sources. In: Proceedings of the ACM SIGMOD Conference (2002)
55. Vitter, J.S., Wang, M.: Approximate computation of multidimensional aggregates of sparse data using wavelets. In: Proceedings of the ACM SIGMOD Conference (1999)
56. Warneke, B., Last, M., Liebowitz, B., Pister, K.S.J.: Smart Dust: Communicating with a Cubic-Millimeter Computer. IEEE Computer **34**(1), 44–51 (2001)
57. Yao, Y., Gehrke, J.: The Cougar Approach to In-Network Query Processing in Sensor Networks. SIGMOD Record **31**(3), 9–18 (2002)
58. Younis, O., Fahmy, S.: HEED: A hybrid, energy-efficient, distributed clustering approach for Ad Hoc sensor networks. IEEE Transactions on Mobile Computing **3**(4) (2004)
59. Zdonik, S.B., Stonebraker, M., Cherniack, M., Cetintemel, U., Balazinska, M., Balakrishnan, H.: The Aurora and Medusa Projects. IEEE Data Engineering Bulletin (2003)
60. Zeinalipour-Yazti, D., Neema, S., Gunopulos, D., Kalogeraki, V., Najjar, W.: Data acquision in sensor networks with large memories. In: Proceedings of the IEEE International Workshop on Networking Meets Databases (2005)