ABSTRACT

Title of dissertation: ACCURATE DATA APPROXIMATION IN CONSTRAINED ENVIRONMENTS

Antonios Deligiannakis,
Doctor of Philosophy, 2005

Dissertation directed by: Professor Nick Roussopoulos
Department of Computer Science

Several data reduction techniques have been proposed recently as methods for providing fast and fairly accurate answers to complex queries over large quantities of data. Their use has been widespread, due to the multiple benefits that they may offer in several *constrained* environments and applications. Compressed data representations require less space to store, less bandwidth to communicate and can provide, due to their size, very fast response times to queries. Sensor networks represent a typical constrained environment, due to the limited processing, storage and battery capabilities of the sensor nodes.

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 data reduction techniques: *aggregation* and *approximation*.

In this dissertation we propose novel data reduction techniques for the transmission of measurements collected in sensor network environments. We first study the problem of summarizing multi-valued data feeds generated at a single sensor node, a step necessary for the transmission of large amounts of historical information collected at the node. The transmission of these measurements may either be periodic (i.e., when a certain amount of measurements has been collected), or in response to a query from the base station. We then also consider the approximate evaluation of aggregate *continuous* queries. A continuous query is a query that runs continuously until explicitly terminated by the user. These queries can be used to obtain a live-estimate of some (aggregated) quantity, such as the total number of moving objects detected by the sensors.

DEDICATION

To my beloved parents

ACKNOWLEDGMENTS

I would like to thank all the people whose help and support over my graduate studies has made this thesis possible and who have made my graduate experience an unforgettable one.

First and foremost I would like to thank my advisor, Professor Nick Roussopoulos, for his invaluable guidance over the past six years. As my advisor, he gave me the opportunity to work on challenging research problems and through his witty remarks and suggestions has helped me focus and improve my work. Without him none of this work would have been possible. Perhaps more importantly, on a personal level he has always expressed intense interest and support for me in difficult times, something I neither expected, nor could I have ever imagined when I started my graduate experience. I could not thank him enough for this.

I would also like to thank the members of the committee, professor Sudarshan Chawathe, professor Lise Getoor, professor Amol Deshpande and professor Virgil Gligor for agreeing to serve on my thesis committee and for sparing their invaluable time reviewing the manuscript and providing comments for the thesis. I would also like to thank professor Louiqa Raschid and professor Samir Khuller for helpful suggestions over the years.

I would also like to express my deep gratitude for my advisor in my undergraduate studies, professor Timos Sellis. He has maintained close contact with me over

the years and his advice and help are something that I am really grateful for.

Over the years I have had the pleasure to cooperate with several amazing and talented colleagues and friends. I have had the pleasure to work with Yannis Sismanis during my initial graduate years on the Dwarf system for computing, storing and indexing data cubes. Yannis is an exceptional researcher and a very good friend. I will never forget the great times we had going out and playing chess during our graduate studies. Yannis Kotidis has been like a co-advisor to me, not only by cooperating with me on a significant part of the work in this thesis, but also by providing his immensely helpful advice on several professional issues. His help and friendship over the years have been invaluable. Minos Garofalakis is another exceptional researcher and individual that I have recently had the pleasure to cooperate with on several wavelet-based algorithms, and who has also provided me with exceptional advice and guidance for my future career.

I would also like to thank many of my friends who have given me nothing but great memories over the past six years. Dimitrios Tsoumakos has been an amazing roommate and friend. I would like to express my gratitude for the fun times that we had as roommates and for his helpful suggestions and discussions involving my research. Alexandros Labrinidis provided me invaluable advice during my first years as a graduate student and was a great roommate as well. I have also had the pleasure to be a roommate of Konstantinos Spiliopoulos and Damianos Karakos, who have provided me with amazing memories from the time that we lived together. I was very lucky to have Konstantinos Bitsakos study in Maryland as well, as Konstantinos has been a great friend for more than 11 years and his friendship is invaluable to me. I

would also like to thank my dear friends Nick Vasiloglou, Dimitris Axiotis, Michael Romesis, Panos Kontovazenitis, Nick Zygouras, Spyros Triantafyllis, Domna Gitakou, Eleni Kotidi, Stella Gressis, Maria Striki, Ioanna Soulioti, Nickolas Stathatos, Vassilis Anagnostopoulos, Andreas Tsirikos, Manolis Hourdakis and Thanos Chryssis for their friendship, support and great memories that they have provided me over these years.

I was very fortunate to work in the environment of the Maryland Database group and interact with amazing people like Laura Bright, Ugur Cetintemel, Manuel Rodriguez-Martinez, Tolga Urhan, Demet Aksoy, Mehmet Altinel and Fatma Ozcan. My interaction with them has been very fruitful and enjoyable.

Finally, I owe my deepest thanks and gratitude to my beloved parents and my brother. My parents have made tremendous sacrifices to provide me with all the necessary tools to study. Their amazing encouragement and support, along with my brother's, has been the main motivational force for my Ph.D. This thesis is dedicated to them.

It is impossible to remember all, and I apologize to those I have inadvertently left out.

Thank you all!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 The Need for Data Reduction Techniques

Several data reduction techniques have been proposed recently as methods for providing fast and fairly accurate answers to complex queries over large quantities of data. The most popular approximate processing techniques include histograms, random sampling and wavelets. Their use has been widespread, due to the multiple benefits that they may offer in several *constrained* environments and applications. The constraints that may necessitate the construction of *data synopses* are numerous, and vary in each case, based on the targeted application. First of all, when the amount of stored data is too large to store and process, data reduction techniques present a natural choice for data compression. Moreover, when fast response times are desired without requiring 100% accuracy, such as in data mining and approximate query processing applications, the constructed *data synopses* can provide significantly faster answers than exact processing techniques since, due to their small size, they can be maintained in main memory. Finally, when data is communicated between different locations and either the available bandwidth or the energy of the nodes is limited, as is typically the case in sensor network applications, transmitting a compressed data representation is the only viable solution.

Depending on the nature and the characteristics of the collected data, different data reduction techniques may be deemed as more appropriate and effective. Moreover, even when considering the same data set within the context of different applications, the technique of choice may vary based on its processing and memory requirements and the corresponding capabilities of the application at hand. For example, while a sensor node with very limited capabilities may elect to approximate its measurements using a sampling algorithm, a more powerful sensor may elect to utilize a wavelet-based technique for the same task, in order to achieve a more accurate data synopsis. Furthermore, different criteria may be used for the selection and tuning of the used techniques. In most cases the user/application specifies the desired size of the reduced data representation and the goal is to maximize the accuracy (or, equivalently, minimize the error) of the approximate reconstructed data. In other cases, a desired accuracy of the approximate data is specified, and the goal is to minimize the size of the reduced representation that achieves the specified accuracy guarantees.

In this dissertation we develop several data reduction techniques that can be applied on a variety of applications in constrained environments. However, we focus our discussion primarily on the area of sensor network applications, and later detail other domains where our techniques are also applicable.

The deployment of densely distributed sensor networks has been made feasible, from both a technological as well as an economical point of view, by recent advances in wireless technologies and microelectronics. These networks are used in a variety of monitoring applications such as military surveillance, habitat monitoring, location tracking and inventory management. Each sensor node may be equipped with several

sensing elements, such as microphones, accelerometers and temperature sensors that allow it to gather low-level measurements of its surroundings. These measurements can then be processed locally and transmitted, either in chunks, or in a continuous fashion, to a *base station* for further analysis. A base station may represent any node of the network with increased storage, battery and processing capabilities.

Besides its sensing elements, each sensor node consists of several parts, including a processing unit (cpu), a memory component, a battery used to power the sensor and a radio used by the sensor to communicate with its surrounding nodes. The characteristics of the used sensor nodes depend on the nature of the application at hand. 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, the Stargate nodes constructed by Crossbow [cro] contain a 400MHz X-Scale Intel processor (model PXA255) and can support up to 256 MB of FLASH memory, while being powered by a Li-Ion battery. On the other hand, the current sensor nodes that are most frequently used have significantly more limited capabilities. For example, the Berkeley Mica2 motes utilize a 7.3828 MHz processor and can support up to 128 KB of FLASH and 32 KB of EEPROM memory, while being powered by 2 AA batteries. As the processing and storage capabilities of sensor nodes tend to follow Moore's Law,[1] we expect that in the near future significantly more powerful sensor nodes will be available and more frequently used, due to their increased processing, storage and sensing capabilities. Our belief is also powered by ongoing efforts by the industry to construct significantly more powerful and energy-efficient sensor motes,

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

such as the Intel motes.

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 may be 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 sensor.

In this dissertation we propose novel data reduction techniques for the transmission of measurements collected in sensor network environments. We first study the problem of summarizing multi-valued data feeds generated at a single sensor node, a step necessary for the transmission of large amounts of historical information collected at the node. The transmission of these measurements may either be periodic (i.e., when a certain amount of measurements has been collected), or in response to a query from the base station. We then also consider the approximate evaluation of

aggregate *continuous* queries. A continuous query is a query that runs continuously until explicitly terminated by the user. These queries can be used to obtain a live-estimate of some (aggregated) quantity, such as the total number of moving objects detected by the sensors.

While our algorithms for the evaluation of continuous queries can be applied, due to their simplicity, to all current models of sensor nodes, the processing requirements of our data approximation algorithms impose some constraints on the models of sensor nodes that can be used to execute them. Our wavelet-based techniques (Section 1.2.2) exhibit near-linear running times and are thus suited even for low-end sensor nodes, such as the often used MICA2 nodes. On the other hand, our Self-Based Regression algorithm (Section 1.2.1) requires sensor nodes with significantly more powerful processors. In Section 3.5.3 a 300 MHz processor required about 9.7 seconds to compress 10,240 data values. We expect that most sensors with processors clocked around 80 MHz (or higher) will be able to easily execute this algorithm, since the amount of transmitted data in most sensor network applications will often be smaller. However, the organization of sensor nodes in localized groups discussed in Section 3.4.4 can help further reduce the processing requirements in low-end sensor nodes.

1.2 Compressing Historical Information

We study the problem of summarizing multi-valued data feeds generated at a single sensor node for the periodic transmission of its measurements. We explore two dif-

ferent approaches for this application: (i) A model-driven data reduction technique appropriate for applications where each sensor monitors several quantities over specific data intervals; and (ii) Wavelet-based techniques applicable to multi-dimensional data sets, with near-linear running times. The choice among these two approaches may depend on the dimensionality of the data and the processing capabilities of the sensor node. We need to mention that both of our approaches only consider as their constraint the desired size of the compressed data representation. While the energy consumed by the sensor during the execution of the algorithm is also an important factor, transmission is the dominant source of energy drain in sensor networks.

1.2.1 Compressing Multiple Time Series

Our first approach builds on the observation that the values of the collected measurements from sensor nodes may exhibit similar patterns over time, or that different measurements are naturally correlated, as is the case between pressure and humidity in weather monitoring applications. At the core of our Self-Based Regression (*SBR*) approximation algorithm lies the notion of a *base signal*, a set of values from the collected measurements that capture prominent features of the data. Following the construction of the *base signal*, the collected data is partitioned into intervals that can be efficiently approximated as linear projections of some part of the *base signal*. We provide techniques for (i) constructing the base signal; (ii) approximating the recorded measurements by exploring piece-wise correlations amongst them and the base signal; and (iii) dynamically updating the base signal to capture new data

trends in subsequent transmissions. Our methods are easily adaptable to different error metrics with a simple change in the used subroutine that performs the linear regression, and have been shown to significantly outperform current data reduction techniques in a variety of data sets.

Sensors networks are built with the premise that nodes should collaborate to perform the task at-hand. Thus, designing localized algorithms is of utmost importance in these networks [EGHK99, Kot05]. In our work we have been able to extend our framework to be used in a localized manner, where groups of sensors that lie within a small distance from each other cooperate to produce a more accurate approximate representation of their collected data. Under the localized mode of operation the sensor nodes transmit their compressed data feeds to the group leader, which in turn applies our compression 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 desired error metric. This localized framework allows the group leader to explore correlations between measurements of different sensors to achieve a significantly more accurate approximation for the same overall *compression ratio*. Such correlations may be either due to the small variations of the monitored physical quantities because of the proximity of the sensors (for instance when obtaining temperature readings), or due to the nature of the measurements (i.e., voice levels tend to fall gradually with the radius from the source).

While the above approach is able to provide very accurate data synopses, its running time, which exhibits a $O(n^{1.5})$ dependency on the size n of the collected data,

may limit its applicability to several current sensor models with restricted processing and memory capabilities. On the other hand, our experimental study demonstrates that the base signal is rarely populated, or in very small parts, after the initial transmissions, since it is already of good quality. This allows us to shortcut some parts of the algorithm in constrained environments and reduce the complexity of the algorithm to only a linear dependency to the size of the processed data. Moreover, since some powerful sensor nodes have already been developed (i.e., [cro]), and due to the rapid increase of the processing and storage capabilities, at a rate similar to Moore's Law [L07], of the sensor nodes, we expect that in the following years a large amount of the available sensors will be able to compress even large amounts of collected data using our techniques.

1.2.2 Extended Wavelets for Multi-Measure Data Sets

For lower-end processors and for applications where the collected data is multi-dimensional we need to develop additional techniques for the efficient and accurate compression of the sensor's collected data. While such multi-dimensional data sets are more common in other application environments (see Section 1.4) they may also arise in several sensor network applications. For example, if the sensor is attached to a moving object, it may additionally record for each set of obtained measurements not only the time, but also its location. Moreover, each sensor is a device that communicates with other nodes within its vicinity. Collecting statistics on the input (output) traffic, such as the types and number of packets received by (transmitted to) other

nodes may help reveal not only areas of congestion and high packet loss, but even nodes that are either malfunctioned, or even behaving in a malicious way.

In our work we focused our attention to the wavelet decomposition technique, since the work in [CGRS00, MVW98, VW99] demonstrated that wavelets can achieve increased accuracy to queries over other data reduction techniques, such as histograms and random sampling. In this dissertation we propose a novel approach for effectively adapting wavelet-based data reduction methods to multi-measure data sets through the use of *extended wavelet coefficients*. Briefly, an extended wavelet coefficient can store multiple coefficient values for different – but not necessarily all – measures. The end result is a flexible, space-efficient storage scheme that can eliminate the disadvantages of prior algorithms. We then consider the problem of constructing effective extended wavelet coefficient synopses (under a given storage constraint) optimized for the (1) *weighted sum-squared error*, and (2) *relative error* in the approximate data reconstruction. Our synopsis-construction problems are natural generalizations of the corresponding problems for conventional (i.e., L_2-error) wavelet synopses [VW99, CGRS01] and probabilistic (i.e., relative-error) wavelet synopses [GG04] for the *single-measure* case. We demonstrate that, in the presence of multiple measure, choosing an effective subset of extended wavelet coefficients gives rise to difficult optimization problems that are significantly more complex than their single-measure counterparts. This is primarily due our more involved extended-coefficient storage format that forces non-trivial dependencies between thresholding decisions made across different measures. We propose optimal solutions based on novel algorithmic formulations that employ *Dynamic-Programming* (DP) ideas. Given the high time and space

complexities of our exact DP schemes, we also introduce fast, greedy approximation algorithms (based on the idea of *marginal error gains*) that produce near-optimal solutions. To the best of our knowledge, our work represents the first principled, methodical study of effective wavelet-based data reduction techniques for multi-measure data sets.

1.3 Evaluating Approximate Continuous Queries

In sensor networks, processing is often driven by designated nodes that monitor the behavior of either the entire, or parts of the network. This monitoring is typically performed by issuing queries, which are propagated through the network, over the data collected by the sensor nodes. The output of the queries is then collected by the monitoring node(s) for further processing. While typical database queries are executed once, queries in monitoring applications are long-running and executed over a specified period, or until explicitly being terminated. These types of queries are known as *continuous queries* [CDTW00, TGNO92].

In these types of monitoring applications, data influencing the result of the query is transmitted almost continuously, since an on-line estimation of the query result is desired. Since under this scenario each node will typically transmit a single value, the data reduction techniques that we have proposed so far are not applicable. Instead, protocols that reduce the number of messages that need to be transmitted are needed.

In our work we have developed new techniques for data dissemination in sen-

sor networks when the monitoring application is willing to tolerate a specified error threshold. Two scenarios are explored, where the monitoring application specifies either the maximum error that it is willing to tolerate, or the desired average bandwidth consumption in the network. Our techniques operate by placing error filters at each sensor node participating in the query evaluation, necessitating the transmission of its data only in certain cases. In order to adjust to potentially different characteristics of the collected data, our algorithms are adaptive and periodically adjust these error filters by considering the potential benefit of increasing the error threshold at a sensor node, which is equivalent to the amount of messages that we expect to save by allocating more resources to the node. The result of using this gain-based approach are two robust algorithms that are able to identify volatile data sources and eliminate them from consideration. Moreover, we have introduced the *r*esidual mode of operation, during which a parent node may eliminate messages from its children nodes in the aggregation tree when the cumulative change from these sensor nodes is small. Finally, unlike previous algorithms [OJW03, OW02] for the same application over non-hierarchical environments, our algorithms operate with only local knowledge, where each node simply considers statistics from its children nodes in the aggregation tree. This allows for more flexibility in designing adaptive algorithms and is a more realistic assumption, especially for sensors nodes with very limited capabilities [MFHH02].

1.4 Additional Potential Applications

The use of data reduction techniques has been widely studied for the area of Decision Support Systems (*DSS*), which require processing huge quantities of data to produce exact answers to posed queries. Due to the sheer size of the processed data, queries in *DSS* systems are typically slow. However, many situations arise when an exact answer to a query is not necessary, and the user would be more satisfied by getting a fast and fairly accurate answer to his query, perhaps with some error guarantees, than by waiting for a long time to receive an answer accurate up to the last digit. This is often the case in OLAP applications, where the user first poses general queries while searching for interesting trends on the data set, and then drills down to areas of interest.

While previous work[CGRS00, MVW98, VW99] demonstrated the applicability of wavelets in such applications, little emphasis had been placed on applying wavelets to data sets containing multiple measures. Such data sets are very common in many database applications. For example, a sales database could contain information on the number of items sold, the revenues and profits for each product, and costs associated with each product, such as the production, shipping and storage costs. Moreover, in order to be able to answer types of queries other than Range-Sum queries, it would be necessary to also store some auxiliary measures. For example, if it is desirable to answer Average queries, then the count for each combination of dimension values also needs to be stored. Our work can be directly applied to these types of applications.

While our SBR algorithm is motivated by applications on sensor networks, the

same technique can also be used in other applications for the lossy compression of multiple time series. For instance the phone-call and stock data sets that we use in our experiments are not directly related to sensor networks. In domains where significantly more powerful processors can be used than in sensor network applications, some of the shortcuts introduced in this dissertation (for example, a larger number of candidate intervals for inclusion in the base signal may be considered) may be relaxed in order to achieve even better approximation.

Chapter 2

Related Work

2.1 Sensor Networks

In recent years there has been a flurry of research in the area of sensor networks. Several recent proposals, such as COUGAR [YG02] and TinyDB [MFHH02], have been issued on infusing database primitives on sensor networks. A declarative query language like SQL provides far greater flexibility than hand-coded programs that are pre-installed at the sensor nodes [MFHH03]. In the database community there is ongoing effort in understanding how embedded database systems can be used to provide energy-based query optimization [MFHH03, YG02].

In-network data aggregation is investigated in [DKR04, IEGH02, MFHH02, YG02, SBLC04, DGR+03, GEH02]. 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 reducing substantially the number of messages in the network. Data transmission protocols also need to be developed for the data aggregation process. In [MFHH02], nodes of the aggregation tree carefully synchronize the periods when they transmit data. The idea is to subdivide each epoch into intervals and have parent nodes in the tree listen for messages from their children during pre-defined time-slots. This allows the nodes

to power-down their radios when not necessary and reduce energy and bandwidth consumption. Another notable method for synchronizing the transmission periods of nodes is the recently proposed wave scheduling approach of [DGR+03]. Our algorithms for the approximate evaluation of continuous aggregate queries should be implemented on top of a protocol like TAG [MFHH02] to obtain their maximum benefits. Decentralized algorithms for aggregate computation have also been proposed [BGMGM03, KDG03].

Recent proposals for combining data modeling with data acquisition can also help in reducing the cost of aggregation [DGM+04, Kot05]. Some additional important issues addressed in recent work include network self-configuration [CE02, Kot05], discovery [EGHK99, HSI+01] and computation of energy-efficient data routing paths [CT00, HCB00, LR02, MFHH02, SWR98, TK03]. The techniques developed in the above work are complementary to our work, since while the above techniques help determine energy-efficient aggregation trees, our algorithms further reduce the amount of information flowing in the network. In [CLKB04], a framework for compensating for packet loss and node failures during query evaluation is proposed. In the database community additional issues such as data modeling and acquisition have been recently addressed [DGM+04, Kot05]. Distributed storage management is another topic that brings together the networking and database communities [DGR+03, GGC03, RKY+02].

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 [AFF+03, CFZ01]. In this dissertation we propose approximation techniques 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. In this scenario 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 [CDTW00, HFC+00, MWA+03, VN02, ZSC+03]. Olston et al. in [OLW01, OW00, OW02] investigated the tradeoffs between precision and performance in cached and replicated data. In [OW02] the emphasis is on determining when cached objects of remote data sources should be refreshed in order to minimize the average divergence of the cached objects given a server-size bandwidth constraint. The used divergence function can either represent the staleness or the update lag of the cached object, or the value deviation between the object's cached and current values. The work in [OJW03] also discusses extensions for the execution of multiple concurrent continuous queries. Several of these ideas can also be combined with our algorithms. The problem of minimizing the number of messages exchanged in the network for a given error constraint has also been studied recently in [SBLC04]. In [SBLC04] the authors suggest using a uniform distribution of the error, while our algorithms distribute the error based on local statistics collected on a node. On the other hand, earlier work in distributed constraint checking [BGM92, SS90] cannot be

directly applied in our setting, because of the different communication model and the limited resources at the sensors.

An online algorithm for minimizing the update cost while the query can be answered within an error bound is presented in [KT01]. The evaluation of probabilistic queries over imprecise data was studied in [CKP03, CP03]. Extending this work to hierarchical topologies, such as the ones studied in our work, is an open research topic. Finally, [BGMGM03, KDG03] investigate decentralized algorithms for aggregate computations with applications in P2P and sensor networks.

2.2 Data Reduction Techniques

The main data reduction mechanisms studied so far include histograms, random sampling and wavelets. These techniques are mainly targeted towards low to medium data dimensionalities, as their accuracy has been shown to decrease for high-dimensional data sets. For high-dimensional data, techniques that try to achieve dimensionality reduction by identifying dependencies among the dimensions seem to be more promising. Examples include the work in [GTK01], using probabilistic relational models, and the work in [DGR01], where the goal is to construct accurate high-dimensional histograms by employing statistical interaction models to identify and exploit dependency patterns in the data.

Histograms are the most extensively studied data reduction technique with wide use in query optimizers to estimate the selectivity of queries, and recently in tools for providing fast approximate answers to queries [IP00, PG99]. A classification of the

types of histograms proposed in literature is presented in [PI97]. The main challenge for histograms is to be able to capture the correlation among different attributes in high-dimensional data sets. Recent work in [TGIK02] addresses the problem of constructing accurate high-dimensional histograms with the use of sketching techniques, which were first introduced in [AMS96].

Random Sampling is based on the idea that a small random sample of the data often represents well the entire data set. The result to an aggregate query is given by appropriately scaling the result obtained by using the random sample. Random sampling possesses several desirable characteristics as a reduction method: the estimator for answering count, sum and average aggregate queries is unbiased, confidence intervals for the answer can be given, construction and update operations are easy and have low overhead, and the method naturally extends to multiple dimensions and measures. In [Vit85] the reservoir sampling algorithm was introduced, which can be used to create and maintain a random sample of a fixed size with very low overhead. This method was further refined in [Gib01] for data sets containing duplicate tuples.

Haar wavelets are a mathematical tool for the hierarchical decomposition of functions with several successful applications in signal and image processing [JS94, SDS96]. A number of recent studies have also demonstrated the effectiveness of the Haar wavelet decomposition as a data-reduction tool for database problems, including selectivity estimation [MVW98] and approximate query processing over massive relational tables [CGRS00, GG02, VW99] and data streams [GKMS01, MVW00]. Briefly, the key idea is to apply the decomposition process over an input data set along with a thresholding procedure in order to obtain a compact data synopsis com-

prising of a selected small set of *Haar wavelet coefficients*. The results of several research studies [CGRS00, GG02, GG04, MVW98, SS02, VW99] have demonstrated that fast and accurate approximate query processing engines can be designed to operate solely over such compact *wavelet synopses*. Furthermore, even though the Haar wavelet decomposition was originally designed with the objective of minimizing the overall mean-squared error (i.e., the *L_2-norm error*) in the data approximation, recent work [GG04, GK04] has also demonstrated their use in optimizing *relative-error* metrics. Relative errors are arguably the most important metrics for approximate query answers and can also enable meaningful, non-trivial *error guarantees* for reconstructed values (by bounding the *maximum relative error* in the approximate reconstruction of individual data values [GG04, GK04]).

The Discrete Cosine Transform (DCT) [ANR74] constitutes the basis of the *mpeg* encoding algorithm and has also been used to construct compressed multi-dimensional histograms [LKC99]. Linear regression has been recently used in [CDH$^+$02] for on-line multidimensional analysis of data streams.

The use of a dictionary is a standard technique in data compression algorithms (for example in gzip). As an abstraction, the base signal that we construct in one of our SBR algorithm (Chapter 3) 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.

Chapter 3

Compression of Multiple Time Series

3.1 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. These networks are used in a variety of monitoring applications such as military surveillance, habitat monitoring, location tracking and inventory management. Each sensor node may be 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.

In this chapter we study the problem of disseminating historical information in sensor networks. We first look at the problem of summarizing multi-valued data feeds generated at a single sensor node. Our techniques build on the observation that the values of the collected measurements exhibit similar patterns over time, or that different measurements are naturally correlated, as is the case between pressure and humidity in weather monitoring applications. At the core of our approximation lies the notion of a *base signal*, a set of values from the collected measurements that can be used to linearly approximate other parts of the data. Following the construction of the *base signal*, the collected data is partitioned into intervals that can be efficiently approximated as linear projections of some part of the *base signal*. As we will show in this chapter, our techniques provide:

- *Increased accuracy and robustness when compared to other approximation techniques:* Our algorithm feeds from intrinsic redundancy in the collected measurements (like many data reduction techniques), but has full control over the data model used to exploit these redundancies (which values to insert into the base

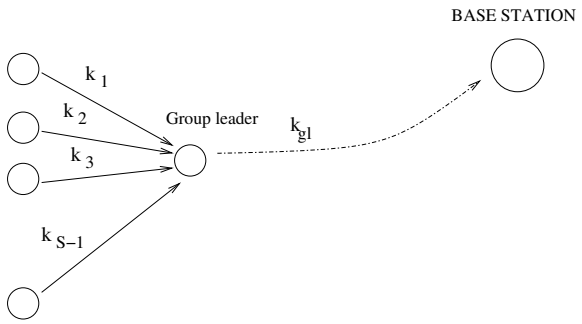


Figure 3.1: Dissemination using localized groups: nodes form groups and elect a group leader. Using a localized algorithm each node in the group obtains a compression ration k_i so that the target overall compression ratio is achieved, while the error of the approximation is minimized.

signal), the amount of space allocated for this data model and the number of coefficients that describe the projections of the data values on this data model. Due to this fact, our algorithm produced up to 27 and 1000 times smaller errors than the next best competitive method for the sum squared and the sum squared relative error metrics, respectively, Moreover, due to its fall-back mechanism to linear regression, as we will later explain, it performs at least as good as linear regression, but in practice is significantly more accurate.

- *Adaptability to different error metrics:* Our algorithm can be adapted with only minor modifications, *which do not alter its time complexity*, to minimize different error metrics, such as the sum squared error, sum squared relative error, and maximum error of the approximation. We further discuss extensions when the application requires strict error bounds or a combination of error and space bounds.

Sensors networks are built with the premise that nodes should collaborate to perform the task at-hand. Thus, designing localized algorithms is of utmost im-

portance in these networks [EGHK99, Kot05]. We have been able to extend our framework to be used in a localized manner, where groups of sensors that lie within a small distance from each other cooperate to produce a more accurate approximate representation of their collected data. Under the localized mode of operation the sensor nodes transmit their compressed data feeds to the group leader, which in turn applies our compression 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 process is illustrated in Figure 3.1. As will be explained, our algorithm is applied on the compressed values directly; there is no need for the group leader to "uncompress" the feeds received by the other nodes in its group. This localized framework allows the group leader to explore correlations between measurements of different sensors to achieve a significantly more accurate approximation for the same overall *compression ratio*. Such correlations may be either due to the small variations of the monitored physical quantities because of the proximity of the sensors (for instance when obtaining temperature readings), or due to the nature of the measurements (i.e. voice levels tend to fall gradually with the radius from the source).

3.2 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

chapter.

3.2.1 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 [SCI+01].

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 [SCI+01] 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.[1] The main source of energy consumption in a node is the data transmission process. There are several reasons for this:

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

1. The energy drain during transmission is much larger than the consumption during processing [EGHK99]. As an example, on a Berkeley MICA Mote sending one bit of data costs as much energy as 1,000 CPU instructions [MFHH03]. Faster processors typically achieve lower power consumptions per CPU instruction, making the above ratio even larger.

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 [MFHH02], 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.

3.2.2 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

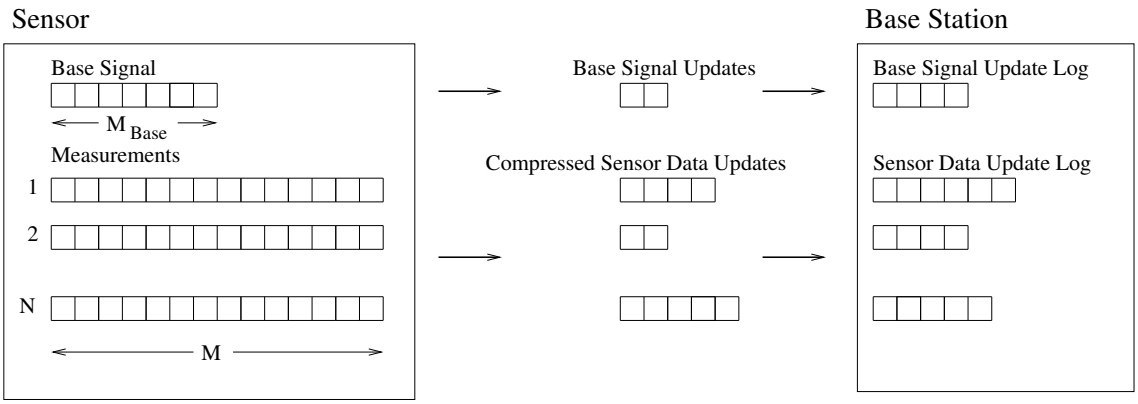


Figure 3.2: Transfer of approximate data values and of the base signal from each sensor to the base station

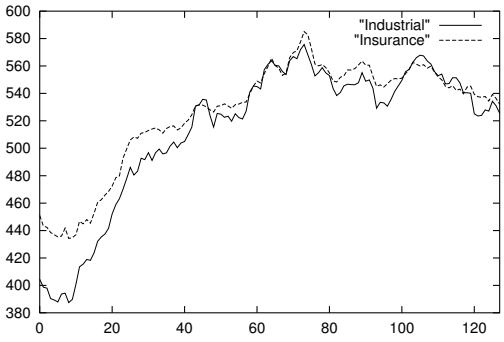


Figure 3.3: Example of two correlated signals (Stock Market)

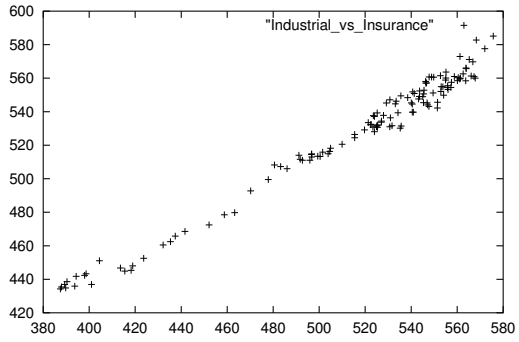


Figure 3.4: XY scatter plot of Industrial (X axis) vs Insurance (Y axis)

enough data is collected and then forward it to the base station [SCI+01]. This form of batch processing allows them to power-down their radio transmitter and prolong their lifetime in a way analogous to [MFHH02].

Within a sensor, the recorded data is depicted in a two dimensional array 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.[2]

As more measurements are obtained, the sensor's memory buffers become full. At this point the latest NxM 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 Figure 3.2.

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 of prominent features 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 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.

[2]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.

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

3.3 The SBR Framework

We now describe our framework in more detail. We start with a motivational example that demonstrates the intuition behind our techniques. Subsection 3.3.2 presents the primitive operations required by our framework while the *SBR* algorithm is presented

Configuration Parameters	
N	Number of input signals
M	Measurements per input signal
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 in current transmission
Ins	Number of base intervals actually inserted in the current transmission

Table 3.1: Configuration, input and derived parameters of our algorithms

in subsection 3.3.3. Table 3.1 contains a brief description of the parameters used in our algorithms.

3.3.1 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 Figure 3.3 we plot the average Industrial and Insurance indexes from the New York stock market for 128 consecutive days.[3] Both signals show similar trends; i.e., they go up and down together. Figure 3.4 depicts a XY scatter plot 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

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

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 Figure 3.3 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 Figure 3.4 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.

3.3.2 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, i.e. using at least four values per time series.

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.[4] 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 Figure 3.5 is at the core of our method. This function pairs a segment of the base signal between values $[start\_x, start\_x + length)$

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

procedure Regression($\vec{X}$, $\vec{Y}$, $start\_x$, $start\_y$, $length$)
Input: Base signal $\vec{X}$; signal $\vec{Y}$; starting points
$start\_x$ and $start\_y$ of the paired intervals; interval length $length$.
Output: Regression parameters a, b, error err of the approximation.
begin
1. //Compute Regression Parameters
2. $sum\_x = \sum_{0 \le i < length} X[i + start\_x]$
3. $sum\_y = \sum_{0 \le i < length} Y[i + start\_y]$
4. $sum\_xy = \sum_{0 \le i < length} X[i + start\_x]Y[i + start\_y]$
5. $sum\_x2 = \sum_{0 \le i < length} X[i + start\_x]^2$
6. $a = \frac{length \times sum\_x\_y - sum\_x \times sum\_y}{length \times sum\_x2 - sum\_x \times sum\_x}$
7. $b = \frac{sum\_y - a \times sum\_x}{length}$
8. // Compute sse of signal $\vec{Y'} = a\vec{X} + b$ in range $[start\_y \ldots start\_y + length)$
9. $err = \sum_{i=0}^{length} (Y[i + start\_y] - (a \times X[i + start\_x] + b))^2$
10. **return** (a,b,err)
end

Figure 3.5: `Regression` Subroutine

with values of $\vec{Y}$ between $[start\_y, start\_y + length)$, as in Figure 3.4, 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 Section 3.3.6 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 Figure 3.6 looks for the best way to approximate an interval I. It shifts I over $\vec{X}$ and calculates the regression parameters and the

procedure BestMap($\vec{X}$, $\vec{Y}$, Interval I)
Input: Base signal $\vec{X}$; signal $\vec{Y}$; mapped interval I.
Output: Regression parameters $I.a$, $I.b$ at the $I.shift$ value than minimizes the approximation error $I.err$ of the interval I.
begin
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** $shift := 0$ **to** $0..length(\vec{X}) - I.length - 1$ **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. **endif**
10. **endfor**
11. **endif**
end

Figure 3.6: Regression and BestMap Subroutines

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 W is a parameter that denotes the length of the intervals that constitute the base signal.[5] 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 Figure 3.7. The approximation obtained is returned as a list of $B/4$ intervals in i_list. This list is

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

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 $\frac{M}{2}$,..., $2^{\log\frac{B}{4N}-1}\times N$ intervals of size $\frac{M}{\log\frac{B}{4N}-1}$. Thus, the overall running time is:

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

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

procedure GetIntervals($\vec{X}$, $\vec{Y_1}$,..., $\vec{Y_N}$, B, W)
Input: Base signal $\vec{X}$; signals $\vec{Y_1}$,..., $\vec{Y_N}$;
space constraint B; length W of intervals in base signal
Output: List *i_list* of approximated intervals.
begin
1. i_list = ()
2. $\vec{Y} = concat(\vec{Y_1}, \ldots, \vec{Y_N})$ // Virtual assignment
3. // Create an interval for each row $\vec{Y_i}$ (M values each)
4. **for** $i := 1$ **to** N **do**
5. ($I.start$, $I.length$) = ((i-1) $\times$ M, M)
6. BestMap($\vec{X}$, $\vec{Y}$, I, W)
7. i_list.push(I)
8. **endfor**
9. $num\_intervals = N$
10. **while** $num\_intervals{+}{+} < B$ / 4 **do**
11. // i_list is sorted on decreasing order of $I.err$
12. $I = i\_list$.pop()
13. // Break I in 2 pieces
14. ($I_{left}.start$, $I_{left}.length$) = ($I.start$, $I.length$/2)
15. BestMap($\vec{X}$, $\vec{Y}$, I_{left}, W)
16. ($I_{right}.start$, $I_{right}.length$) = ($I.start$+$I.length$/2, $I.length$/2)
17. BestMap($\vec{X}$, $\vec{Y}$, I_{right}, W)
18. i_list.push(I_{left})
19. i_list.push(I_{right})
20. **endwhile**
21. **return** (i_list)
end

Figure 3.7: `GetIntervals` Algorithm

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 NxM values, as depicted in Figure 3.2. 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

procedure GetBase($\vec{Y_1}, \ldots, \vec{Y_N}$, W, M, $maxIns$)
Input: Signals $\vec{Y_1}$,..., $\vec{Y_N}$; length W of intervals in base signal;
length M of signals; maximum number $maxIns$ of selected base intervals
Output: List base_list of candidate base intervals
begin
1. Create $K = \frac{N \times M}{W}$ CBIs of width W
2. For each CBI $Cand_i$, set its benefit to 0
3. Maintain unsorted list Q with CBIs
4. Maintain list base_list with selected stored intervals
5. $LinearErr(Cand_j)$ is the error of approximating $Cand_j$ using standard linear regression
6. **for** $i := 1$ **to** K **do**
7. **for** $j := 1$ **to** K **do**
8. // Calculate error of approximating the j-th CBI by using as base the i-th CBI
9. error=Regression($Cand_i$,$Cand_j$,0,0,W)
10. **if** $error \leq LinearErr(Cand_j)$ **then**
11. $Cand_i.benefit$+=$LinearErr(Cand_j)$-$error$
12. **endif**
13. **endfor**
14. Q.insert($Cand_i$)
15. **endfor**
16. **for** $i := 1$ **to** $maxIns$ **do**
17. C = Q.popBestInterval()
18. base_list.insert(C)
19. **for** $j := 1$ **to** $|Q|$ **do**
20. adjust(Q[j].benefit, C)
21. **endfor**
22. **endfor**
23. **return** (base_list)
end

Figure 3.8: `GetBase` Algorithm

W. For simplicity, we assume that both M and M_{base} are multiples of W. We note here that in Figure 3.7 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 (Figure 3.8) is called during the initialization and update procedure of the base signal. The algorithm receives as inputs the N signals, each of size M, the size W of each base interval, and the maximum number of intervals

CBI	Approximated CBI			Total Benefit
	1	2	3	
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		Total Benefit
	2	3	
2	0.05	0.05	0.10
3	0	0.5	0.50

Adjusted Benefits of Non-Stored CBIs

Figure 3.9: Example of the GetBase Algorithm

$maxIns$ 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 subsection 3.3.3 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 Figure 3.8 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, 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 Figure 3.9. 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.

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 next subsection) 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 $O(maxIns \times n^{1.5})$.

3.3.3 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 $|\vec{X}| \leq M_{base}$.[6] 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 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 Figure 3.10. 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

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

procedure SBR($\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}$, M, $TotalBand$, M_{base})
Input: Current base signal $\vec{X}$; signals $\vec{Y_1}, \ldots, \vec{Y_N}$; length M of each signal;
total bandwidth constraint $TotalBand$; maximum base signal size M_{base}
Output: New base signal stored at $\vec{X}$; approximated data intervals
begin
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. **endif**
14. $\vec{X} = \vec{X}_{new}$
15. Transmit the inserted base intervals, their offsets in the base signal and the regression intervals
end

Figure 3.10: SBR Algorithm

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 Figure 3.12. 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

procedure CalculateError($\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, B, W, Errors, pos$)
Input: Base signal $\vec{X}$; signals $\vec{Y_1}, \ldots, \vec{Y_N}$; $\vec{Y}$; space for intervals B; length W of intervals in base signal; array of calculated errors $Errors$ and position pos of interest
Output: Approximation error when inserting pos intervals into the base signal
begin
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. **endif**

end

Figure 3.11: CalculateError Subroutine

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 $O(n^{1.5} + (nlog(\frac{TotalBand}{N}) + TotalBand \times \sqrt{n} \times M_{base}) \times \log(maxIns))$, 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}.

3.3.4 Design Issues in SBR

Several decisions in the design of the `SBR` algorithm were made to limit its running time complexity. We initially decided to simply look for and exploit linear piece-wise 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),

procedure Search($\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, W, B, base\_list, Errors, start, end$)
Input: Base signal $\vec{X}$; signals $\vec{Y_1}, \ldots, \vec{Y_N}$; length W of base intervals;
array of calculated errors $Errors$; starting and ending positions of binary search
Output: Number of inserted CBIs that achieves the smallest approximation error
begin
1. **if** end == start **then**
2. **return** start
3. **endif**
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. **endif**
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. **endif**
21. **endif**
end

Figure 3.12: `Search` SubRoutine

hoping that the increased model expressiveness 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 3.7) 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 Section 3.5 (Table 3.3), 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.

3.3.5 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 Section 3.5). 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,[7] 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: $O(nlog(\frac{TotalBand}{N}) + TotalBand \times \sqrt{n} \times M_{base})$. In this case the algorithm exhibits a linear dependency on the size of the processed data n.

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

3.3.6 Handling Other Error Metrics

We now present the necessary modifications to the Regression algorithm of Section 3.3.2 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 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 $Y[i + start\_y]$ value is either zero, or close to zero. The Regression algorithm that minimizes the sum squared relative error of the approximation is presented in Figure 3.13.

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.

3.3.7 Providing Strict Error Bounds

The SBR algorithm, as presented above, seeks to minimize a user-defined error metric (i.e., the 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 Section 3.3.6) 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.

3.3.8 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

procedure Regression($\vec{X}$, $\vec{Y}$, $start\_x$, $start\_y$, $length$, $sanity$)
Input: Base signal $\vec{X}$; signal $\vec{Y}$; starting points $start\_x$ and $start\_y$ of the paired intervals; interval length $length$; sanity bound $sanity$
Output: Regression parameters a, b, error err of the approximation.
begin
1. //Compute Regression Parameters
2. $sum\_x = \sum_{0 \le i < length} \frac{X[i+start\_x]}{max\{sanity, |Y[i+start\_y]|\}}$
3. $sum\_y = \sum_{0 \le i < length} \frac{Y[i+start\_y]}{max\{sanity, |Y[i+start\_y]|\}}$
4. $sum\_xy = \sum_{0 \le i < length} \frac{X[i+start\_x]Y[i+start\_y]}{max\{sanity, |Y[i+start\_y]|\}}$
5. $sum\_x2 = \sum_{0 \le i < length} \frac{X[i+start\_x]^2}{max\{sanity, |Y[i+start\_y]|\}}$
6. $sum\_z = \sum_{0 \le i < length} \frac{1}{max\{sanity, |Y[i+start\_y]|\}}$
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. $b = \frac{sum\_y - a \times sum\_x}{sum\_z}$
9. // Compute sum squared relative error of signal $\vec{Y'} = a\vec{X} + b$
   // in range $[start\_y \ldots start\_y + length)$
10. $err = \sum_{i=0}^{length-1} (\frac{Y[i+start\_y] - (a \times X[i+start\_x] + b)}{max\{sanity, |Y[i+start\_y]|\}})^2$
11. **return** (a,b,err)
end

Figure 3.13: Regression subroutine for the sum squared relative error

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 and 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 have do not reuse any of the algorithm's steps), or keep a list of the choices (Line 12 in Algorithm 3.7) 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 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. Thus, in this case only the

results of GetBase() can be reused.

3.4 Localized Groups

3.4.1 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 Figure 3.1. 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 piece-wise 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 Figure 3.1, 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:

$$k_1 \times (S-1) + k_{gl} \times S \times H \leq \ k \times S \times H \Longrightarrow k_1 \leq (k - k_{gl}) \times \frac{S}{S-1} H \qquad (3.1)$$

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

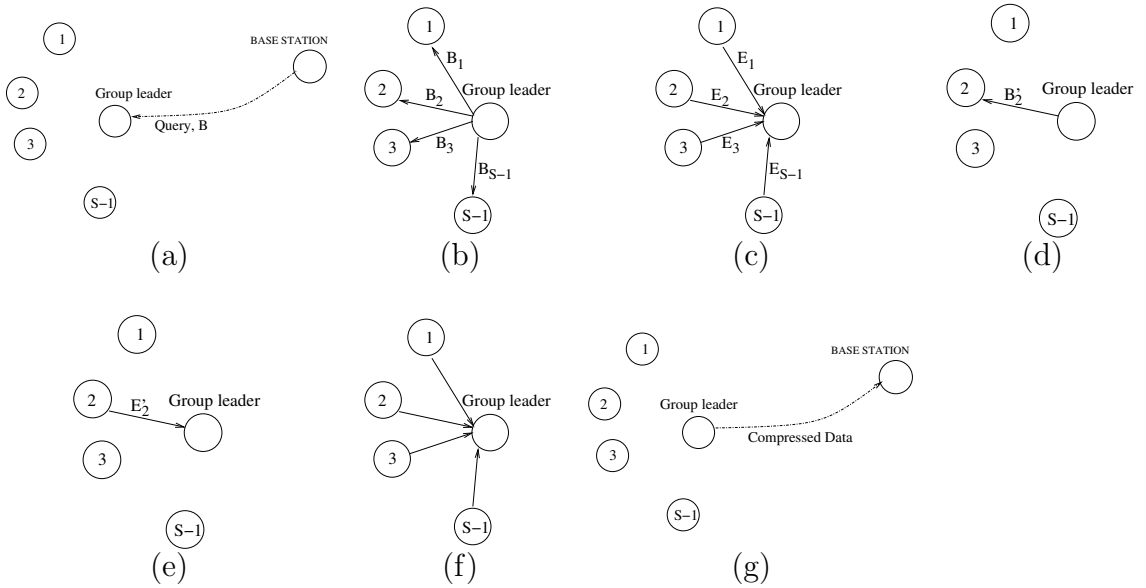


Figure 3.14: (a) Transmission of query and desired Bandwidth consumption B. (b) Group Leader assigns initial bandwidth limit to nodes in group. (c) Nodes report error with given bandwidth limit. (d,e) Handshake procedure: Group Leader assigns additional bandwidth to node with largest error, and receives 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.

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.

3.4.2 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 Section 3.4.1). 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. λ=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 Figure 3.14 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, Figure 3.14(b,c). In Figure 3.14(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()` (Figure 3.14(e)). These two steps are repeated α times, assigning each time additional bandwidth to the node with the largest error. In Figure 3.14(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.

3.4.3 Selecting the Group Leader

In our discussion so far we have not referenced the procedure with which the sensor nodes (i) determine how many 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 [YF04], 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 [YF04], the techniques proposed in [YF04] 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 ([YF04]). 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 HEED 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 [YF04]. 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).

3.4.4 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 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 Section 3.4.2), 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 Section 3.4.2, 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.

3.5 Experiments

In this section, we provide an experimental evaluation of our techniques. In subsection 3.5.1 we compare the SBR algorithm against standard approximation techniques (Wavelets, DCT, Histograms). In subsection 3.5.2 we compare the `GetBase()` algorithm against alternative base-signal constructions, while in subsection 3.5.3 we present an analysis of the SBR algorithm. In subsection 3.5.4 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:

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, dewpoint temperature, wind speed, wind peak, solar irradiance and relative humidity weather measurements for the station in the university of Washington, and for year 2002.[8]

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

Compression Ratio	Weather Data				Stock Data			
	SBR	Wavelets	DCT	Histograms	SBR	Wavelets	DCT	Histograms
5%	1.160	2.187	35.835	27.692	0.089	0.123	0.232	0.283
10%	0.403	0.824	20.169	11.294	0.033	0.056	0.208	0.233
15%	0.209	0.514	14.328	5.432	0.017	0.034	0.192	0.214
20%	0.118	0.356	10.774	3.009	0.009	0.022	0.179	0.199
25%	0.069	0.258	8.975	1.507	0.006	0.015	0.166	0.182
30%	0.043	0.191	6.526	0.995	0.003	0.011	0.153	0.169

Table 3.2: Average SSE Error Varying the Compression Ratio for Weather and Stock Data Sets

Compression Ratio	Average SSE Error				Total Sum Squared Relative Error			
	SBR	Wavelets	DCT	Histograms	SBR	Wavelets	DCT	Histograms
5%	9,631	29,938	15,714	165,241	922	38,477	9,019	139,528
10%	5,071	12,349	10,173	45,610	503	19,186	3,002	62,337
15%	3,192	7,998	6,767	23,311	325	12,885	1,400	36,812
20%	2,170	5,821	5,661	15,581	222	10,954	1,192	34,820
25%	1,527	4,468	4,791	11,340	158	6,915	823	33,237
30%	1,091	3,537	4,157	8,689	116	3,865	721	30,010

Table 3.3: Errors Varying the Compression Ratio for Phone Call Data Set

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.

3.5.1 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 [CGRS00], equi-depth Histograms [PIHS96] 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.

Varying the Compression Ratio

We varied the compression ratio (size of the transmitted data $TotalBand$ over the data size n) from 5% to 30%. 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 3.2 and 3.3 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 4.4 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 3.3. Depending on the compression ratio, our method was up to 49 times better than Wavelets, 9.8 times better than DCT and 258 times better than Histograms. We note here that for this comparison we used Haar Wavelets that are optimal only under the sum-squared-error. The work of [GG02] 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. Moreover, as the number of base intervals stored in the base signal increases, the chances of accurately mapping a data interval on some area of the base signal is increased as well, a feature unique in SBR when compared to the other techniques that utilize a fixed, predetermined basis for the data compression. This is why the improvements of SBR over the competitive techniques increase as the size of the transmitted data increases (more base intervals can be inserted as the bandwidth constraint increases). On the other hand, the same argument leads us to believe that SBR might not be the method of choice if very small compression ratios (i.e., 1%) are desired. In such cases, unless the base station is willing to accept a larger size of transmitted data in the initial transmission, the small bandwidth constraint would prevent the inclusion of base intervals, thus resulting in an approximation of the data using only standard linear regression. Of course, the use of very small compression ratios might not be desirable in sensor network applications, since they may result in significant approximation errors.

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 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.

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

Table 3.4: Errors Varying the Compression Ratio for the Mixed Data Set

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 3.4 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 3.4 simply shows that SBR is more robust, than Wavelets and Histograms for

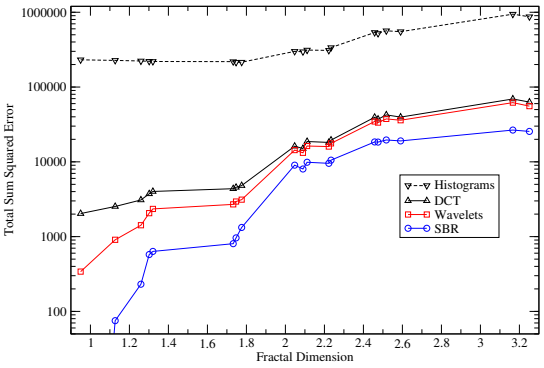


Figure 3.15: Total SSE error vs Fractal Dimension

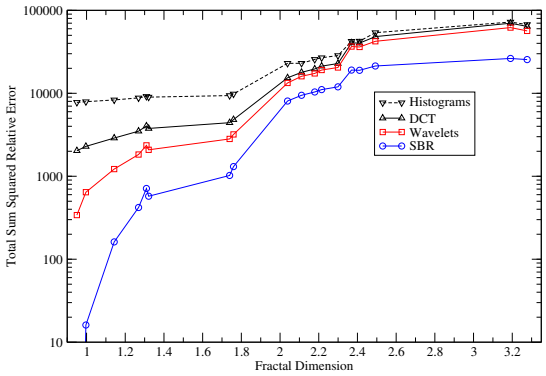


Figure 3.16: Total SSRE error vs Fractal Dimension

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 the 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. In Figures 3.15 and 3.16 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 [BF95]. As explained in [BF95] 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.

3.5.2 Alternative Base Signal Constructions

We present two alternative techniques to our `GetBase()` algorithm. 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

Data Set	Error over GetBase()		
	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

Table 3.5: Comparison to Alternative Base Signals

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 [PTVF92]) 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 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).[9] 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

[9] [KLKF00] presents an application of this observation in a different context.

Data Set	Transmission									
	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

Table 3.6: Number of Inserted Base Intervals per Transmission

`GetBaseDCT()`. We notice that we do not need to store these intervals implicitly, as they can be computed on the fly.

In Table 3.5 we compare the approximations obtained by using the base signals computed in algorithm `GetBase()` 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 3.3.2 would further improve the results of our method. The compression ratio was set to 10%. We notice that `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.

3.5.3 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 Figure 3.17 we plot the average time of each transmission operation for the

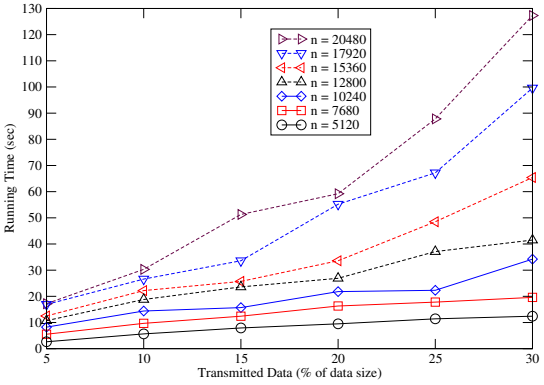


Figure 3.17: Average Running Time vs TotalBand

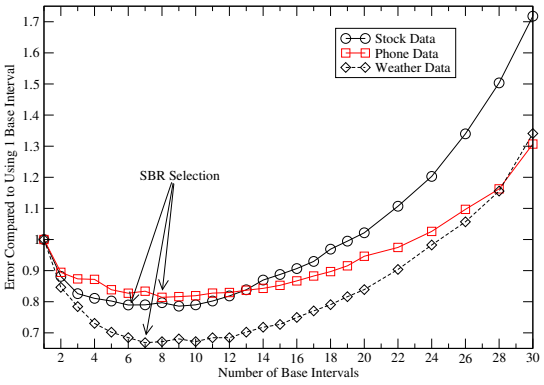


Figure 3.18: SSE error vs base signal size

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,[10] 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 300MHz processor. As expected (see Section 3.3.3) 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 Figure 3.18 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

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

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 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 3.6 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.

3.5.4 Localized Groups

We now seek to evaluate the benefits of localized group processing described in Section 3.4. 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.[11] 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 equation 3.1.

In all cases the number of transmissions (update operations) was set to 6, while $\lambda = 0.6$, and $M_{base} = 2048$. Figure 3.19 presents the total sum squared error for all

[11] 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.

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 3.20. Table 3.7 also presents the 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 Figures 3.19 and 3.20 and Table 3.7:

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 3.7), 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 $r + \delta r$ (ex: $\delta r = 1$) for the same value of k. We

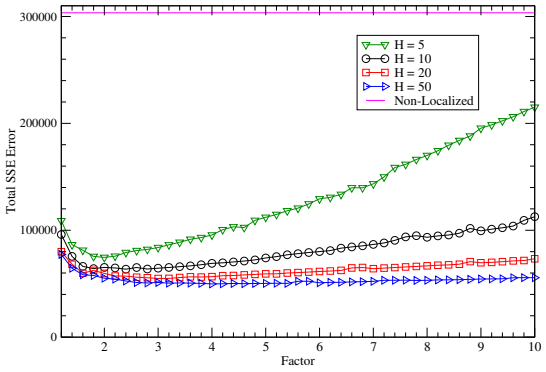


Figure 3.19: Selecting the ratio $\frac{k_1}{k_{gl}}$ vs H

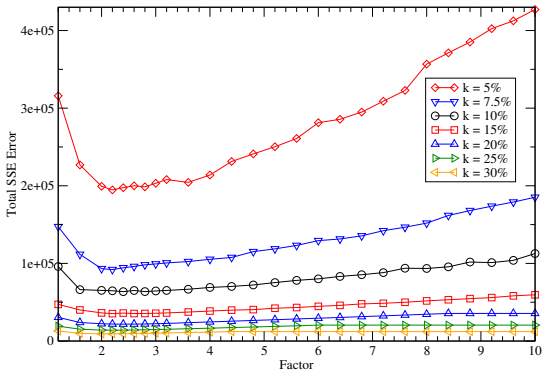


Figure 3.20: Selecting the ratio $\frac{k_1}{k_{gl}}$ vs k

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 3.7: Error comparison of Localized vs non-Localized algorithm

Update	Sensor				
	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%

Table 3.8: Compression ratios for sensors (k=10%, H=10)

can then show that:

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

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 3.8 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 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 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.

Chapter 4

Extended Wavelets for Data Sets with Multiple Measures

4.1 Introduction

Approximate query processing techniques have been proposed recently as methods for providing fast and fairly accurate answers to complex queries over large quantities of data. The most popular approximate processing techniques include histograms, random sampling and wavelets. In recent years there has been a flurry of research on the application of these techniques to such areas as selectivity estimation and approximate query processing. The work in [CGRS00, GG02, GG04, MVW98, SS02, VW99] demonstrated that wavelets can achieve increased accuracy to queries over histograms and random sampling.

Despite the surge of interest in wavelet-based data reduction and approximation in database systems, relatively little attention has been paid to the application of wavelet techniques to complex tabular data sets *with multiple measures* (multiple numeric entries for each table cell). Such massive, multi-measure tables arise naturally in several application domains, including OLAP environments and time-series analysis/correlation systems. As an example, a corporate sales database may

tabulate, for each available product, (1) the number of items sold, (2) revenue and profit numbers for the product, and (3) costs associated with the product, such as shipping and storage costs. Similarly, real-life applications that monitor continuous time-series typically have to deal with several readings (measures) that evolve over time; for example, a network-traffic monitoring system takes readings on each time-tick from a number of distinct elements, such as routers and switches, in the underlying network and typically several measures of interest need to be monitored (e.g., input/output traffic numbers for each router or switch interface) even for a fixed network element [net].

Traditionally, two obvious strategies, termed *Individual* and *Combined*, have been employed when adapting wavelet-based methods over such multi-measure data sets. The *Individual* algorithm performs the wavelet decomposition on each of the individual measures, and stores the important coefficients for each measure separately. On the other hand, the *Combined* algorithm performs a joint wavelet decomposition on the multi-measure data set by treating all the measures as a *vector* of values and, at the end, determines a subset of vectors of coefficient values to retain in the synopsis. As demonstrated in this chapter, such obvious *individual* or *combined* approaches can lead to poor synopsis-storage utilization and suboptimal solutions even in very simple cases. Due to the nature of the wavelet decomposition and the possible correlations across different measures, there are many scenarios in which multiple – but not necessarily all – wavelet coefficients at the same coordinates have large values, and are thus beneficial to retain, for instance, in an L_2-optimized synopsis. In such cases, the *Individual* algorithm essentially replicates the storage of the shared coordinates

multiple times, wasting valuable synopsis storage. The *Combined* algorithm, on the other hand, stores *all* coefficient values sharing the same coordinates, thus wasting space by storing small, unimportant values for certain measures.

In this chapter we propose a novel approach for effectively adapting wavelet-based data reduction methods to multi-measure data sets through the use of *extended wavelet coefficients*. Briefly, an extended wavelet coefficient can store multiple coefficient values for different – but not necessarily all – measures. The end result is a flexible, space-efficient storage scheme that can eliminate the disadvantages of both the *Individual* and *Combined* algorithms discussed above. We then consider the problem of constructing effective extended wavelet coefficient synopses (under a given storage constraint) optimized for the (1) *weighted sum-squared error*, and (2) *relative error* in the approximate data reconstruction. Our synopsis-construction problems are natural generalizations of the corresponding problems for conventional (i.e., L_2-error) wavelet synopses [VW99, CGRS01] and probabilistic (i.e., relative-error) wavelet synopses [GG04] for the *single-measure* case. We demonstrate that, in the presence of multiple measure, choosing an effective subset of extended wavelet coefficients gives rise to difficult optimization problems that are significantly more complex than their single-measure counterparts. This is primarily due to our more involved extended-coefficient storage format that forces non-trivial dependencies between thresholding decisions made across different measures. We propose optimal solutions based on novel algorithmic formulations that employ Dynamic-Programming (DP) ideas. Given the high time and space complexities of our exact DP schemes, we also introduce fast, greedy approximation algorithms (based on the idea of *marginal*

error gains) that produce near-optimal solutions. To the best of our knowledge, our work represents the first principled, methodical study of effective wavelet-based data reduction techniques for multi-measure data sets.

4.2 Preliminaries

In this section, we provide a quick introduction to the conventional Haar wavelet decomposition and wavelet-coefficient synopses in both one and multiple dimensions. We also discuss the existing *Individual* and *Combined* strategies for handling multiple measures and demonstrate some of their important shortcomings. Finally, we introduce the notion of an *extended wavelet coefficient* which forms the basis for our proposed approach and data-reduction algorithms.

4.2.1 One-Dimensional Haar Wavelets

Wavelets are a useful mathematical tool for hierarchically decomposing functions in ways that are both efficient and theoretically sound. Broadly speaking, the wavelet decomposition of a function consists of a coarse overall approximation along with detail coefficients that influence the function at various scales [SDS96]. The wavelet decomposition has excellent energy compaction and de-correlation properties, which can be used to effectively generate compact representations that exploit the structure of data. Suppose we are given the one-dimensional data vector A containing the $N = 8$ data values $A = [2, 2, 0, 2, 3, 5, 4, 4]$. The Haar wavelet transform of A can be computed as follows. We first average the values together pairwise to get a new "lower-

resolution" representation of the data with the following average values $[2, 1, 4, 4]$. In other words, the average of the first two values (that is, 2 and 2) is 2, that of the next two values (that is, 0 and 2) is 1, and so on. Obviously, some information has been lost in this averaging process. To be able to restore the original values of the data array, we store some *detail coefficients*, that capture the missing information. In Haar wavelets, these detail coefficients are simply the differences of the (second of the) averaged values from the computed pairwise average. Thus, in our simple example, for the first pair of averaged values, the detail coefficient is 0 since $2 - 2 = 0$, for the second we again need to store -1 since $1 - 2 = -1$. Note that no information has been lost in this process – it is fairly simple to reconstruct the eight values of the original data array from the lower-resolution array containing the four averages and the four detail coefficients. Recursively applying the above pairwise averaging and differencing process on the lower-resolution array containing the averages, we get the following full decomposition:

Resolution	**Averages**	**Detail Coefficients**
3	[2, 2, 0, 2, 3, 5, 4, 4]	—
2	[2, 1, 4, 4]	[0, -1, -1, 0]
1	[3/2, 4]	[1/2, 0]
0	[11/4]	[-5/4]

The *wavelet transform* (also known as the *wavelet decomposition*) of A is the single coefficient representing the overall average of the data values followed by the detail coefficients in the order of increasing resolution. Thus, the one-dimensional Haar wavelet transform of A is given by $W_A = [11/4, -5/4, 1/2, 0, 0, -1, -1, 0]$. Each entry in W_A is called a *wavelet coefficient*. The main advantage of using W_A

instead of the original data vector A is that for vectors containing similar values most of the detail coefficients tend to have very small values. Thus, eliminating such small coefficients from the wavelet transform (i.e., treating them as zeros) introduces only small errors when reconstructing the original data, resulting in a very effective form of lossy data compression [SDS96].

Note that, intuitively, wavelet coefficients carry different weights with respect to their importance in rebuilding the original data values. For example, the overall average is obviously more important than any detail coefficient since it affects the reconstruction of all entries in the data array. In order to equalize the importance of all wavelet coefficients, we need to *normalize* the final entries of W_A appropriately. A common normalization scheme [SDS96] is to divide each wavelet coefficient by $\sqrt{2^l}$, where l denotes the *level of resolution* at which the coefficient appears (with $l = 0$ corresponding to the "coarsest" resolution level). Thus, the normalized coefficient, c_i^*, is $c_i/\sqrt{2^{\texttt{level}(c_i)}}$.

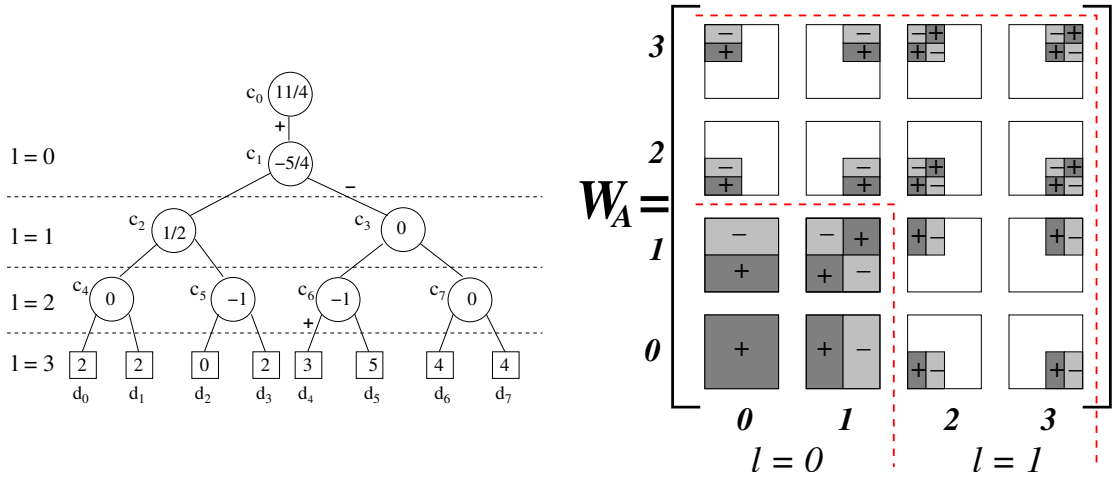
The Haar Coefficient Error Tree. A helpful tool for exploring and understanding the key properties of the Haar wavelet decomposition is the *error tree* structure [MVW98]. The error tree is a hierarchical structure built based on the wavelet transform process (even though it is primarily used as a conceptual tool, an error tree can be easily constructed in linear $O(N)$ time). Figure 4.1(a) depicts the error tree for our example data vector A. Each internal node c_i $(i = 0, \ldots, 7)$ is associated with a wavelet coefficient value, and each leaf d_i $(i = 0, \ldots, 7)$ is associated with a value in the original data array; in both cases, the index/coordinate i denotes the positions in

Symbol	Description ($i \in \{0, \ldots, N-1\}$, $j \in \{1, \ldots, M\}$, j index/subscript is dropped for $M = 1$)
N	Number of data-array cells
D	Data-array dimensionality
M	Number of data-set measures
B	Space budget for synopsis
A, W_A	Input data and wavelet transform arrays
d_{ij}	Data value for i^{th} cell and j^{th} measure of data array
$\hat{d}_{ij}$	Reconstructed data value for i^{th} cell and j^{th} measure
c_{ij}, c^*_{ij}	Un-normalized/normalized Haar coefficient at coordinate i for the j^{th} measure
$\texttt{path}(u)$	All non-zero proper ancestors of u in the error tree
EC_i	Extended wavelet coefficient at coordinate i
H	Storage space for the extended wavelet coefficient header (coefficient coordinates and bitmap)

Table 4.1: Notation.

the data array or error tree. For example, c_0 corresponds to the overall average of A. The resolution levels l for the coefficients (corresponding to levels in the tree) are also depicted. We use the terms "node" and "coefficient" interchangeably in what follows. Table 4.1 summarizes some of the key notational conventions used in this chapter; additional notation is introduced when necessary. Detailed symbol definitions are provided at the appropriate locations in the text. For simplicity, the notation assumes one-dimensional wavelets – extensions to multi-dimensional wavelets (Section 4.2.2) are straightforward.

Given a node u in an error tree T, let $\texttt{path}(u)$ denote the set of all proper ancestors of u in T (i.e., the nodes on the path from u to the root of T, including the root but not u) with non-zero coefficients. A key property of the Haar wavelet decomposition is that the reconstruction of any data value d_i depends only on the values of coefficients on $\texttt{path}(d_i)$; more specifically, we have $d_i = \sum_{c_j \in \texttt{path}(d_i)} \delta_{ij} \cdot c_j$,



(a) **(b)**

Figure 4.1: (a) Error-tree structure for our example data vector A ($N = 8$). (b) Support regions and signs for the sixteen nonstandard two-dimensional Haar basis functions. The coefficient magnitudes are multiplied by $+1$ (-1) where a sign of $+$ (respectively, $-$) appears, and 0 in blank areas.

where $\delta_{ij} = +1$ if d_i is in the left child subtree of c_j or $j = 0$, and $\delta_{ij} = -1$ otherwise. Thus, reconstructing any data value involves summing at most $\log N + 1$ coefficients. For example, in Figure 4.1, $d_4 = c_0 - c_1 + c_6 = \frac{11}{4} - (-\frac{5}{4}) + (-1) = 3$. The *support region* for a coefficient c_i is defined as the set of (contiguous) data values that c_i is used to reconstruct; the support region for a coefficient c_i is uniquely identified by its coordinate i.

Similarly, the evaluation of the sum of data values in the range $[i..j]$ depends only on the values of coefficients on $\texttt{path}(d_i)$ or $\texttt{path}(d_j)$; to see why this is the case, consider that any coefficient (besides the overall average) whose support region lies entirely within the query range will make contributions to the data values lying to its left and right subtrees that will cancel out each other. Thus, evaluating any range query involves considering the contributions of at most $2 \log N - 1$ coefficients. To calculate the contribution of each of these coefficients, we need to multiply its value

by the difference in the number of data values within the range query that lie to the left and to the right of the coefficient in the error tree. For example, in Figure 4.1, $Sum[2..5] = 4 \times c_0 + (-2) \times c_2 + 2 \times c_3 = 11 - 1 + 0 = 10$.

4.2.2 Multi-Dimensional Haar Wavelets

The Haar wavelet decomposition can be extended to *multi-dimensional* data arrays using two distinct methods, namely the *standard* and *nonstandard* Haar decomposition [SDS96]. Each of these transforms results from a natural generalization of the one-dimensional decomposition process described above, and both have been used in a wide variety of applications, including approximate query answering over high-dimensional DSS data sets [CGRS00, VW99].

As in the one-dimensional case, the Haar decomposition of a D-dimensional data array A results in a D-dimensional wavelet-coefficient array W_A with the same dimension ranges and number of entries. (The full details as well as efficient decomposition algorithms can be found in [CGRS00, VW99].) Consider a D-dimensional wavelet coefficient W in the (standard or nonstandard) wavelet-coefficient array W_A. W contributes to the reconstruction of a D-dimensional rectangular region of cells in the original data array A (i.e., W's *support region*). Further, the sign of W's contribution ($+W$ or $-W$) can vary along the quadrants of W's support region in A. As an example, Figure 4.1(b) depicts the support regions and signs of the sixteen nonstandard, two-dimensional Haar coefficients in the corresponding locations of a 4×4 wavelet-coefficient array W_A. The blank areas for each coefficient correspond

to regions of A whose reconstruction is independent of the coefficient, i.e., the coefficient's contribution is 0. Thus, $W_A[0,0]$ is the overall average that contributes positively (i.e.,"$+W_A[0,0]$") to the reconstruction of all values in A, whereas $W_A[3,3]$ is a detail coefficient that contributes (with the signs shown in Figure 4.1(b)) only to values in A's upper right quadrant. Each data cell in A can be accurately reconstructed by adding up the contributions (with the appropriate signs) of those coefficients whose support regions include the cell. Figure 4.1(b) also depicts the two *levels of resolution* ($l = 0, 1$) for our example two-dimensional Haar coefficients; as in the one-dimensional case, these levels define the appropriate constants for normalizing coefficient values [CGRS00, SDS96].

Error-tree structures for multi-dimensional Haar wavelets can be constructed (once again in linear $O(N)$ time) in a manner similar to those for the one-dimensional case, but their semantics and structure are somewhat more complex. A major difference is that, in a D-dimensional error tree, each node (except for the root, i.e., the overall average) actually corresponds to a *set* of $2^D - 1$ wavelet coefficients that have the same support region but different quadrant signs and magnitudes for their contribution. Furthermore, each (non-root) node t in a D-dimensional error tree has 2^D children corresponding to the quadrants of the (common) support region of all coefficients in t.[1] (Note that the sign of each coefficient's contribution to the leaf (data) values residing at each of its children in the tree is determined by the coefficient's quadrant sign information.) As an example, Figure 4.2 depicts the error-tree structure

[1] The number of children (coefficients) for an internal error-tree node can actually be less than 2^D (respectively, $2^D - 1$) when the sizes of the data dimensions are not all equal. In these situations, the exponent for 2 is determined by the number of dimensions that are *"active"* at the current level of the decomposition (i.e., those dimensions that are still being recursively split by averaging/differencing).

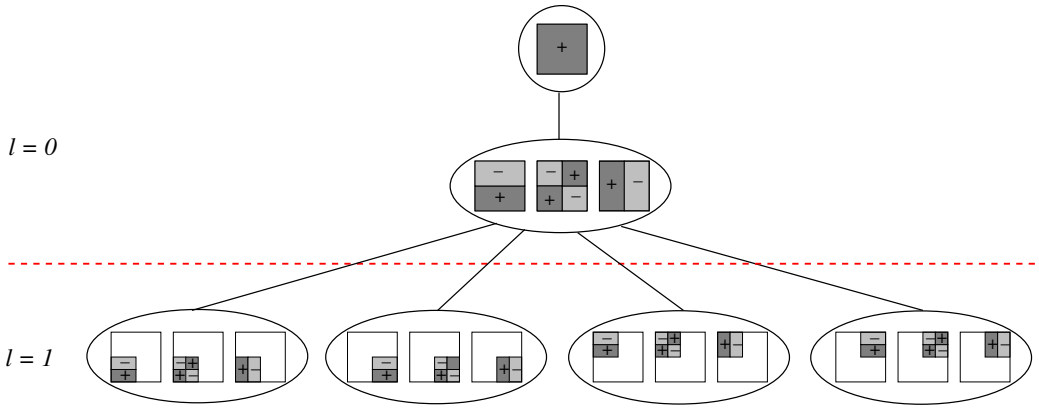


Figure 4.2: Error-tree structure for the sixteen nonstandard two-dimensional Haar coefficients for a 4×4 data array (data values omitted for clarity).

for the two-dimensional 4×4 Haar coefficient array in Figure 4.1(b). Thus, the (single) child t of the root node contains the coefficients $W_A[0,1], W_A[1,0]$, and $W_A[1,1]$, and has four children corresponding to the four 2×2 quadrants of the array; the child corresponding to the lower-left quadrant contains the coefficients $W_A[0,2], W_A[2,0]$, and $W_A[2,2]$, and all coefficients in t contribute with a "+" sign to all values in this quadrant.

Based on the above generalization of the error-tree structure to multiple dimensions, we can naturally extend the process for data-value reconstruction to multi-dimensional Haar wavelets. Once again, the reconstruction of d_i depends only on the *coefficient sets* for all error-tree nodes in $\texttt{path}(d_i)$, where the sign of the contribution for each coefficient W in node $t \in \texttt{path}(d_i)$ is determined by the quadrant sign information for W.

4.2.3 Wavelet-based Data Reduction: Coefficient Thresholding

Given a limited amount of storage for building a *wavelet synopsis* of the input data array A, a thresholding procedure retains a certain number $B \ll N$ of the coefficients in W_A as a highly-compressed approximate representation of the original data (the remaining coefficients are implicitly set to 0). The goal of coefficient thresholding is to determine the "best" subset of B coefficients to retain, so that some overall error measure in the approximation is minimized. The method of choice for the vast majority of earlier studies on wavelet-based data reduction and approximation [CGRS00, MVW98, MVW00, VW99] is *conventional coefficient thresholding* that greedily retains retains the B largest Haar-wavelet coefficients in *absolute normalized value*. It is a well-known fact that this thresholding method is in fact *provably optimal* with respect to minimizing the overall sum-squared error (i.e., L_2*-norm error*) in the data compression [SDS96].

More formally, letting $\hat{d}_i$ denote the (approximate) reconstructed data value for cell i, retaining the B largest normalized coefficients implies that the resulting synopsis minimizes the quantity $\sum_i (\hat{d}_i - d_i)^2$ (for the given amount of space B). This fact follows from the *orthonormality* of the normalized Haar-wavelet basis which, by *Parseval's theorem*, implies that the *energy* of the signal is the same in both the data and the (normalized) wavelet domain – that is, $\sum_i d_i^2 = \sum_i (c_i^*)^2$ (thus, the "energy" loss is minimized when dropping the smallest normalized coefficients from the synopsis).

Each Haar coefficient is stored in the wavelet synopsis as a pair $< i, c_i^* >$, where

Coordinate	Value	Normalized Value	Stored Tuple
0	$11/4$	$11/4$	$< 0, 11/4 >$
1	$-5/4$	$-5/4$	$< 1, -5/4 >$
2	$1/2$	$\sqrt{2}/4$	$< 2, \sqrt{2}/4 >$
3	0	0	—
4	0	0	—
5	-1	$-1/2$	$< 5, -1/2 >$
6	-1	$-1/2$	$< 6, -1/2 >$
7	0	0	—

Table 4.2: Wavelet-Coefficient Tuples for Example Data Vector A ($N = 8$).

i denotes the index/coordinate of the coefficient and c_i^* denotes its (normalized) value. In the case of D-dimensional data, each synopsis coefficient is stored as a $(D + 1)$-tuple $< i_1, i_2, \ldots, i_D, c^*_{i_1,i_2,\ldots,i_D} >$, where $i_1, \ldots, i_D$ denote the coefficient's coordinates in the (D-dimensional) wavelet-coefficient array W_A. Table 4.2 depicts the tuples corresponding to each Haar wavelet coefficient for our example array A (of course, zero-valued coefficients are never retained in a synopsis).

4.2.4 Existing Approaches for Multiple Measures

Two existing approaches [SDS96] (termed *Individual* and *Combined*) have been proposed for adapting wavelet-based data reduction to data sets with multiple measures – both are straightforward generalizations of the single-measure case. The *Individual* strategy performs an independent wavelet decomposition for each individual measure, and the decisions on which coefficients to retain are made independently for each measure. In the *Combined* approach, both the original data values and the produced wavelet coefficients are treated as M-component vectors (where M denotes the number of data measures). The pairwise averaging and differencing procedure

described above is then performed between corresponding vector components (i.e., values for the same measure). The *Combined* thresholding procedure is very similar to that for the single-measure case: the coefficient vectors retained in the synopsis are the ones with the largest values for the L_2 vector norm.

We also use the terms *individual* and *combined* coefficient to refer to the coefficient values that result from the corresponding decomposition algorithms. Thus, a combined coefficient is an M-component vector that stores individual coefficient values for each of the M measures in the data set (at given coordinates). An example of the *Combined* decomposition algorithm is shown in Table 4.3. Our example data set here comprises two measures: the values for the first measure are identical to those in our first example array in Section 4.2.1, while the values for the second measure are $[4, 6, 3, 5, 2, 8, 3, 3]$. Thus, in Table 4.3, the first (second) row of each vector corresponds to either data or coefficient values for the first (respectively, second) measure. The final set of combined coefficients is $W_A = [\begin{bmatrix} 11/4 \\ 17/4 \end{bmatrix}, [\begin{bmatrix} -5/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ -3 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}]$.

The *Combined* data-reduction strategy is expected to achieve better storage utilization than the *Individual* algorithm for the L_2 error metric in data sets where multiple component values for the same combined coefficient are simultaneously "large" (in terms of their absolute normalized value). In such cases, the coordinates of such large combined coefficients are stored only once, thus allowing for a more compact representation. More compact representations imply the ability to store larger numbers of coefficient values and, thus, improved result accuracy (for a given space budget). On

Resolution	Averages	Detail Coefficients
3	$[\begin{bmatrix}2\\4\end{bmatrix}\ \begin{bmatrix}2\\6\end{bmatrix}\ \begin{bmatrix}0\\3\end{bmatrix}\ \begin{bmatrix}2\\5\end{bmatrix}\ \begin{bmatrix}3\\2\end{bmatrix}\ \begin{bmatrix}5\\8\end{bmatrix}\ \begin{bmatrix}4\\3\end{bmatrix}\ \begin{bmatrix}4\\3\end{bmatrix}]$	—
2	$[\begin{bmatrix}2\\5\end{bmatrix}\ \begin{bmatrix}1\\4\end{bmatrix}\ \begin{bmatrix}4\\5\end{bmatrix}\ \begin{bmatrix}4\\3\end{bmatrix}]$	$[\begin{bmatrix}0\\-1\end{bmatrix}\ \begin{bmatrix}-1\\-1\end{bmatrix}\ \begin{bmatrix}-1\\-3\end{bmatrix}\ \begin{bmatrix}0\\0\end{bmatrix}]$
1	$[\begin{bmatrix}3/2\\9/2\end{bmatrix}\ \begin{bmatrix}4\\4\end{bmatrix}]$	$[\begin{bmatrix}1/2\\1/2\end{bmatrix}\ \begin{bmatrix}0\\1\end{bmatrix}]$
0	$[\begin{bmatrix}11/4\\17/4\end{bmatrix}]$	$[\begin{bmatrix}-5/4\\1/4\end{bmatrix}]$

Table 4.3: Example Combined Wavelet Decomposition.

Case A					Case B				
	Coordinate	Values				Coordinate	Values		
Available	0	100	0	0	Available	0	100	100	100
Coefficients	1	0	100	0	Coefficients	1	0	100	0
	Coordinate	Values				Coordinate	Values		
Combined Retains	0	100	0	0	Combined Retains	0	100	100	100
	Coordinate	Value	Measure			Coordinate	Value	Measure	
Individual Retains	0	100	1		Individual Retains	0	100	3	
	1	100	2			0	100	2	
Combined Benefit $= 100^2 = 10000$					Combined Benefit $= 100^2 + 100^2 + 100^2 = 30000$				
Individual Benefit $= 100^2 + 100^2 = 20000$					Individual Benefit $= 100^2 + 100^2 = 20000$				
$\frac{\text{Combined Benefit}}{\text{Individual Benefit}} = \frac{10000}{20000} = 50\%$					$\frac{\text{Individual Benefit}}{\text{Combined Benefit}} = \frac{20000}{30000} \approx 66.7\%$				

Table 4.4: Sub-optimality of the *Combined* and *Individual* Strategies.

the other hand, in many scenarios, a combined coefficient might help reduce the error significantly in only one, or few, measures. In such cases, some of the space occupied by the combined-coefficient components is essentially wasted, without improving the overall quality of the approximate results.

Table 4.4 depicts only two combined coefficients for a one-dimensional data set with three measures. The actual data that helped construct these two coefficients, or the remaining set of coefficients are not important, since the sole purpose of this example is to show that both the *Individual* and *Combined* strategies can result in

poor choices, even when choosing between just two coefficients. We assume that each dimension coordinate and each coefficient value require one unit of space. Under this scenario, each combined-coefficient tuple occupies four space units (one coordinate + three measure values), while each individual coefficient occupies only two space units. For a storage constraint of four units of space, the *Combined* algorithm can thus select only one tuple to store, while the *Individual* algorithm can store up to two individual coefficients. By Parseval's theorem (Section 4.2.3), the benefit of retaining any single coefficient value is equal to the its squared normalized value. As Table 4.4 shows, in Case A, for the given storage constraint, the *Combined* algorithm chooses a solution with only half the benefit of the solution picked by the *Individual* algorithm. The roles are reversed in Case B, where the *Individual* algorithm selects a solution with only two thirds of the benefit achieved by the *Combined* algorithm. Note that, in Case B, the ties on the retained coefficients for the *Individual* algorithm are broken arbitrarily, as four individual coefficients have the same benefit. By increasing the number of measures in the data set for Case A, and the number of dimensions for Case B, one can easily create examples where the quality of the sub-optimal solutions returned by the *Combined* and *Individual* strategies (respectively) is significantly worse than the optimal choice. For instance, by expanding our one-dimensional data set of Table 4.4 to a data set with M measures, and considering which $\frac{M+1}{2}$ candidate individual coefficients to retain under a storage constraint of $M+1$ space units, it can be shown that, in Case A: $\frac{\text{Combined Benefit}}{\text{Individual Benefit}} = \frac{2}{M+1}$, while, in Case B: $\frac{\text{Individual Benefit}}{\text{Combined Benefit}} = \frac{M+1}{2M}$.

An additional disadvantage of the *Combined* data-reduction strategy is that it

cannot be easily adapted for cases when one would like to assign different weights on the quality of the answers for different measures. For example, in colored image databases, data sets form two-dimensional arrays with three measures, namely the pixel values for each of the three basic colors (Red, Green and Blue). It has been shown [FvDFH90] that better image compression is possible by first converting the image values from the RGB color space to the YIQ color space, thus separating the luminance (Y) from the chromatic information (I and Q). Since human perception is more sensitive to variations in Y and less sensitive to variations in Q, we may want to specify a larger weight for errors in Y and a smaller weight for errors in Q. In such scenarios, the use of the *Combined* algorithm becomes problematic, since it cannot devote different fractions of the available space to different measures, even though the coefficient values within each vector are weighted differently. Moreover, for data sets with several measures, it seems quite unlikely that the coefficient values across *all* measures would be simultaneously large or small (i.e., all measure values in the data are positively correlated). For such data sets, the *Combined* algorithm would clearly waste synopsis storage space, without significantly improving the accuracy of the resulting approximation for all measures.

On the other hand, there are cases when we can expect *multiple* coefficient values of a combined coefficient to have large values. As discussed in Sections 4.2.1-4.2.2, Haar coefficient values are normalized based on their respective resolution levels. Due to this normalization, coefficient values at lower (i.e., "coarser") resolution levels typically tend to have larger values, and this occurs for all measures. Another scenario where multiple large coefficient values might occur at the same coordinate(s) arises

for *sparse* data sets (that are typical in real-life high-dimensional data-analysis applications). In such data sets, there often are sparse regions of the data space with only one or very few data tuples present. Depending on the sizes of such regions and the tuple data values, such "spikes" in the input data signal can potentially create large Haar coefficient values across many measures. Clearly, in such scenarios, the *Combined* algorithm can provide significant advantages over an *Individual* strategy by avoiding the replication of coordinates for multiple measures in the synopsis.

4.2.5 Our Approach: Extended Wavelet Coefficients

The above-described *Individual* and *Combined* essentially represent the two extremes of the design spectrum, by assuming that either *one* or *all* values of a Haar coefficient are important for the synopsis (and can share their coordinates). Given the important shortcomings of both techniques when dealing with multi-measure data sets, we now introduce the notion of an *extended wavelet coefficient* that tries to bridge the gap between the two extremes by providing an efficient, flexible storage format for retaining *any subset* of coefficient values.

Definition 1 *An extended wavelet coefficient EC for a D-dimensional data set with M measures is defined as a triple* $EC =< C, \beta, V >$ *consisting of: (1) The coordinates C of the coefficient; (2) A bitmap* β *of size M where the* i^{th} *bit denotes the existence or absence of a coefficient value for the* i^{th} *measure; and, (3) The set of stored coefficient values V.* ∎

The bitmap of an extended wavelet coefficient determines exactly which of the (at most M) per-measure values of the combined coefficient at the given coordinates C have actually been stored. Thus, an extended wavelet coefficient combines the positive aspects of both the *Individual* and *Combined* algorithms as a flexible storage method that can store anywhere from one to M values for any combination of coefficient coordinates. (We refer to the (coordinates, bitmap) pair for an extended wavelet coefficient as the coefficient's *header*.)

In the remainder of the chapter, we address the problem of building effective extended wavelet coefficient synopses (under a given storage constraint) for different classes of target error metrics in the approximate data reconstruction. (Our development typically assumes that the unit of storage space is equal to the space needed to store a single coefficient value (e.g., size of a float denoted as `sizeof(float)`), and all space requirements/constraints are expressed in terms of this unit.) Since our focus is on selecting the specific coefficient values to store, our algorithms address only the final *thresholding* step of the wavelet-based data reduction process. The input to all of our thresholding schemes is the complete set of combined wavelet coefficients, which can be trivially created using the same decomposition process as in either the *Individual* or the *Combined* algorithm (Section 4.2.4).

4.3 Extended Wavelet Synopses for Weighted Sum-Squared Error

As discussed earlier, the most common optimization objective for conventional wavelet synopses in the single-measure case is the sum-squared (i.e., L_2) error in the data approximation. Thus, a natural extension for data sets with multiple measures is to optimize for a *weighted* sum-squared error across all data measures. More formally, our optimization problem can be stated as follows.

> **[Weighted Sum-Squared Error Minimization for Extended Coefficients]** Given a collection W_A of candidate combined wavelet coefficients of a D-dimensional data set with M measures, a storage constraint B, and an M-vector of measure weights $\overline{w}$, select a synopsis $\mathcal{S}$ of extended wavelet coefficients that minimizes the weighted sum of the L_2-error norms across all measures; that is, minimize $\sum_{j=1}^{M} \left(w_j \times \sum_i (d_{ij} - \hat{d}_{ij})^2 \right)$ subject to the constraint $\sum_{EC \in \mathcal{S}} |EC| \leq B$. ∎

Based on Parseval's theorem and the discussion in Section 4.2.3, and using c^*_{ij} to denote the normalized value for the j^{th} measure of the i^{th} input combined wavelet coefficient, we can restate the above optimization problem in the following equivalent (and, easier to process) form.

[Weighted Sum-Squared Benefit Maximization for Extended Coefficients] Given a collection W_A of candidate combined wavelet coefficients of a D-dimensional data set with M measures, a storage constraint B, and an M-vector of measure weights $\overline{w}$, select a synopsis $\mathcal{S}$ of extended wavelet coefficients that maximizes the weighted sum of the retained normalized squared coefficient values across all measures; that is, maximize $\sum_{EC=<C,\beta,V>\in\mathcal{S}} \sum_{j,\beta(j)=1} w_j \times (c_{ij}^*)^2$ subject to the constraint $\sum_{EC\in\mathcal{S}} |EC| \leq B$. ∎

4.3.1 DynProgL2: An Optimal Dynamic-Programming Algorithm

We now propose a thresholding algorithm (termed DynProgL2) based on Dynamic-Programming (DP) ideas, that optimally solves the optimization problem described above. Our DynProgL2 algorithm takes as input a set of combined coefficients W_A, a space constraint B, and an M-vector of weights $\overline{w}$ (used to weight the benefit of the coefficient values for each measure). DynProgL2 treats the individual coefficient values for each input combined coefficient as *subitems* in our problem, utilizing an implicit mapping that maps the j^{th} coefficient value of the i^{th} combined coefficient to the subitem index $k = (i-1)*M+j$. (Thus, the M per-measure values for each combined coefficient correspond to consecutive subitems.) Our discussion in this section makes use of this mapping in order to simplify the development of our algorithms. Finally, we should emphasize here that our DynProgL2 thresholding algorithm is applicable (without any modifications) independently of the data dimensionality, since increasing the dimensionality only affects the header size H for the retained extended wavelet

coefficients.

Let $k = (i-1) * M + j$ denote the subitem index corresponding to the j^{th} coefficient value of the i^{th} combined coefficient. Retaining the k^{th} subitem in our synopsis, gives us a weighted benefit equal to $w_j \times (c^*_{i,j})^2$. However, the space overhead for storing this subitem obviously depends on whether this is the *first* coefficient value being stored from the i^{th} combined coefficient, or not. If this is indeed the first value retained from the i^{th} combined coefficient, then its space requirements are equal to the space needed to store the extended coefficient header (coordinates and bitmap) plus the space for storing the coefficient value. On the other hand, if other subitems for the coefficient have already been stored, then we have already paid the space penalty for the coefficient header, and the subitem's space requirements are simply the space for storing the coefficient value. These dependencies on the storage-space requirements across coefficient values (due to the shared space for extended coefficient headers) render the design of an optimal solution to our problem significantly more complex than that of known (pseudo-polynomial) DP algorithms to traditional *knapsack-style* problems. (Note, of course, that in our problem scenario, the space bound B is always upper bounded by $|W_A|$ and, thus, is polynomial (linear) in the input size.)

We now try to formulate a DP recurrence for our optimization problem. Note that, to be able to tabulate partial solutions in our DP table, our development here requires that all subitems occupy an integral number of space units – this can be done, for instance, by assuming a basic space unit of 1 bit. [2] For the optimal solution

[2] Some techniques, like encoding the bitmap within some coefficient coordinate(s) for small numbers of measures, can help increase the size of the space unit, thus decreasing the memory requirements of our DP tables. Still, such optimizations do not improve the asymptotic space complexity of our problem.

using synopsis space of at most S, and considering the first k subitems, three cases may arise:

1. The optimal solution is the same as using $k-1$ subitems and the same space S;
2. The optimal solution is achieved by including subitem k, and k is the *first* subitem of its combined coefficient included in the optimal solution; or,
3. The optimal solution is achieved by including subitem k, and k is not the first subitem of its combined coefficient included in the optimal solution.

It is important to note that, in the third case, the k^{th} subitem needs to be combined with the optimal partial solution P that (a) uses at most the first $k-1$ subitems and space at most $S - \texttt{sizeof(float)}$; *and* (b) includes *at least one more subitem* (i.e., other than k) from the corresponding, i^{th} combined coefficient. This second requirement results in a perhaps surprising observation: The partial solution P is *not necessarily optimal* for our optimization problem when using only the first $k-1$ subitems and up to $S - \texttt{sizeof(float)}$ units of space. An example is shown in Table 4.5, which depicts just two coefficients corresponding to a three-dimensional data set with three measures. To keep things simple, assume that the storage bound B is equal to the size of one tuple augmented by a bitmap of three bits, and that each measure has a weight of 1. Note that, under this storage bound, it is obviously impossible to store two coefficient values from different coefficients. Let S_1 denote the space needed to store a single coefficient value along with the pertinent header information; that is, $S_1 = H + \texttt{sizeof(float)}$. Now, consider the optimal solution

Candidate Coefficients					
Coordinates			Values		
0	0	1	100	1	2
1	2	0	99	98	97

Considered Subitems	SubItems in Optimal Solution For Space Bound		
	S_1	$S_1 + \texttt{sizeof(float)}$	$S_1 + 2\texttt{sizeof(float)}$
First 1	1	1	1
First 2	1	1,2	1,2
First 3	1	1,3	1,2,3
First 4	1	1,3	1,2,3
First 5	1	**4,5**	4,5
First 6	1	4,5	4,5,6

Table 4.5: Unexpected Optimal Solution Arises for Space Bound $S_1 + \texttt{sizeof(float)}$.

for space bound $S_1 + \texttt{sizeof(float)}$, when considering up to the first five subitems. It is easy to see that, at this point, the optimal solution is to store subitems 4 and 5 (both corresponding to the second combined coefficient); however, it is also easy to see that subitem 4 is not part of *any* optimal solution involving only the first four subitems (for any storage bound $\leq B$), since subitem 1 can always be used in its place to give a solution with larger benefit.

The above discussion basically shows that our optimization problem for extended wavelet synopses basically violates the key *principle of optimality* for conventional dynamic programming [CLR90], and requires us to come up with a novel algorithmic solution. An important observation here is that we only need to store, for each possible subitem×space (k, S) combination only a single suboptimal solution, namely the best solution (for at most S units of space and considering up to the k^{th} subitem) which *forces* at least one subitem of the combined coefficient corresponding to k to be included in the synopsis. Our dynamic program employs an array FORCE$[k, S]$ to tabulate such suboptimal solutions (for each subitem×space combination), in addition to the more conventional OPT$[k, S]$ DP array which tabulates the

(partial) optimal solutions to our problem (when using space $\leq S$ and considering the first k subitems).

Our DynProgL2 algorithm (depicted in Figure 4.3) computes the entries for the OPT$[k, S]$ and FORCE$[k, S]$ arrays (both of size $(N \cdot M) \times B$) in a mutually-recursive manner. Each cell of these two arrays comprises two numeric fields: (1) a "benefit" field recording the total benefit for the corresponding partial solution, and (2) a "choice" field used to code the choice made by our dynamic program when deciding the benefit of a cell (to be explained shortly). (Both fields are necessary in order to retrace the actions of our algorithm when building the optimal solution.) DynProgL2 begins by initializing some entries for both the OPT and FORCE arrays: Lines 1–3 are based on the fact that no coefficient value can be stored in space less than $H +$ `sizeof(float)`. Similarly, the optimal solution for space of at least $H +$ `sizeof(float)` and considering only the first subitem obviously only includes this subitem (Lines 4–6).

DynProgL2 then iteratively fills in the values for the remaining cells (Lines 7–17). For the OPT array, the benefit for the optimal solution using space $\leq S$ and considering up to the first k subitems, is computed as the best (i.e., maximum-benefit) choice from the three cases described above (Lines 12–13). A similar choice is made for the corresponding best solution for the FORCE array entries (Lines 14–15). Note that some of the three cases are valid only if the current subitem satisfies some conditions, namely that it corresponds to a coefficient value with a measure index of at least two (i.e., $j > 1$). The "choice" field of our DP array entries, which codes the choice made when determining the benefit of an entry, is assigned a value of 2, 3, or

procedure DynProgL2(W_A, B, $\overline{w}$)
Input: $N \times M$ vector of combined wavelet coefficients W_A; space constraint B;
per-measure weight vector $\overline{w}$.
Output: Optimal set of extended wavelet coefficients and benefit of optimal solution.

1. **for each** $S < H +$ `sizeof(float)` **do** // initialize DP array entries
2. OPT$[*, S]$.benefit := FORCE$[*, S]$.benefit := 0
3. OPT$[*, S]$.choice := FORCE$[*, S]$.choice := 1
4. **for each** $S \geq H +$ `sizeof(float)` **do**
5. OPT$[1, S]$.benefit := FORCE$[1, S]$.benefit := $w_1 * (c^*_{1,1})^2$
6. OPT$[1, S]$.choice := FORCE$[1, S]$.choice := 3
7. **for** $k := 2$ **to** $N \cdot M$ **do**
8. let $i := 1 + (k-1) \div M$ // combined coefficient index
9. let $j := 1 + (k-1) \mod M$ // measure index
10. let $f :=$ `sizeof(float)` // space for a single coefficient value
11. **for** $S := H + f$ **to** B **do**
12. OPT$[k, S]$.benefit := $\max \begin{cases} \text{OPT}[k-1, S].\text{benefit} & \\ \text{OPT}[k-1, y-H-f].\text{benefit} + w_j * (c^*_{i,j})^2 & \\ \text{FORCE}[k-1, y-f].\text{benefit} + w_j * (c^*_{i,j})^2 & j > 1 \end{cases}$
13. Depending on which of the three choices listed above produced the maximum benefit value, set OPT$[k, S]$.choice equal to 2, 3 or 4 (respectively)
14. FORCE$[k, S]$.benefit := $\max \begin{cases} \text{FORCE}[k-1, S].\text{benefit} & j > 1 \\ \text{OPT}[k-1, y-H-f].\text{benefit} + w_j * (c^*_{i,j})^2 & \\ \text{FORCE}[k-1, y-f].\text{benefit} + w_j \times (c^*_{i,j})^2 & j > 1 \end{cases}$
15. Depending on which of the three choices listed above produced the maximum benefit value, set FORCE$[k, S]$.choice equal to 2, 3 or 4 (respectively)
16. **endfor**
17. **endfor**
18. Build the optimal solution by doing a reverse traversal starting from the entry OPT$[N \cdot M, B]$, and moving based on the choice field of the current entry
19. **return**(OPT$[N \cdot M, B]$.benefit) // return benefit of optimal synopsis

end

Figure 4.3: The Optimal DynProgL2 Algorithm.

4 (respectively), depending on which of the three above-described cases produced the optimal solution. Entries corresponding to cases where no subitem can be stored in the specified space have a "choice" field value of 1 (Line 3).

At the end, the total benefit for the optimal extended wavelet synopsis is the

benefit achieved when considering all the $N \cdot M$ subitems and using at most B space units, i.e., OPT$[N \cdot M, B]$.benefit. We can build the optimal solution by retracing the actions of DynProgL2, starting from cell $[N \cdot M, B]$ and moving depending on the "choice" field of the current cell. More formally, our optimal synopsis construction process starts by initializing the synopsis $\mathcal{S}$ to ϕ and, assuming that at some point we are at cell $[k, S]$, the action performed is determined based on the value of the cell's "choice" field as follows:

- choice = 1: End of traversal.
- choice = 2: Move to cell $[k-1, S]$ of the same array.
- choice = 3: Add an extended wavelet coefficient containing the k^{th} subitem to $\mathcal{S}$, and move to cell $[k-1, S-H-$ `sizeof(float)`$]$ of the OPT array.
- choice = 4: Add the k^{th} subitem to $\mathcal{S}$ (creating a new extended wavelet coefficient, if necessary), and move to cell $[k-1, S-$ `sizeof(float)`$]$ of the FORCE array.

Time and Space Complexity. The space requirements of our DynProgL2 algorithm are essentially determined by the size of the OPT and FORCE arrays, which is $O(NMB)$. Given that the value of each cell is computed in constant O(1) time for both of our DP arrays, the overall time complexity of DynProgL2 is also $O(NMB)$. Our reverse-traversal procedure for constructing the optimal extended wavelet synopsis takes $O(NM)$ time, since each step essentially checks (in constant time) whether a subitem k belongs in the synopsis and then proceeds to subitem $k-1$.

4.3.2 GreedyL2: An Efficient, Provably Near-Optimal Approximation Algorithm

We now present a greedy solution to the optimization problem of Section 4.3. Our algorithm, to which we will refer as GreedyL2, is based on transforming the optimization problem to match the 0-1 Knapsack Problem, and then selecting which coefficient values to store based on a *per space benefit* metric. The notation used in this section is consistent with the one described in Table 4.1. However, we revert to our original definition of the space unit to be equal to the size of storing one coefficient value (i.e., equal to `sizeof(float)`). Note that the GreedyL2 algorithm, similarly to the DynProgL2 algorithm, can be applied without any modifications, independently of the data dimensionality, since the increased dimensionality simply affects the size of the header of any stored extended wavelet coefficients.

Similar to the dynamic programming algorithm presented in the previous section, GreedyL2 receives as input a set of candidate *combined* wavelet coefficients W_A, a set of weights $\overline{w}$, and a storage constraint B. Instead of considering the benefit of each coefficient value individually, GreedyL2 considers at each step the optimal benefit achieved by selecting a set of k ($1 \leq k \leq M$) coefficient values of the same *combined* coefficient that have not already been stored. It easy to see that the optimal selection will include the non-stored coefficient values that have one of the k largest benefits: $w_j \times (c_{ij}^*)^2$, where w_j is the weight corresponding to the coefficient value, and c_{ij}^* is its normalized value. The storage space for these k values will be equal to $H + k$, if no value of this *combined* coefficient has been stored before, and k otherwise. GreedyL2

procedure GreedyL2(W_A, B, W)
Input: $N \times M$ vector of combined wavelet coefficients W_A; space constraint B;
per-measure weight vector $\overline{w}$.
Output: Selected set of extended wavelet coefficients and benefit of greedy solution
1. A max-heap structure *struct* is used to maintain the optimal benefits of the candidate sets of coefficient values.
2. Each entry in *struct* has 4 fields:
   psb: per space benefit
   num: number of coefficient values in candidate set
   space: space needed for storing the set's coefficient values
   index: index of combined coefficient the set belongs to
3. $Stored[i]$ denotes the number of coefficient values from the i-th input combined coefficient that have already been selected to be stored.
4. **for** $i := 1$ **to** N **do**
5. Sort coefficient values in descending order of their weighted benefit $w_j \times (c^*_{ij})^2$
6. Set $SortOrder[i, j]$ to the index of the measure with the j-th top weighted benefit
7. $Stored[i] := 0$
8. InsertSets($i, struct, Stored, SortOrder, \overline{w}$)
9. **endfor**
10. SpaceLeft := B
11. **while** ($SpaceLeft \geq 1$) AND ($struct.size() > 0$) **do**
12. **repeat** $PickedSet = struct.pop()$ **until** $PickedSet.space \leq SpaceLeft$
13. Also remove all candidate sets of the combined coefficient $PickedSet.index$
14. SpaceLeft -= $PickedSet.space$
15. $Stored[PickedSet.index] += PickedSet.num$
16. Remove from *struct* all sets belonging to coefficient $PickedSet.index$
17. InsertSets($PickedSet.index, struct, Stored, SortOrder, \overline{w}$)
18. **endwhile**
19. For each *combined* coefficient store the Stored coefficient values with the largest weighted benefit
end

Figure 4.4: The GreedyL2 Algorithm

maintains a structure with all the optimal sets of size k ($1 \leq k \leq M$) of all the *combined* coefficients, and selects the set with the largest per space benefit. The coefficient values belonging to the selected set are stored, and the benefits of the optimal sets for the chosen *combined* coefficient have to be recalculated to only consider values that have not already been stored. The algorithm is presented in Figure 4.4.

For each input *combined* coefficient, the first step is to decide the sort order of its coefficient values based on their weighted benefit (Lines 5-6). For each *combined* coefficient we also maintain the number of its coefficient values that have been selected

procedure InsertSets(i, *struct*, *Stored*, *SortOrder*, $\overline{w}$, *SpaceLeft*)
Input: Index i of input combined coefficient; structure *struct* of candidate sets;
Symbols *Stored*, *SortOrder*, $\overline{w}$ and *SpaceLeft* defined as in GreedyL2 algorithm
Output: Candidate sets inserted in *struct* structure
begin
1. $cumulativeBen := 0$
2. **for** $j := Stored[i] + 1$ **to** M **do**
3. p = $SortOrder[i, j]$
4. $cumulativeBen += w_p \times (c^*_{ip})^2$
5. **if** $(Stored[i] > 0)$ **then**
6. $spaceNeeded = j - Stored[i]$
7. **else**
8. $spaceNeeded = H + j$
9. **if** $(spaceNeeded < SpaceLeft)$ **AND** $((c^*_{ip})^2 > 0)$ **then**
10. struct.insert(candidateSet($\frac{cumulativeBen}{spaceNeeded}$, j, $spaceNeeded$, i))
11. Also connect all inserted sets of same combined coefficient using a cyclic list
12. **endfor**
end

Figure 4.5: The InsertSets Subroutine

for storage in a *Stored* array, whose entries are initialized to 0 at the beginning of the algorithm (Line 7). Due to the way our algorithm is formulated, we do not need to remember which of the coefficient's values have been selected for storage, since these will always be the ones with its *Stored*[] maximum weighted benefits. We then calculate the optimal benefits of sets containing k coefficient values, $1 \leq k \leq M$ (Line 8). The maximum number of such sets is at most M, but not always equal to M, since we do not need to create any sets that include any coefficient values with zero benefit. The space needed to store each of these k sets is $H + k$. The per space benefit of each set, along with its occupied space, the number of coefficient values within that set, and the identifier of the coefficient it belongs to, are then inserted in a max-heap structure, where its elements are ordered based on their per-space benefit. We chose to use such a structure, since each of the insert, delete and finding the maximum value operations has at most logarithmic cost. However, any other

data structure with similar characteristics can be used in its place.

The algorithm then repeatedly (Lines 11-18) picks the set with the maximum per space benefit that can fit in the remaining space. The values corresponding to this set are uniquely identified by the identifier field of the corresponding *combined* coefficient (stored in the *index* field of each set), its *Stored* variable, and the size of the picked set. For the corresponding combined coefficient of the picked set, the optimal benefits of its sets have to be recalculated to include only non-stored coefficient values. This coefficient's previous sets are removed from the tree and the newly calculated ones are then inserted. Note that the space required for the newly inserted sets does not include the size of the header, since this has already been taken into account. The entire procedure terminates when no set can be stored without violating the storage constraint ($SpaceLeft < 1$). In order to create the output *extended* coefficients, we simply have to parse the list of the *combined* coefficients, and for any coefficient that has a $Stored[]$ value greater than 0, create an *extended* wavelet coefficient and store in it its $Stored[]$ coefficient values with the largest weighted benefits.

Theorem 1: *The* GreedyL2 *algorithm has an approximation ratio bound of* $\min\{2, 1 + \frac{1}{\frac{B}{H+M}-1}\}$ ∎

Proof: The proof is similar to the corresponding proof for the 0-1 knapsack problem. A significant observation is that whenever we select a set $PickedSet$ of coefficient values from a *combined* coefficient *Coeff* for storage, any candidate set *subOpt* of *Coeff* that will later be inserted in the max-heap for consideration cannot have a larger per space benefit than the one of $PickedSet$. We will prove this by contradiction. Assume

that *subOpt* has a larger per space benefit than *PickedSet*. By the way the candidate sets are formed, the following observations hold:

1. The sets *PickedSet* and *subOpt* cannot share any coefficient values.

2. The largest benefit of a coefficient value of *subOpt* cannot be larger than the smallest benefit of a coefficient value of *PickedSet*.

3. The space overhead of *subOpt* does not include the size of the header, while for *PickedSet* this depends on whether it is the first set of *Coeff* selected for storage.

If we depict the benefits of the coefficient values of *PickedSet* as $v_1, v_2, \ldots, v_k$ and the benefits of the coefficient values of *subOpt* as $v'_1, v'_2, \ldots, v'_p$, then the per space benefits of the two sets are, correspondingly, $\frac{\sum_{i=1}^{k} v_i}{\delta \times H + k}$ and $\frac{\sum_{i=1}^{p} v'_i}{p}$, where δ has a value of 1 or 0, depending on whether *PickedSet* is the first set of *Coeff* selected for storage. Since by hypothesis *subOpt* has a larger per space benefit than *PickedSet*:

$$\frac{\sum_{i=1}^{p} v'_i}{p} > \frac{\sum_{i=1}^{k} v_i}{\delta \times H + k} \Longrightarrow \sum_{i=1}^{p} v'_i \times (\delta \times H + k) > \sum_{i=1}^{k} v_i \times p \tag{4.1}$$

At the time *PickedSet* was selected, a candidate set *notPicked* of *Coeff* with $k + p$ subitems existed, having a benefit which is at least equal to the set *union* containing all subitems in *PickedSet* and *subOpt*. Comparing the per space benefit

of $notPicked$ and $PickedSet$, we have:

$$benefit(notPicked) - benefit(PickedSet) \geq benefit(union) - benefit(PickedSet)$$
$$= \frac{\sum_{i=1}^{k} v_i + \sum_{i=1}^{p} v'_i}{\delta \times H + (k+p)} - \frac{\sum_{i=1}^{k} v_i}{\delta \times H + k} = \frac{\sum_{i=1}^{p} v'_i \times (\delta \times H + k) - \sum_{i=1}^{k} v_i \times p}{(\delta \times H + (k+p)) \times \delta \times H + k} > 0 \tag{4.2}$$

The last part of formula (2) follows immediately from the inequality of formula (1). At this point we have reached a contradiction, since $PickedSet$ should not have a smaller per space benefit than the set $notPicked$. Therefore, *subOpt* cannot have a larger per space benefit than $PickedSet$.

The above observation also implies that each candidate set inserted in the max-heap after the first set selection made by the algorithm cannot have a larger per space benefit than the ones that have already been selected. To prove this, consider any such set $nowInserted$ that is inserted in the max-heap following the selection for storage of another set *nowStored*, of the same candidate combined coefficient *Coeff*, and consider the following two observations:

1. Following the preceding proof, the set $nowInserted$ cannot have a larger per space benefit than any set already picked for storage from the same candidate combined coefficient *Coeff*.

2. Consider any candidate set $otherSet$ already selected for storage, which corresponds to a candidate combined coefficient other than *Coeff*. At the moment $otherSet$ was selected for storage, the candidate set of *Coeff* with the largest per space benefit that was at that time in the max-heap could not have a larger

per space benefit than $otherSet$, since it would have been selected for storage instead of it, and cannot have a smaller per space benefit than $nowStored$.

Now, consider that GreedyL2 solution has selected to store the sets $S_1, S_2, \ldots, S_l$, and that S_{l+1} is the set with the largest per space benefit that cannot be stored due to space constraints.[3] Let $BenStored = \sum_{i=1}^{l} S_i.psb \times S_i.space$ denote the sum of benefits of the l sets included in the solution (using the notation of Figure 4.4), $BenFraction = S_{l+1}.psb \times S_{l+1}.space$ denote the benefit of set S_{l+1} and $BenOptimal$ denote the benefit of the optimal solution. If the remaining storage space at this point of our algorithm is $SpaceLeft$, it can easily be shown [4] that the optimal solution has at most benefit equal to: $BenStored + BenFraction \times \frac{SpaceLeft}{S_{l+1}.space}$.

Obviously, $S_l.psb \leq \frac{BenStored}{B-SpaceLeft} \leq \frac{BenStored}{\max\{H+1, B-(H+M)\}}$ (since the space of the stored sets must be at least $H+1$ and at least equal to $B-(H+M)$), and, since the per space benefit of S_{l+1} cannot be larger than the one of S_l, $S_{l+1}.psb \leq S_l.psb \Rightarrow BenFraction \times \frac{SpaceLeft}{S_{l+1}.space} \leq (H+M) \times S_l.psb \leq \frac{BenStored \times (H+M)}{\max\{H+1, B-(H+M)\}}$. Therefore,

$$\begin{aligned} BenOptimal &\leq BenStored + BenFraction \times \frac{SpaceLeft}{S_{l+1}.space} \\ &\leq BenStored \times (1 + \frac{H+M}{\max\{H+1, B-(H+M)\}}) \end{aligned}$$

For $B \geq 2H + M + 1$, the proof of the approximation ratio bound is complete. For $B < 2H + M + 1$, the solution Z = $\max\{BenFraction, BenStored\}$ has at least half the benefit of the optimal solution, since:

[3] For simplicity, consider that S_{l+1} could fit by itself within the original storage constraint.

[4] The proof is identical to the optimal proof for the fractional knapsack problem.

$$BenOptimal \leq BenFraction + BenStored \leq 2 \times max\{BenFraction, BenStored\}$$

∎

The following lemma can now be easily derived based on the above theorem.

Lemma 2: *Let $B - (H + M) < B_{used} \leq B$ denote the space occupied by the* GreedyL2 *algorithm's solution at the time the first PickedSet that requires space larger than $SpaceLeft = B - B_{used}$ is selected. The* GreedyL2 *algorithm always selects the optimal solution for the space constraint B_{used}.* ∎

Proof: Using the notation of the above proof, for a space constraint B_{used} there is no unused space (i.e. $SpaceLeft' = 0$). From the above proof we can then infer that $BenFraction = 0$, resulting in $BenOptimal \leq BenStored + BenFraction = BenStored$. ∎

Time and Space Complexity. Each of the N input *combined* coefficients creates at most M candidate sets. Therefore, the space for the max-heap is $O(NM)$. For each combined coefficient, maintaining the sort order requires $O(M)$ space. The size of the input *combined* coefficients is $O(N(D + M))$, making the overall space complexity of the algorithm $O(N(D + M))$.

Determining the sort order for the values of each *combined* coefficient requires time $O(M \log M)$. Calculating the benefits of the sets produced by each coefficient then takes only $O(M)$ time. The original construction of a max-heap with $O(NM)$ elements can be done in $O(NM)$ time. Thus, the overall running time for the max-

heap construction is $O(NM \log M)$. Each time a set is picked for inclusion in the result, the search requires $O(\log(NM))$ time. Then, we need to make $O(M)$ deletions from the max-heap, corresponding to all the sets of the chosen *combined* coefficient. Finding all such nodes on the tree requires $O(M)$ time, if they are connected by a cyclic list. Note that all the sets of the same *combined* coefficient are created at the same time, thus making it easy to create such a list. Each of the $O(M)$ insertion and deletion operation then requires $O(\log(NM))$ time. Since at most $O(\frac{M \times B}{H+M})$ sets can be picked (for each extended wavelet coefficient, the smallest average space per stored coefficient value occurs when all the M coefficient values are stored), the total time complexity should be $O(NM \log M + \frac{BM}{H+M} \times Mlog(NM))$. However, a small complication arises because the algorithm may at some point repeatedly select candidate sets that do not fit within the remaining space. Since at most $O(NM)$ such sets may be selected and removed, the running time cost of this step might dominate the algorithm's running time. However, at this step, the algorithm may deviate from its behavior of removing the candidate set with the maximum per space benefit and scan the entire heap for the set with the maximum per space benefit that fits within the remaining space $SpaceLeft$. Since the maximum value of the remaining space when this occurs must be less than $H + M$, and since the minimum size of each candidate set is 1, at most $O(H + M)$ steps with linear search cost may be incurred. This results in a total running time complexity of $\mathrm{O}(NM(H + M) + \frac{BM}{H+M} \times Mlog(NM))$.

Note that according to Theorem 1 and Lemma 2, the solution of the GreedyL2 algorithm at the time the first set that does not fit within the space bound is selected

is of good quality. To further improve the running time of the algorithm, one may ignore the small unused space and thus always only insert at most one candidate set for each combined coefficient, namely the set with the maximum per space benefit, since this is the only one that may be selected for inclusion in the final solution. This reduces the space of the max-heap to $O(N)$ and requires $O(1)$ deletion and insertion operations for each picked set, while maintaining the same tight approximation ratio bound. Thus, the running time is improved in this case to $O(NM \log M + \frac{BM}{H+M} \log N)$.

4.4 Extended Probabilistic Wavelet Synopses for Relative Error

Unfortunately, conventional and extended wavelet synopses optimized for overall L_2 error metrics (as described in Sections 4.2-4.3), may not always be the best choice for approximate query processing systems. As observed in the recent work of Garofalakis and Gibbons [GG04], conventional L_2-optimized wavelet synopses suffer from several important problems, including the introduction of severe bias in the data reconstruction and wide variance in the quality of the data approximation, as well as the lack of non-trivial guarantees for individual approximate answers. To address these shortcomings, their work introduces *probabilistic wavelet synopses*, a novel approach for constructing (single-measure) wavelet data summaries optimized for *maximum relative-error metrics* in the approximate data reconstruction. Given the pitfalls and shortcomings of synopses optimized for overall L_2 errors [GG04], it is obviously impor-

tant to extend the ideas of (relative-error optimized) probabilistic wavelet synopses to the setting of *multi-measure* data and extended wavelet coefficients. This turns out to be a challenging problem, mandating novel algorithmic solutions. Before discussing the details of our approach, however, we provide some necessary background material on (single-measure) probabilistic wavelets [GG04]. (To simplify the exposition, our development here focuses primarily on the one-dimensional case; extensions to multi-dimensional wavelets are described in Section 4.4.5.)

4.4.1 Probabilistic Wavelet Synopses for Single-Measure Data

Consider the wavelet-transform array W_A containing the wavelet coefficients for an input data vector. Rather than deterministically retaining the largest coefficients (in absolute normalized value) in a data summary, a *probabilistic wavelet synopsis* is constructed using a probabilistic thresholding process based on *randomized rounding* [MR95]. In a nutshell, the basic idea in to randomly round each coefficient either up to a larger *rounding* value (i.e., coefficient is retained) or down to zero (i.e., coefficient is dropped), so that the value of each coefficient is correct *on expectation*. More formally, each non-zero wavelet coefficient c_i is associated with a *rounding value* λ_i such that $0 < \frac{c_i}{\lambda_i} \leq 1$, and the value of coefficient c_i in the synopsis becomes a random variable $C_i \in \{0, \lambda_i\}$, where,

$$C_i = \begin{cases} \lambda_i & \text{with probability } \frac{c_i}{\lambda_i} \\ 0 & \text{with probability } 1 - \frac{c_i}{\lambda_i}. \end{cases}$$

In other words, a probabilistic wavelet synopsis essentially "rounds" each non-zero wavelet coefficient c_i *independently* to either λ_i or zero by flipping a biased coin with success probability $\frac{c_i}{\lambda_i}$. Note that the above rounding process is *unbiased*; that is, the expected value of each rounded coefficient is $\mathrm{E}[C_i] = \lambda_i \cdot \frac{c_i}{\lambda_i} + 0 \cdot (1 - \frac{c_i}{\lambda_i}) = c_i$, i.e., the actual coefficient value. Thus, since each data value can be reconstructed as a simple linear combination of wavelet coefficients (Section 4.2), and by linearity of expectation, it is easy to see that probabilistic wavelet synopses guarantee unbiased approximations of individual data values as well as range-aggregate query answers [GG02].

A different way to view the above probabilistic thresholding process is as an assignment of *fractional storage* $y_i \in (0, 1]$ to each non-zero coefficient c_i, where $y_i = \frac{c_i}{\lambda_i}$ (the probability of retaining the coefficient in the synopsis). Given a set of rounding values $\{\lambda_i\}$ (and the corresponding fractional storage assignments $\{y_i\}$), the variance of the value in the probabilistic wavelet synopsis for each coefficient $c_i \neq 0$ is

$$\mathrm{Var}(i, y_i) = \mathrm{Var}(C_i) = (\lambda_i - c_i) \cdot c_i = \frac{1 - y_i}{y_i} \cdot c_i^2 \tag{4.3}$$

and the expected size of the synopsis is simply $\mathrm{E}[|\text{synopsis}|] = \sum_{i|c_i \neq 0} y_i = \sum_{i|c_i \neq 0} \frac{c_i}{\lambda_i}$.

Garofalakis and Gibbons [GG04] propose several different algorithms for building probabilistic wavelet synopses. The key, of course, is to select the coefficient rounding values $\{\lambda_i\}$ such that some desired error metric for the data approximation is minimized while not exceeding a prescribed space limit B for the synopsis (i.e., $\mathrm{E}[|\text{synopsis}|] \leq B$). Their winning strategies are based on formulating appropri-

ate *Dynamic-Programming (DP)* recurrences over the Haar error-tree that explicitly minimize either (a) the maximum normalized standard error (MinRelVar), or (b) the maximum normalized bias (MinRelBias), for each reconstructed value in the data domain. As explained in [GG04], the rationale for these probabilistic error metrics is that they are directly related to the *maximum relative error* (with an appropriate *sanity bound* $\mathtt{S}$)[5] in the approximation of individual data values based on the synopsis; that is, both the MinRelVar and MinRelBias schemes try to (probabilistically) control the quantity $\max_i\{\frac{|\hat{d}_i - d_i|}{\max\{d_i, \mathtt{S}\}}\}$, where $\hat{d}_i$ denotes the data value reconstructed based on the wavelet synopsis. Note, of course, that $\hat{d}_i$ is again a *random variable*, defined as the ± 1 summation of all (independent) coefficient random variables on $\mathtt{path}(d_i)$. Bounding the maximum relative error in the approximation also allows for meaningful *error guarantees* to be provided on reconstructed data values [GG04].

As an example, Equation (4.4) depicts the DP recurrence in [GG02, GG04] for minimizing the maximum Normalized Standard Error (NSE) in the data reconstruction, defined as

$$\max_i \mathrm{NSE}(\hat{d}_i) = \max_i \frac{\sqrt{\mathrm{Var}(\hat{d}_i)}}{\max\{|d_i|, \mathtt{s}\}},$$

where $\mathrm{Var}(\hat{d}_i) = \sum_{c_j \in \mathtt{path}(d_i)} \mathrm{Var}(j, y_j)$. $R[i, B]$ here denotes the minimum value of the *squared* NSE (i.e., NSE^2) among all data values in the subtree of the error-tree rooted at coefficient c_i assuming a space budget of B, and $\mathrm{Norm}(i) = \max\{d^2_{\min_i}, \mathtt{S}^2\}$, where $d_{\min_i}$ is the minimum data value under c_i's subtree, is a normalization term for that subtree. (Indices $2i$ and $2i+1$ in the recurrence correspond to the left and

[5]The role of the sanity bound is to ensure that relative-error numbers are not unduly dominated by small data values [HS92, VW99].

right child (respectively) of c_i in the error-tree structure (Figure 4.1).) Intuitively, the DP recurrence in Equation (4.4) states that, for a given space budget B at c_i, the optimal fractional-storage allotments $\{y_k\}$ and the corresponding maximum NSE^2 are fixed by minimizing the larger of the costs for paths via c_i's two child subtrees (including the root in all paths), where the cost for a path via a subtree is the sum of: (1) the variance penalty incurred at c_i itself, assuming a setting of y_i, divided by the normalization term for that subtree, and (2) the optimal cost for the subtree, assuming the given space budget. This minimization, of course, is over all possible values of y_i and, given a setting of y_i, over all possible allotments of the remaining $B - y_i$ space "units" amongst the two child subtrees of c_i. Of course, if $c_i = 0$ then no space budget needs to be allocated to node i, which results in the simpler recurrence in the second clause of Equation (4.4). Finally, data-value nodes (characterized by indices $i \geq N$, see Figure 4.1) cost no space and incur no cost, and the "otherwise" clause handles the case where we have a non-zero coefficient but zero budget ($c_i \neq 0$ and $B = 0$).

$$R[i,B] = \begin{cases} \min\limits_{\substack{y_i \in (0,\min\{1,B\}];\\ b_L \in [0,B-y_i]}} \left\{ \max \left\{ \begin{array}{l} \frac{\mathrm{Var}(i,y_i)}{\mathrm{Norm}(2i)} + R[2i, b_L]\,, \\ \frac{\mathrm{Var}(i,y_i)}{\mathrm{Norm}(2i+1)} + R[2i+1, B-y_i-b_L] \end{array} \right\} \right\} & \text{if } i < N, \\ & c_i \neq 0, \\ & \text{and } B > 0 \\ \min_{b_L \in [0,B]} \{ \max\{ R[2i, b_L]\,,\, R[2i+1, B-b_L] \} \} & \text{if } i < N \\ & \text{and } c_i = 0 \\ 0 & \text{if } i \geq N \\ \infty & \text{otherwise} \end{cases} \tag{4.4}$$

As demonstrated by Garofalakis and Gibbons [GG02, GG04], the DP recurrence in Equation (4.4) characterizes the optimal solution to the maximum NSE min-

imization problem for the case of *continuous* fractional-storage allotments $y_i \in (0, 1]$ (modulo certain technical conditions that may require small "perturbations" of zero coefficients [GG02, GG04]). A similar DP recurrence can also be given for the maximum normalized bias metric. Their MinRelVar and MinRelBias algorithms then proceed by *quantizing the solution space*; that is, they assume the storage allotment variables y_i and b_L in Equation (4.4) to take values from a discrete set of choices corresponding to integer multiples of $1/\mathtt{q}$, where $\mathtt{q} > 1$ is an input integer parameter to the algorithms. (Larger values of $\mathtt{q}$ imply results closer to the optimal, continuous solution.) The running time of their (quantized) MinRelVar and MinRelBias algorithms is $O(N\mathtt{q}^2 B \log(\mathtt{q}B))$ with an overall space requirement of $O(N\mathtt{q}B)$ (and an in-memory working-set size of $O(\mathtt{q}B \log N)$); furthermore, their techniques also naturally extend to multi-dimensional data and wavelets, with a reasonable increase in time and space complexity [GG04]. Experimental results over synthetic and real-life data in [GG02, GG04] have demonstrated the superiority of MinRelVar and MinRelBias probabilistic synopses as an approximate query answering tool over conventional wavelet synopses.

4.4.2 Extended Probabilistic Wavelets for Multiple Measures: Problem Formulation

In what follows, we introduce algorithms for building effective probabilistic synopses comprising extended wavelet coefficients for multi-measure data sets. As in [GG04], our primary focus is on synopses that minimize the maximum relative error (with

appropriate sanity bounds) in the data reconstruction.[6] Employing the more complex extended-coefficient format enables effective space utilization, but, at the same time, significantly increases the complexity of the probabilistic-thresholding process, rendering the DP schemes of [GG04] inapplicable in our problem setting.

The Problem with Extended Coefficients. In a nutshell, the key difficulty in probabilistic thresholding for extended wavelet coefficients stems from the common header space (i.e., coordinates + bitmap) for all stored coefficient values. The ability to share this header is the main benefit of the extended coefficient storage format but, at the same time, this sharing of storage introduces non-trivial dependencies in the thresholding process across coefficients for different measures, and implies that the selection probabilities for such coefficients are no longer independent. More formally, consider a data set with M measures, and let c_{ij} denote the Haar coefficient value corresponding to the j^{th} measure at coordinate i, and let y_{ij} denote the retention probability (i.e., fractional storage) for c_{ij} in the synopsis. Also, let EC_i be the extended wavelet coefficient at coordinate i, and let H denote the space required by an extended-coefficient header. (In our discussion, the unit of space is set equal to the space required to store a single coefficient value (e.g., size of a float), and all space requirements are expressed in terms of this unit.) The expected space requirement of extended coefficient EC_i can be computed as

$$\mathrm{E}[|EC_i|] = \sum_{j|c_{ij}\neq 0} y_{ij} + H \times (1 - \prod_{j=1}^{M} (1 - y_{ij})). \tag{4.5}$$

[6] Our techniques also naturally extend to other approximation-error metrics, including maximum weighted relative error and maximum absolute error.

The first summand in the above formula captures the expected space for all (non-zero) individual coefficient values at coordinate i. The second summand captures the expected header overhead. To see this, note that if at least one coefficient value is stored, then a header space of H must also be allotted. And, of course, the probability of storing ≥ 1 coefficient values is just one minus the probability that none of the coefficients is stored.

Equation (4.5) clearly demonstrates that the sharing of header space amongst the individual coefficient values c_{ij} for different measures creates a fairly complex dependency of the overall extended-coefficient space requirement on the individual retention probabilities y_{ij}. Given a space budget B for the wavelet synopsis, exploiting header-space sharing and this storage dependency across different measures is crucial for achieving effective storage utilization in the final synopsis. Essentially, this implies that our probabilistic-thresholding strategies for allocating synopsis space cannot operate on each measure individually; instead, space allocation must explicitly account for the storage dependencies across groups of coefficient values (corresponding to different measures). This requirement significantly complicates the design of probabilistic-thresholding schemes for extended wavelet coefficients.

Problem Statement and Approach. Our goal is to minimize the maximum relative reconstruction error for each individual data value; this would also allow us to provide meaningful *guarantees* on the accuracy of each reconstructed value. More formally, we aim to produce estimates $\hat{d}_{ij}$ of the data values d_{ij}, for each coordinate i and measure index j, such that $|\hat{d}_{ij} - d_{ij}| \leq \epsilon \cdot \max\{|d_{ij}|, \mathtt{s}_j\}$, for given per-measure sanity

bounds $\mathtt{s}_j > 0$, where the error bound $\epsilon > 0$ is minimized subject to the given space budget for the synopsis. Since probabilistic thresholding implies that $\hat{d}_{ij}$ is again a random variable, and using an argument based on the Chebyshev bound [GG02], it is easy to see that *minimizing the overall* NSE *across all measures* guarantees a maximum relative error bound that is satisfied *with high-probability*. Thus, we can define our probabilistic-thresholding problem for extended wavelet coefficients as follows.

[**Maximum** NSE **Minimization for Extended Coefficients**] Find the fractional-storage assignments y_{ij} for coefficients c_{ij} that *minimize* the maximum NSE for each reconstructed data value across all measures; that is,

$$\text{Minimize} \quad \max_{\substack{i\in\{0,\ldots,N-1\}\\ j\in\{1,\ldots,M\}}} \quad \frac{\sqrt{\mathrm{Var}(\hat{d}_{ij})}}{\max\{|d_{ij}|, \mathtt{s}_j\}} \tag{4.6}$$

subject to the constraints $0 < y_{ij} \leq 1$ for all non-zero c_{ij} and $\mathrm{E}[|\text{synopsis}|] = \sum_i \mathrm{E}[|EC_i|] \leq B$, where the expected size $\mathrm{E}[|EC_i|]$ of each extended coefficient is given by Equation (4.5). ∎

We focus on the above maximum NSE minimization problem for multi-measure data in the remainder of this section; we do note, however, that our techniques and algorithms also naturally extend to the multi-measure variants of the maximum normalized-bias minimization problems of [GG02, GG04]. Our algorithms exploit both the error-tree structure of the Haar decomposition and the above-described storage dependencies (Equation (4.5)) for extended coefficients in order to intelligently assign fractional storage $\{y_{ij}\}$ to non-zero coefficients within the overall space-budget

constraint B. As in [GG02, GG04], our schemes also rely on *quantizing* the space allotments to integer multiples of $1/\mathtt{q}$, where $\mathtt{q} > 1$ is an integer input parameter; that is, we modify the constraint $0 < y_{ij} \leq 1$ to $y_{ij} \in \{\frac{1}{\mathtt{q}}, \frac{2}{\mathtt{q}}, \ldots, 1\}$ in the above problem formulation. (Remember that our space unit corresponds to the size of a coefficient value.) Our first algorithm is based on an *exact*, generalized DP formulation that extends earlier schemes for the single-measure case [GG02, GG04] to the multi-measure setting; unfortunately, this generalization comes at the cost of a significant increase in computational complexity (as our empirical study also clearly shows). Our second algorithm is a very fast, greedy approximation heuristic (termed GreedyRel) for probabilistic multi-measure thresholding; our results show that GreedyRel consistently provides near-optimal performance and can easily scale to problem sizes that are simply unattainable for DP-based solutions (even for the simpler single-measure case!).

4.4.3 PODPRel: An Optimal Partial-Order Dynamic Programming Solution

Consider an input data set with M measures. At a high level, our maximum NSE minimization problem for extended wavelet coefficients (Section 4.4.2) can be seen as a generalization of the single-measure NSE-minimization setting of [GG04], where our final goal is to minimize *the maximum component of an M-component vector of* NSE*s* (i.e., the NSE values for the approximation of each individual measure). Of course, the key complication here is that, for a given synopsis space budget, these M per-measure NSE values are not independent and cannot optimized individually; this is, again, due

to the intricate storage dependencies that arise between the approximation at different measures because of the shared header space (Equation (4.5)). As already discussed in Section 4.4.2, it is crucial that our thresholding algorithms are able to exploit these dependencies to ensure effective synopsis-space utilization. This essentially implies that our thresholding schemes have to treat these M-component NSE vectors *as a unit* during the optimization process.

Consider once again the DP recurrence in Equation (4.4) for the single-measure case. Remember that the recurrence computes in $R[i, B]$ the minimum value of the NSE^2 among all data values under coefficient i in the error tree assuming a space budget of B. Based on our discussion above, extending the formulation to the case of multiple measures requires that we generalize $R[i, B]$ to denote an *M-component vector* of NSE^2 values corresponding to all M measures for the data values in the subtree rooted at the coefficient with coordinate i, and assuming a total space of B allotted to extended coefficients in that subtree. We similarly generalize Var and Norm in Equation (4.4) to the M-component vectors of per-measure variances $\text{Var}(i, y_{ij})$ and normalization values $\text{Norm}(i, j)$ at extended coefficient i (for a total allotment of y_i); also, addition and division operators denote the corresponding component-wise operations on the operand vectors. As in the single-measure case [GG04], we can simplify the minimization problem of Equation (4.6) by normalizing the variance value at each node with the normalization terms $\text{Norm}(i, j) = \max\{d^2_{\min_{ij}}, \mathtt{s}^2_j\}$ of its subtrees. Thus, we compute the j-th component of the $R[i, B]$ vector at node i for a given retention probability y_{ij} of the c_{ij} coefficient value, and solutions $R[2i, b_{2i}]$ and

$R[2i+1, b_{2i+1}]$ from the node's left and right subtrees as:

$$R[i,B]_j = \max \begin{cases} \frac{\text{Var}_{(i,y_{ij})}}{\text{Norm}_{(2i,j)}} + R[2i, b_{2i}][j] \\ \frac{\text{Var}_{(i,y_{ij})}}{\text{Norm}_{(2i+1,j)}} + R[2i+1, b_{2i+1}][j] \end{cases}$$

Our goal, of course, is to minimize the maximum component of the vector $R[\texttt{root}, B]$; that is, minimize $\max_{k=1,\ldots,M}\{R[\texttt{root}, B]_k\}$.

Unfortunately, this generalization of $R[i, B]$ to an M-component vector also implies that, to ensure optimality, the bottom-up computation of the DP recurrence can no longer afford to maintain just the locally-optimal partial solution for each subtree (as in the single-measure case). In other words, merely tabulating the $R[i, B]$ vector with the minimum max-component for each internal tree node and each possible space allotment is no longer sufficient – more information needs to be maintained and explored during the bottom-up computation. As a simple example, consider the scenario depicted in Figure 4.6 for the case $M = 2$. (Slightly abusing notation, we use $R[2i, B-y]$ and $R'[2i, B-y]$ to denote two possible NSE^2 vectors for space $B-y$ at node $2i$.) To simplify the example, assume that the right child of node i also gives rise to the exact same solution vectors $R[]$ and $R'[]$. (The figure also depicts the normalized-variance vector for the coefficient values at node i assuming a total space of y.) It is easy to see that, in this case, even though $R'[2i, B-y]$ is locally-suboptimal at node $2i$ (since its maximum component is larger than that of $R[]$), it gives a superior overall solution of $[1+2,\ 3+0.5] = [3,\ 3.5]$ at node i when combined with i's local variance vector.

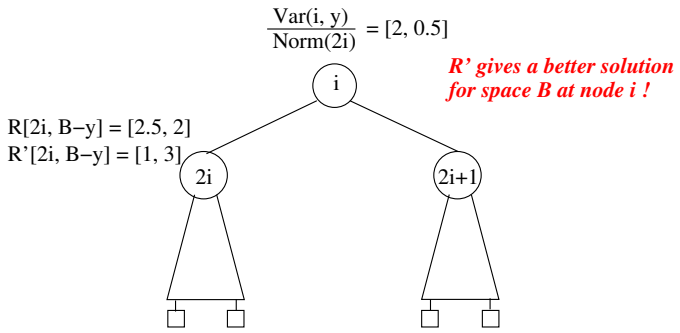


Figure 4.6: Example for partial-order pruning.

The key here is that, unlike conventional dynamic programming, using an M-component vector $R[i, B]$ to capture the per-measure squared NSEs corresponding to different partial solutions in the DP computation also means that the conventional *principle of optimality* based on a *total ordering of partial solutions* [CLR90] is no longer applicable. Thus, locally-suboptimal $R[i, B]$'s (i.e., with large maximum component NSE^{2}s) cannot be safely pruned since they may, in fact, be part of an optimal solution higher up in the tree. However, there does exist a safe pruning criterion based on a *partial ordering* of the $R[i, B]$ vectors defined through the *M-component less-than* operator $\preceq_M$, which is defined over M-component vectors u, v as follows:

$$u \preceq_M v \qquad \text{if and only if} \qquad u_i \leq v_i, \forall i \in \{1, \ldots, M\}.$$

For a given coordinate i and space allotment B, we say that a partial solution $R'[i, B]$ *is covered* by another partial solution $R[i, B]$ if and only if $R[i, B] \preceq_M R'[i, B]$ – it is easy to see that, in this case, $R'[i, B]$ can be safely pruned from the set of partial solutions for the (i, B) combination since, intuitively, $R[i, B]$ can always be used in its place to give an overall solution of at least as good quality.

Our proposed *Partial-Order Dynamic Programming (PODP)* solution to the maximum NSE minimization problem for extended coefficients (termed PODPRel) generalizes the corresponding DP formulation in [GG04, GG02] based on the above observations.[7] That is, our partial, bottom-up computed solutions $R[i, B]$ are M-component vectors of per-measure NSE^2 values for coefficient subtrees, and such partial solutions are only pruned based on the $\preceq_M$ partial order. Thus, for each coordinate-space combination (i, B), our PODPRel algorithm essentially tabulates *a collection* $\mathcal{R}[i, B]$ of incomparable solutions, that represent the "boundary points" of $\preceq_M$,

$$\mathcal{R}[i,B] = \{R[i,B] : \text{ for any other } R'[i,B] \in \mathcal{R}[i,B],$$
$$R[i,b] \not\preceq_M R'[i,B] \quad \text{and} \quad R'[i,b] \not\preceq_M R[i,B]\}.$$

Of course, for each allotment of space B to the coefficient subtree rooted at node i, PODPRel needs to iterate over all partial solutions computed in $\mathcal{R}[i, B]$ in order to compute the full set of (incomparable) partial solutions for node i's parent in the tree. Similarly, at leaves or intermediate root nodes, we consider all possible space allotments $\{y_{ij}\}$ to each individual measure and estimate the the overall space requirements of the extended coefficient using Equation (4.5).

The main drawback of our PODP-based solution is the dramatic increase in time and space complexity compared to the single-measure case. PODPRel relies on a much stricter, partial-order criterion for pruning suboptimal solutions which implies

[7] Ganguly et al. [GHK92] also discuss PODP in a completely different context, namely designing a System-R-style algorithm for optimizing join orders in parallel database systems.

that the sets of incomparable partial solutions $\mathcal{R}[i, B]$ that need to be stored and explored during the bottom-up computation can become very large. For instance, in the simple case of a leaf coefficient, it is easy to see that the number of options to consider can be as high as $O(\mathtt{q}^M)$, compared to only $O(\mathtt{q})$ in the single-measure case; furthermore, this number of possibilities can grow extremely fast (in the worst case, *exponentially*) as partial solutions are combined up the error tree.

4.4.4 GreedyRel: An Efficient, Greedy Approximation Heuristic

Given the very high running-time and space complexities of our PODP-based solution, we seek to devise an effective approximation algorithm to our maximum NSE minimization problem for extended coefficients. In this section, we propose a very efficient, greedy heuristic algorithm (termed GreedyRel) for this optimization problem. Briefly, GreedyRel tried to exploit some of the properties of dynamic-programming solutions, but allocates the synopsis space to extended coefficients greedily based on the idea of *marginal error gains*. The key idea here is to try, at each step, to allocate additional space to a *subset of extended wavelet coefficients* in the error tree that result in the *largest reduction* in the target error metric (i.e., maximum NSE^2) *per unit of space used*.

Our GreedyRel algorithm relies on three basic operations: (1) Estimating the maximum per-measure NSE^2 values at any node of the error tree; (2) Estimating the best marginal error gain for any subtree by identifying the subset of coefficients in the subtree that are expected to give the largest per-space reduction in the maximum

NSE^2; and, (3) Allocating additional synopsis space to the best overall subset of extended coefficients (in the entire error tree). We describe these three operations in detail in the remainder of this section. Consider a step of our GreedyRel algorithm, and let y_{ij} denote the currently-assigned retention probability (i.e., fractional storage) for each individual coefficient c_{ij} (i.e., at coordinate i for the j^{th} measure); also, let T_{ij} denote the error subtree (for the j^{th} measure) rooted at c_{ij}.

Estimating Maximum NSE^2 at Error-Tree Nodes. In order to determine the potential reduction in the maximum squared NSE due to extra space, GreedyRel first needs to obtain an estimate for the current maximum NSE^2 at any error-tree node. GreedyRel computes an *estimated maximum* NSE^2 $G[i,j]$ over any data value for the j^{th} measure in the T_{ij} subtree, using the recurrence:

$$G[i,j] = \begin{cases} \max \begin{cases} \frac{\mathrm{Var}(c_{ij},y_{ij})}{\mathrm{Norm}(2i,j)} + G[2i,j] \\ \\ \frac{\mathrm{Var}(c_{ij},y_{ij})}{\mathrm{Norm}(2i+1,j)} + G[2i+1,j] \end{cases} & \text{if } i < N \\ \\ 0 & \text{if } i \geq N. \end{cases}$$

The above formula is similar to the DP recurrence for computing $R[i,B]$ in Equation (4.4): The estimated maximum NSE^2 value is the maximum of two costs calculated for the node's two child subtrees, where each cost sums the estimated maximum NSE^2 of the subtree and the node's variance divided by the subtree normalization term. Note, however, that, while Equation (4.4) is exact for the maximum squared NSE in

the optimal, continuous solution for single-measure data [GG04], the above recurrence is only meant to provide an *easy-to-compute estimate* for a node's maximum NSE^2 (under a given space allotment) that GreedyRel can use in its computation.

Estimating the Best Marginal Error Gain for Subtrees. Given an error subtree T_{ij} (for the j^{th} measure), our GreedyRel algorithm computes a subset $\mathsf{potSet}[i,j]$ of coefficient values in T_{ij} which, when allotted additional space quanta, are estimated to provide the *largest per-space reduction* of the maximum squared NSE over all data values in the T_{ij} subtree. (Remember that our algorithms allocate the synopsis space budget in space quanta of $1/\mathsf{q}$, where $\mathsf{q} > 1$.) Let $G[i,j]$ be the current estimated maximum NSE^2 for T_{ij} (as described above), and let $G_{\mathsf{pot}}[i,j]$ denote the *potential* estimated maximum NSE^2 for T_{ij} assuming that a (minimal) additional space of $1/\mathsf{q}$ is allotted to all coefficient values in $\mathsf{potSet}[i,j]$. Also, let $\mathsf{potSpace}[i,j]$ denote the increase in the overall synopsis size, i.e., the cumulative increase in the space for the corresponding *extended* coefficients, when allocating the extra space to the coefficient values in $\mathsf{potSet}[i,j]$. We now describe how our GreedyRel algorithm computes $\mathsf{potSpace}[i,j]$, and how the best error-gain subsets $\mathsf{potSet}[i,j]$ are estimated through the underlying error-tree structure.

Consider a coefficient value $c_{kj} \in \mathsf{potSet}[i,j]$. Based on Equation (4.5), it is easy to see that an increase of δy_{kj} in the retention probability of c_{kj} results in an increase in the expected-space requirement $\mathrm{E}[|EC_k|]$ of the corresponding extended coefficient

EC_k (and, thus, the overall expected synopsis size) of:

$$\delta_j(\mathrm{E}[|EC_k|], \delta y_{kj}) = \delta y_{kj} \cdot (1 + H \times \prod_{p \neq j}(1 - y_{kp})). \tag{4.7}$$

The total extra space $\mathsf{potSpace}[i,j]$ for all coefficient values in $\mathsf{potSet}[i,j]$ can be obtained by adding the results of Equation (4.7) for each of these values (with $\delta y_{kj} = \frac{1}{\mathsf{q}}$); that is,

$$\mathsf{potSpace}[i,j] = \sum_{c_{kj} \in \mathsf{potSet}[i,j]} \delta_j(\mathrm{E}[|EC_k|], \frac{1}{\mathsf{q}}).$$

The *marginal error gain* for $\mathsf{potSet}[i,j]$ is then simply estimated as $\mathsf{gain}(\mathsf{potSet}[i,j]) = (G[i,j] - G_{\mathsf{pot}}[i,j])/\ \mathsf{potSpace}[i,j]$.

To estimate the $\mathsf{potSet}[i,j]$ sets, and the corresponding $G_{\mathsf{pot}}[i,j])$ (and $\mathsf{gain}()$) values at each node, GreedyRel performs a bottom-up computation over the error-tree structure. For a *leaf* coefficient c_{ij}, the only possible choice is $\mathsf{potSet}[i,j] = \{c_{ij}\}$, which can result in a reduction in the maximum NSE^2 if $c_{ij} \neq 0$ and $y_{ij} < 1$ (otherwise, the variance of the coefficient is already 0 and can be safely ignored); in this case, the new maximum NSE^2 at c_{ij} is simply $G_{\mathsf{pot}}[i,j] = \frac{\mathrm{Var}_{(c_{ij}, y_{ij} + \frac{1}{\mathsf{q}})}}{\mathrm{Norm}_{(i,j)}}$.[8] For a *non-leaf* coefficient c_{ij}, GreedyRel considers three distinct cases of forming $\mathsf{potSet}[i,j]$ and selects the one resulting in the largest marginal error gain estimate: (1) $\mathsf{potSet}[i,j] = \{c_{ij}\}$ (i.e., select only c_{ij} for additional storage); (2) $\mathsf{potSet}[i,j] = \mathsf{potSet}[k,j]$, where $k \in \{2i,\ 2i+1\}$ is such that $G[i,j] = G[k,j] + \mathrm{Var}(c_{ij}, y_{ij})/\mathrm{Norm}(k,j)$ (i.e., select the potSet from the child subtree whose estimated maximum NSE^2 determines the current

[8] As in [GG02, GG04], in our implementation, we actually cap the contribution of coefficient c_{ij} to the overall variance at c_{ij}^2. This essentially implies (see Section 4.2) that we only need to consider non-zero allotments $y_{ij} > 1/2$ to coefficient c_{ij}.

maximum NSE^2 estimate at c_{ij}); and, (3) $\mathsf{potSet}[i,j] = \mathsf{potSet}[2i,j] \cup \mathsf{potSet}[2i+1,j]$ (i.e., select the union of the potSets from both child subtrees). Among the above three choices, GreedyRel selects the one resulting in the largest value for $\mathsf{gain}(\mathsf{potSet}[i,j])$, and records the choice made for coefficient c_{ij} (1, 2, or 3) in a variable ch_{ij}.[9] In order to estimate $\mathsf{gain}(\mathsf{potSet}[i,j])$ for each choice, GreedyRel uses the following estimates for the new maximum NSE^2 $G_{\mathsf{pot}}[i,j]$ at c_{ij} (index k is defined as in case (2) above, and $l = \{2i,\, 2i+1\} - \{k\}$):

$$G_{\mathsf{pot}}[i,j] = \begin{cases} \max \left\{ \begin{array}{l} \frac{\mathrm{Var}(c_{ij}, y_{ij} + \frac{1}{\mathsf{q}})}{\mathrm{Norm}(2i,j)} + G[2i,j] \\ \frac{\mathrm{Var}(c_{ij}, y_{ij} + \frac{1}{\mathsf{q}})}{\mathrm{Norm}(2i+1,j)} + G[2i+1,j] \end{array} \right. & \mathsf{ch}_{ij} = 1 \\ \max \left\{ \begin{array}{l} \frac{\mathrm{Var}(c_{ij}, y_{ij})}{\mathrm{Norm}(k,j)} + G_{\mathsf{pot}}[k,j] \\ \frac{\mathrm{Var}(c_{ij}, y_{ij})}{\mathrm{Norm}(l,j)} + G[l,j] \end{array} \right. & \mathsf{ch}_{ij} = 2 \\ \max \left\{ \begin{array}{l} \frac{\mathrm{Var}(c_{ij}, y_{ij})}{\mathrm{Norm}(2i,j)} + G_{\mathsf{pot}}[2i,j] \\ \frac{\mathrm{Var}(c_{ij}, y_{ij})}{\mathrm{Norm}(2i+1,j)} + G_{\mathsf{pot}}[2i+1,j] \end{array} \right. & \mathsf{ch}_{ij} = 3 \end{cases}$$

As an example, consider the scenario depicted in Figure 4.7 for $M = 2$. The figure shows, for each of the children of node i, the computed G, G_{pot}, and potSpace values, along with the value of G and the current normalized variance for node i. The three cases of forming potSet for each measure at node i are enumerated, the corresponding potential reductions (`Diff`) in the estimated maximum NSE^2 value for each measure are calculated, and the choice that results in the largest per-space

[9] It is easy to see that combining the root node c_{ij} with one or both of its child potSets cannot have better marginal error gain than the best of the three options we consider.

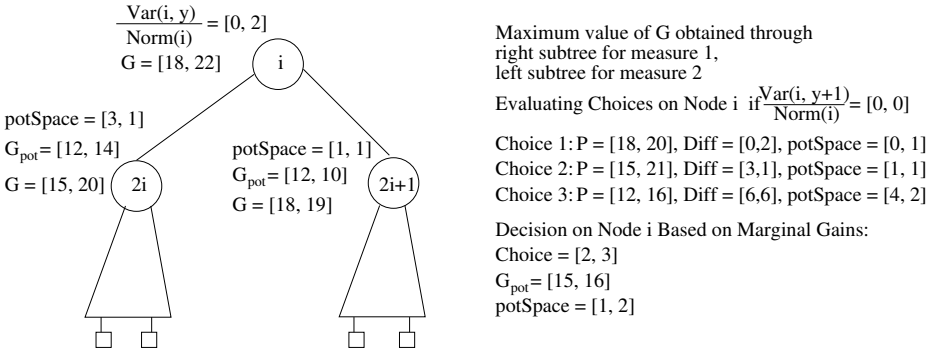


Figure 4.7: Example for GreedyRel algorithm.

reduction is selected for each measure. This figure also depicts why it is important to simultaneously increase the retention probabilities of more than one coefficient values. At any node i where the calculated G values through its children are the same, or differ only slightly, for some measure j (as is the case with measure 2 in our example), then any individual assignment of additional space to a coefficient value of that measure below node i would only result in either zero, or very small marginal gains, and would therefore not be selected, independently of how much it would reduce the maximum NSE^2 value through its subtree. This happens because the estimated value of $G[i, j]$ through the other subtree would remain the same. As the authors of [GG02] describe, in single-measure data sets the value of G through both subtrees is the same in the optimal solution, thus implying that the above situation is expected to occur very frequently.

An important point to note is that GreedyRel does not need to store the coefficient sets $\mathsf{potSet}[i, j]$ at each error-tree node. These sets can be reconstructed on the fly, by traversing the error-tree structure, examining the value of the ch_{ij} variable at each node c_{ij}, and continuing along the appropriate subtrees of the node, until we reach

nodes with $\mathsf{ch}_{ij} = 1$.

Distributing the Available Synopsis Space. After completing the above-described steps, our GreedyRel algorithm has computed the estimated current and potential maximum NSE^2 values $G[0, j]$ and $G_{\mathsf{pot}}[0, j]$ (along with the corresponding potSet and potSpace) at the root coefficient (node 0) of the error tree, for each data measure j. Since our overall objective is to minimize the maximum squared NSE among all measures over the entire domain, GreedyRel selects, at each step, the measure $j_{\max}$ with the maximum estimated NSE^2 value at the root node (i.e., $j_{\max} = \arg\max_j\{G[0, j]\}$), and proceeds to allocate additional space of $\mathsf{potSpace}[0, j_{\max}]$ to the coefficients in $\mathsf{potSet}[0, j_{\max}]$. This is done in a recursive, top-down traversal of the error tree, starting from the root node and proceeding as follows (i denotes the current node index): (1) If $\mathsf{ch}_{ij_{\max}} = 1$, set $y_{ij_{\max}} := y_{ij_{\max}} + \frac{1}{\mathsf{q}}$, (2) If $\mathsf{ch}_{ij_{\max}} = 2$, then recurse to the child subtree T_k, $k \in \{2i, 2i + 1\}$ through which the maximum NSE^2 estimate $G[i, j_{\max}]$ is computed at node i, and (3) If $\mathsf{ch}_{ij_{\max}} = 3$, then recurse to both child subtrees T_{2i} and T_{2i+1}; furthermore, after each of the above steps, compute the new values for $G[i, j_{\max}]$, $G_{\mathsf{pot}}[i, j_{\max}]$, $\mathsf{potSpace}[i, j_{\max}]$, and $\mathsf{ch}_{ij_{\max}}$ at node i.

A pseudocode description of our GreedyRel algorithm is depicted in Figure 4.8. Note that in the later steps of the algorithm, the available synopsis space may become smaller than $\mathsf{potSpace}[i, j_{\max}]$; in this case, rather than recursing on both child subtrees of a node (when $\mathsf{ch}_{ij_{\max}} = 2$), GreedyRel first recurses on the child causing the maximum estimated squared NSE, and then recurses on the other child with any remaining space

procedure GreedyRel(W_A, B, q, S)
Input: $N \times M$ array W_A of Haar wavelet coefficients; space constraint B;
quantization parameter $\mathsf{q} > 1$; vector of per-measure sanity bounds S.
Output: Array y of retention probabilities y_{ij} for all $N \times M$ coefficients.
begin
1. **for** $i := N - 1$ **downto** 0 **do** *// traverse error tree bottom-up*
2. **for** $j := 1$ **to** M **do**
3. $y_{ij} = 0$
4. Compute $G[i, j]$, $G_{\mathsf{pot}}[i, j]$, $\mathsf{potSpace}[i, j]$, and ch_{ij}
5. **endfor**
6. **endfor**
7. spaceLeft = B
8. **while** (spaceLeft > 0) **do**
9. $j_{\max} := \arg\max_j \{G[0, j]\}$
10. occupiedSpace := traverse(0, $j_{\max}$, q, y, spaceLeft)
11. spaceLeft := spaceLeft - occupiedSpace
12. **if** (occupiedSpace = 0) **then return**(y) *//not enough space*
13. **endwhile**
14. **return**(y)
end

Figure 4.8: GreedyRel Algorithm Pseudocode.

(Lines 12–16 of traverse).

Time and Space Complexity. For each of the N error-tree nodes, GreedyRel maintains the variables $G[i, j]$, $G_{\mathsf{pot}}[i, j]$, $\mathsf{potSpace}[i, j]$, and ch_{ij}. Thus, the space requirements per node are $O(M)$, resulting in a total space complexity of $O(NM)$.

In the bottom-up initialization phase (Lines 1–6), GreedyRel computes, for each error-tree node, the values of the $G[i, j]$, $G_{\mathsf{pot}}[i, j]$, $\mathsf{potSpace}[i, j]$, and ch_{ij} variables (for each measure j). Each of these $O(M)$ calculations can be done in $O(1)$ time, making the total cost of the initialization phase $O(NM)$.

Then, note that each time GreedyRel allocates space to a set of K coefficients, the allocated space is $\geq K \times 1/\mathsf{q}$ (see Equation (4.7)). To reach these K coefficients, GreedyRel traverses exactly K paths of maximum length $O(\log N)$. For each

procedure traverse(i, j, q, y, spaceLeft)
Input: Index i of error-tree node; measure j chosen for space allocation; quantization parameter q; array y of current retention probabilities; maximum synopsis space to allocate (spaceLeft).
Output: Space allocated to the T_{ij} subtree at this step.
begin
1. allocatedSpace := 0
2. **if** ($\mathsf{ch}_{ij} = 1$) **then**
3. neededSpace := $\delta_j(\mathrm{E}[|EC_i|], 1/\mathsf{q})$ // *see Equation (4.7)*
4. **if** (neededSpace $\leq$ spaceLeft) **then**
5. $y_{ij} := y_{ij} + 1/\mathsf{q}$
6. allocatedSpace := neededSpace
7. **endif**
8. **else if** ($\mathsf{ch}_{ij} = 2$) **then**
9. Find index k of child subtree through which $G[i, j]$ occurs
10. allocatedSpace := traverse(k, j, q, y, spaceLeft)
11. **else**
12. Find index k of child subtree through which $G[i, j]$ occurs
13. Let l be the index of the other subtree
14. allocatedSpace := traverse(k, j, q, y, spaceLeft)
15. **if** (spaceLeft > allocatedSpace) **then**
16. allocatedSpace += traverse(l, j, q, y, spaceLeft-allocatedSpace)
17. **endif**
18. Recompute $G[i, j]$, $G_{\mathsf{pot}}[i, j]$, potSpace$[i, j]$, and ch_{ij}
19. **return**(allocatedSpace)
end

Figure 4.9: Subroutine traverse

visited node and just for the selected measure $j_{\max}$ (chosen at the root), we need to compute the new values of G, G_{pot}, potSpace, and ch, which requires $O(1)$ time. Finding the measure $j_{\max}$ with the maximum estimated NSE^2 value at the root requires time $O(\log M)$.[10] Thus, overall, GreedyRel distributes space $\geq K \times 1/\mathsf{q}$, in time $O(K \log N + \log M)$, making the amortized time per-space-quantum $1/\mathsf{q}$ equal to $O(\log N + \log M/K) = O(\log(NM))$. Since the total number of such quanta that we need to distribute is $B\mathsf{q}$, the overall running time complexity of GreedyRel is $O(NM +$

[10] Just for the root node, we may store the $G[0, j]$ values in a heap.

$B\mathsf{q}\log(NM)$).

4.4.5 Extensions to Multi-Dimensional Wavelets

We now discuss the key ideas for extending our techniques to multi-dimensional data. An introduction to multi-dimensional wavelets and the structure of the error tree in these data sets was presented in Section 4.2.2.

Extending PODPRel**.** Our PODPRel algorithm for multi-dimensional data sets generalizes the corresponding multi-dimensional MinRelVar strategy in [GG04], in a way analogous to the one-dimensional case. In a nutshell, PODPRel needs to consider, at each internal node of the error tree, the optimal allocation of space to the $\leq 2^D - 1$ wavelet coefficients of the node and its $\leq 2^D$ child subtrees. The extension of PODPRel to multi-dimensional data sets is therefore a fairly simple adaptation of the multi-dimensional MinRelVar algorithm. However, as discussed in Section 4.4.3, PODPRel needs to maintain, for each node i and each possible space allotment B, a *collection* $\mathcal{R}[i, B]$ of incomparable solutions. This requirement, once again, makes the time/space requirements of PODPRel significantly higher than those of MinRelVar.

Extending GreedyRel**.** The first modification involved in extending our GreedyRel algorithm to multi-dimensional data sets has to do with the computation of $G[i, j]$, which now involves examining the estimated NSE^2 values over $\leq 2^D$ child subtrees and maintaining the maximum such estimate. Let $\mathcal{S}(i)$ denote the set of the $\leq 2^D - 1$ coefficients of node i, and let $i_1, \ldots, i_p$ be the indexes of i's child nodes in the error

tree. Then,

$$G[i,j] \;=\; \begin{cases} \max \begin{cases} \sum_{c_k \in \mathcal{S}(i)} \frac{\mathrm{Var}_{(c_{kj}, y_{kj})}}{\mathrm{Norm}_{(i_1, j)}} + G[i_1, j] & \\ \ldots & i < N \\ \sum_{c_k \in \mathcal{S}(i)} \frac{\mathrm{Var}_{(c_{kj}, y_{ij})}}{\mathrm{Norm}_{(i_p, j)}} + G[i_p, j] & \end{cases} \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i \geq N \end{cases}$$

The only other necessary modification involves the estimation of marginal error gains at each node. In Section 4.4.4, we consider a total of three possible choices for forming $\mathsf{potSet}[i,j]$ for each (node, measure) combination. Now, each node has up to 2^D child subtrees, resulting in a total of $2^D + 1$ possible choices of forming $\mathsf{potSet}[i,j]$. The first choice is to increase the retention probability for measure j of one of the $\leq 2^D - 1$ coefficients in node i. In this case, we simply include in $\mathsf{potSet}[i,j]$ the coefficient in node i that is expected to exhibit the largest marginal gain for measure j. For each of the remaining 2^D possible choices of forming $\mathsf{potSet}[i,j]$, the k-th choice ($1 \leq k \leq 2^D$) considers the marginal gain of increasing the retention probabilities in the child subtrees through which the k maximum $\textsc{nse}^2$ values occur, as estimated in the right-hand side of the above equation for $G[i,j]$. At each node, the computation of $G_{\mathsf{pot}}[i,j]$, $\mathsf{potSpace}[i,j]$, and ch_{ij} incurs a worst-case time cost of $O(D \times 2^D)$ due to the possible ways of forming $\mathsf{potSet}[i,j]$, and the required sorting operation of 2^D quantities. Again, let N denote the total number of cells in the multi-dimensional

data array and N_{max} denote the maximum domain size of any dimension. Then, the running time complexity of GreedyRel becomes $O(D \times 2^D \times (NM + BM\mathsf{q} \log N_{max}))$. Note, of course, that in most real-life scenarios using wavelet-based data reduction, the number of dimensions is typically a small constant (e.g., 4–6).

Improving the Complexity of GreedyRel. In the wavelet decomposition process of a multi-dimensional data set, the number of non-zero coefficients produced may be significantly larger than the number N_z of non-zero data values. In [GG04] the authors proposed an adaptive coefficient thresholding procedure that retains at most N_z wavelet coefficients *without* introducing any reconstruction bias. Using this procedure, the authors in [GG04] demonstrated how the MinRelVar algorithm can be modified so that its running time and space complexities have a dependency on N_z, and not on N (i.e., the total number of cells in the multi-dimensional data array). It would thus be desirable if the GreedyRel algorithm could be modified in a similar way, in order to decrease its running time and space requirements.

Let N_z denote the number of error tree nodes that contain non-zero coefficient values, possibly after the afore-mentioned thresholding process. We will first illustrate that for any node in the error tree containing zero coefficient values, and which has at most one node in its subtree that contains non-zero coefficient values, no computation is needed. Equivalently, our algorithm will need to compute G, G_{pot} values in only: (i) nodes containing non-zero coefficient values; or (b) nodes that contain zero coefficient values, but which are the least common ancestor of at least two non-zero tree nodes beneath them in the error tree.

Let k be a node that is the only node in its subtree with non-zero coefficient values. Obviously we do not need to consider the G, G_{pot} values in the descendant nodes of k, since they will be zero. An important observation is that for any ancestor of k that contains just a single non-zero error tree beneath it (which is certainly the subtree of node k), no computation is necessary, since the G, G_{pot} values of k can always be used instead. The only additional computation is needed in any node n with zero-coefficients that has at least two non-zero error tree nodes beneath it in the error tree (in different subtrees). In this case, the G, G_{pot} values of node n needs to be calculated, using as input the G, G_{pot} values of its non-zero descendant tree nodes. It is easy to demonstrate that at most $N_z - 1$ such nodes may exist. Thus, the GreedyRel algorithm will need to calculate the G, G_{pot} values in at most $O(2N_z-1)$=$O(N_z)$ nodes, thus yielding running time and space complexities of $O(D\times 2^D\times\ (N_zM+\ BM\mathsf{q}\log N_{max}))$ and $O(N_zM)$, respectively. We here need to note that in order to implement our algorithm as described here, we need to sort the N_z coefficients based on their postorder numbering in the error tree. This requires an additional $O(N_z \log N_z)$ time for the sorting process. However, this running time is often significantly smaller than the benefits of having running time and space dependencies based on N_z, rather than on N.

Table 4.6 contains a synopsis of the running time and space complexities of our GreedyRel and the MinRelVar algorithm of [GG04].

Algorithm	Space	Running Time
GreedyRel	$O(N_z M)$	$O(D2^D \times (N_z M + BM\mathsf{q} \log N_{max}))$
MinRelVar	$O(N_z MB2^D \mathsf{q})$	$O(N_z BM2^D \mathsf{q}(\mathsf{q} \log(\mathsf{q}B) + D2^D))$

Table 4.6: GreedyRel and MinRelVar complexities.

4.4.6 Comparing GreedyRel and GreedyL2

When comparing the GreedyL2 and the GreedyRel algorithms, we can observe that, while the two algorithms share some common characteristics, there are distinct differences in the way that the two algorithms operate.

Both algorithms operate on all the measures of the data set simultaneously and utilize the notion of extended wavelets in order to achieve better storage utilization and, thus, increased accuracy. Moreover, both algorithms allocate at each step space to a *group* of coefficient values. The GreedyL2 algorithm, at each step, stores a group of coefficient values corresponding to the same coefficient coordinates, since this grouped space allocation policy often results in better per space benefits than storing individual coefficient values one-by-one. On the other hand, the GreedyRel algorithm increases, at each step, the retain probabilities of a group of coefficient values corresponding to the same measure. The intuition behind this allocation policy is that a group assignment may result in a larger per space decrease of the maximum NSE value than increasing the retain probabilities of individual coefficient values, especially in cases when two, or more, subtrees have similar maximum NSE values. The storage dependencies among coefficient values are taken into account when calculating the per space benefits of each space assignment on each node. We thus expect that the improvements in the accuracy of the obtained approximation will be larger in the case of minimizing

the weighted sum squared error, since the intra-coefficient storage dependencies are taken into account more directly. While non-zero coefficient values always have some benefit in reducing the weighted sum squared error of the approximation, this is often not true when trying to minimize the maximum relative error of any data value, since these non-zero coefficient values need to lie on root-to-leaf paths with high NSE values in order for the algorithm to increase their retention probabilities. Finally, while the GreedyRel algorithm naturally extends to multi-dimensional data sets, the extensions of GreedyL2 is even more straightforward, since it operates directly on the input candidate combined coefficients, without needing to take into account the error tree structure.

4.5 Experimental Study

In this section, we present an extensive experimental study of our proposed algorithms for constructing wavelet synopses over data sets with multiple measures. Besides validating the effectiveness of our extended wavelet coefficient approach compared to existing *Individual* and *Combined* schemes, one of the main objectives of our study was to evaluate the accuracy and scalability of our proposed GreedyL2 and GreedyRel algorithms over several real-life and synthetic multi-measure data sets. The main findings of our study can be summarized as follows.

- **Highly Scalable Solutions.** Our GreedyL2 and GreedyRel algorithms provide fast and highly-scalable solutions for constructing conventional and probabilistic wavelet synopses over large multi-measure data sets. Unlike earlier schemes (and

the DynProgL2 and PODPRel algorithms), the GreedyRel and GreedyL2 algorithms exhibit a linear and near-linear, correspondingly, dependency on the domain size. Moreover, for probabilistic wavelet synopses, the running time and space requirements of earlier techniques yield our GreedyRel algorithm as the only viable solution, even for the single-measure case, for large real-life data sets.

- **Near Optimal Results.** The GreedyL2 and GreedyRel algorithms consistently provide near-optimal solutions when compared to DynProgL2 and PODPRel (respectively), demonstrating that they constitute efficient techniques for constructing accurate conventional and probabilistic synopses over large multi-measure data sets.

- **Improved Accuracy for Individual Reconstructed Answers through the use of Extended Wavelet Coefficients.** Compared to earlier approaches that operate on each measure individually, our GreedyL2 and GreedyRel algorithms significantly reduce the weighted sum squared and maximum relative error of the approximation. These improvements are often by a factor of $2-3$, but in many cases we also observe up to 7 times smaller errors than the closest competitive technique. The improvements in the obtained accuracy are, of course, due to the flexible storage format of the extended wavelet coefficients and the improved storage utilization that they achieve.

All experiments reported in this section were performed on a personal computer using an Athlon XP 1800+ processor with 512 MB of RAM memory.

4.5.1 Techniques and Parameter Settings

Our experimental study is split into two parts, based on the error metric that our algorithms try to minimize. We here need to emphasize that, besides the presented techniques, we also performed a comparative analysis of the GreedyL2 and GreedyRel algorithms in multi-measure data sets. However, the results were, as expected, qualitatively similar to the ones presented in [GG04], with the GreedyL2 algorithm producing wavelet synopses with smaller weighted sum squared errors, while the GreedyRel algorithm consistently produced significantly smaller maximum relative errors. This is not surprising, as the two algorithms are designed to minimize different error metrics, and the existence of multiple measures in data sets cannot result in a qualitatively different behavior than in single-measure data sets. We thus omit this comparison from our discussion.

• **Weighted Sum Squared Error.** We initially compared the performance of the GreedyL2 and the DynProgL2 algorithms for constructing conventional wavelet synopses over multi-measure data sets to the following four algorithms: (1) Random Sampling (RS), using the Reservoir algorithm described in [Vit85], since the data sets that we used did not contain duplicate tuples; (2) *Ind*, where the individual space allocated to each measure is proportional to its weight, and the *Individual* algorithm is then run for each measure; (3) *IndSorted*, where the individual coefficients from all measures are sorted according to their weighted benefit, and the ones with the highest benefits are retained, without imposing any limit to the size allocated to each measure; and (4) *Combined*, where the *combined* coefficients are sorted according to their overall

weighted benefit, and the ones with the highest benefits are retained.

Histograms were not included in the performance evaluation, due to their inability to extend to data sets with multiple measures. Creating a separate histogram for each measure might be an alternative, but the work in [CGRS00, VW99] leads us to expect that they would perform worse than the Ind algorithm. As input to our algorithms we used the output of the decomposition step of the *Combined* algorithm, which we found to produce better results than the corresponding output of the *Individual* algorithm.

• **Maximum Relative Error.** In the second part of our experimental study we compare our GreedyRel and PODP algorithms for constructing probabilistic data synopses over multi-measure data sets, along with a technique, which we will term as IndDP that partitions the available space equally over the measures and then operates on each measure individually by utilizing the dynamic programming MinRelVar algorithm proposed by Garofalakis and Gibbons in [GG02]. To provide a more fair comparison to the IndDP algorithm, the majority of our experiments includes data sets where all the measured quantities exhibit similar characteristics, thus yielding the uniform partitioning of the synopsis space over all the measures as the appropriate space allocation technique. The only parameter in our algorithms is the quantization parameter q, which is assigned a value of 10 for the GreedyRel and IndDP algorithms, and a smaller value of 4 for the PODP algorithm to reduce its running time (the accuracy of the produced synopses was similar in PODP with larger values of q). Moreover, the sanity bound of each measure is set to the 5%-quantile value of the

measure's data values.

4.5.2 Data Sets

We experiment with several one-dimensional and multi-dimensional synthetic multi-measure data sets and present in this section a representative set of results. The input parameters of our Zipfian data generator along with their default values are described in Table 4.7. The generator begins by populating $n\_regions$ rectangular regions of a D-dimensional array, whose size is determined by the number of the data set's dimensions and the cardinalities $Card_i$ of each dimension. The number of cells within each region is bound by the values V_{min} and V_{max}. The total sum Sum_i of values for each measure is partitioned across the $n\_regions$ rectangular regions through the use of a Zipf function with parameter Z. Then, within each region each measure's values are distributed by using one of the five distributions described in Table 4.8, with the parameter's values ranging from z_{min_i} to z_{max_i}. Notice the use of the *Altered-X*[11] distribution, which helps create pairs of measures with similar, but not identical, data distributions. The data generator then also populates a number of cells in the remaining D-dimensional space, outside the dense regions. The fraction of such cells over the total number of populated cells is defined by the $spCount$ parameter, and the total sum of the values of these cells by the $spSum_i$ parameter. Especially for the case of one-dimensional data sets, the produced zipfian distributions span the entire domain (i.e., $n\_regions = 1$, $V_{min} = V_{max} = Card_1$ and $spCount = spSum_i = 0$).

In our experimental study we also use real-life data sets. The `Weather` data

[11] X can be either one of the NoPerm, Normal, Middle or PipeOrgan distributions.

Parameter	Description	Default Value
D	Number of dimensions	2
M	Number of measures	6
$Card_i$	Cardinality of dimension i	1024
n_regions	Number of dense regions	10
V_{min}, V_{max}	Minimum and maximum volume of regions	$4900, 4900$
Z	Skew across regions	0.5
z_{min_i}, z_{max_i}	Minimum and maximum skew within region i	1, 1
Sum_i	Sum of values for measure i	1,000,000
spCount	Fraction of populated cells in sparse areas	0.05
$spSum_i$	Sum of values of populated cells in sparse area i	0.05

Table 4.7: Data Generator Input Parameters and Default Values

Distribution	Description
NoPerm	Cells with smaller L1-distance from lower-left corner have larger values
Normal	Cells with smaller L1-distance from center have larger values
PipeOrgan	Cells with smaller L1-distance from center have smaller values
Middle	Consider a hyper-rectangle centered at the region's center, and having for each dimension, half the length of the corresponding region length. Cells with smaller L1-distance from this hyper-rectangle have larger values
Altered-X	This measure follows the same distribution as X distribution, but its values are randomly altered by up to 50%

Table 4.8: Data Generator Value Distributions

set contains meteorological measurements obtained by a station at the university of Washington (data at `http://www-k12.atmos.washington.edu/k12/grayskies`). For this data set we extract the following 6 measured quantities: wind speed, wind peak, solar irradiance, relative humidity, air temperature and dewpoint temperature. The `Phone` data set includes the total number of long distance calls per minute originating from 6 states (CA, GA, NJ, NY, TX, WA).

Approximation Error Metrics. The reported approximation error metric in each case depends on our optimization problem. For conventional wavelet synopses we

mainly report the weighted absolute error for all queries of our workload, calculated as:

$$(\sum_{j=1}^{M} W_j)^{-1} \times \sum_{i=1}^{M} (W_i \times |actualresult_i - estimatedresult_i|),$$

where the variables $actualresult_i$ and $estimatedresult_i$ denote the exact and the estimated values of the query result for measure i, correspondingly. The weighted sum squared and relative errors are defined similarly, with the weighted sum squared error also being reported in most cases. In the case of probabilistic wavelet synopses, we focus on the maximum relative error of the approximation, which can provide guaranteed error-bounds for the reconstruction of any individual data value, and is the error metric that our probabilistic wavelet synopses algorithms try to minimize.

4.5.3 Weighted Sum Squared Error Algorithms

Synthetic Data Sets

We first investigate how close the weighted benefit achieved by the GreedyL2 algorithm is to the one achieved by the DynProgL2 algorithm. We created a synthetic 2-dimensional data set with 4 measures, following the Normal, Altered-Normal, Middle and PipeOrgan distributions, and set the remaining parameters to the default values of Table 4.7. We modified the storage constraint from 1200 to 6000 bytes and present the results in Table 4.9. The deviation factor presented in this table is defined as: $1 - \frac{Benefit(GreedyL2)}{Benefit(\text{DynProgL2})}$. The GreedyL2 algorithm produced solutions with benefit very close to the optimal one, as expected by its tight approximation bound. Due to the large amount of memory required by the DynProgL2 algorithm, we were unable to

	Storage Constraint (Bytes)				
	1200	2400	3600	4800	6000
Deviation Factor	3×10^{-5}	5×10^{-4}	10^{-6}	10^{-4}	10^{-6}

Table 4.9: Deviation Factor of GreedyL2 Benefit when Compared to the DynProgL2 Benefit

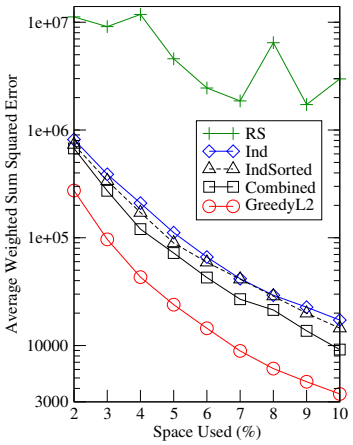


Figure 4.10: Average Weighted Squared Error

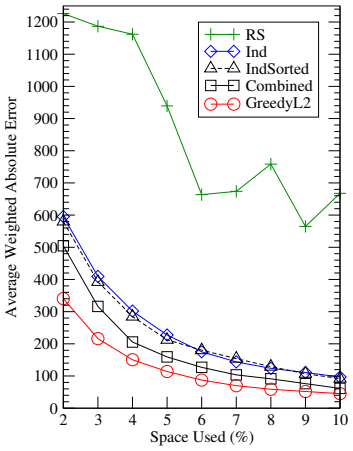


Figure 4.11: Average Weighted Absolute Error

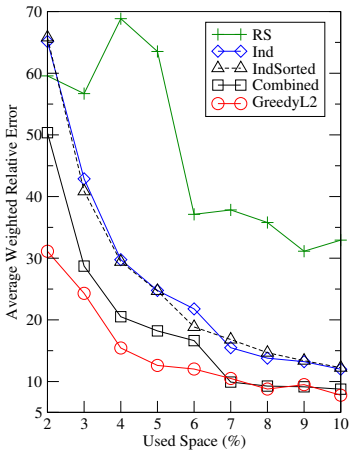


Figure 4.12: Average Weighted Relative Error

execute it for the remaining experiments and is, thus, omitted from the presented results.

We now evaluate the impact that several parameters have on the performance of our GreedyL2 algorithm. In each experiment, unless specified otherwise, the data generator parameters were set to their default values. For the default number of measures (6), the data distributions were: Normal, Altered-Normal, PipeOrgan, Altered-PipeOrgan, Middle and Altered-Middle. The query workload always consisted of 100 range queries, with the width of the range on each dimension being equal to 10. The queries targeted the dense areas with greater probability, since most of the data is stored there. The default storage bound was set to 5% of the data set's size.

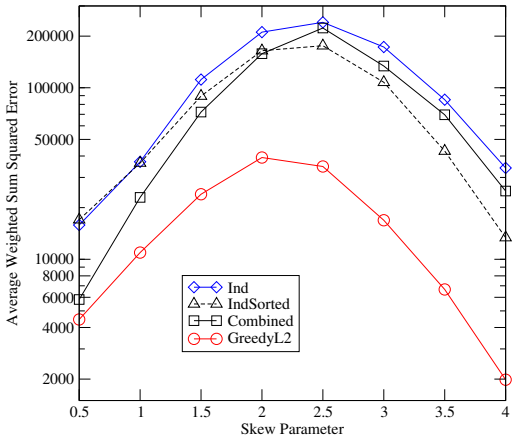


Figure 4.13: Sensitivity to Skew: Sum Squared Error

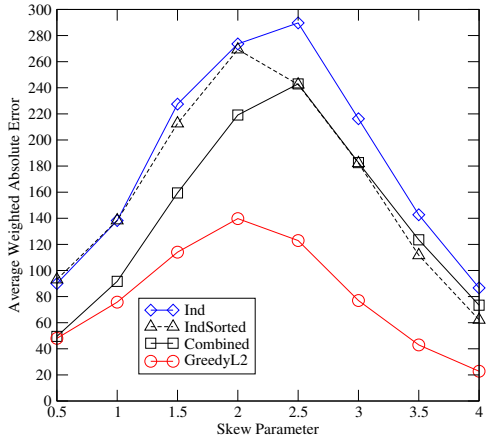


Figure 4.14: Sensitivity to Skew: Absolute Error

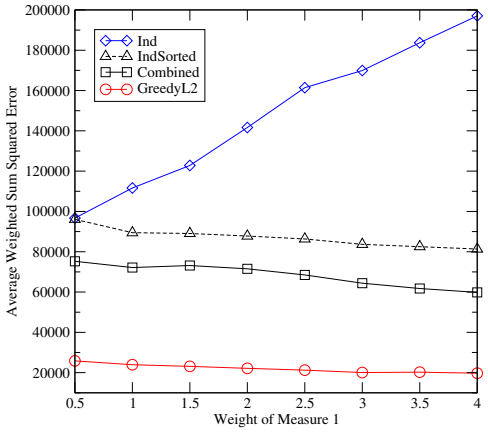


Figure 4.15: Sensitivity to Weights: Sum Squared Error

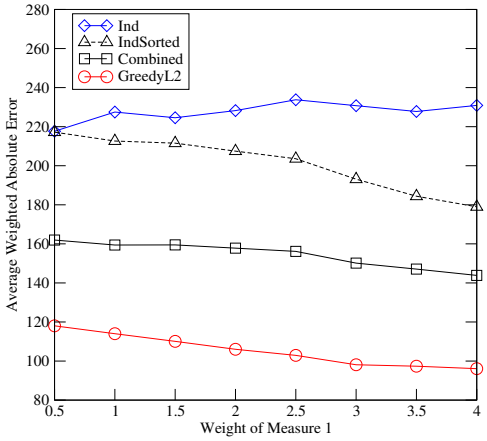


Figure 4.16: Sensitivity to Weights: Absolute Error

Storage Space. In Figures 4.10, 4.11, 4.12 we present the average weighted sum squared, absolute and relative errors, respectively, for all the algorithms as the storage space is varied from 2 to 10% of the data set's size. The skew of the data distributions within each region was set to 1.5. Note that the y-axis of Figure 4.10 is logarithmic, due to the large errors exhibited by Random Sampling. The GreedyL2 algorithm produced results with considerably smaller errors than the ones of the other algorithm.

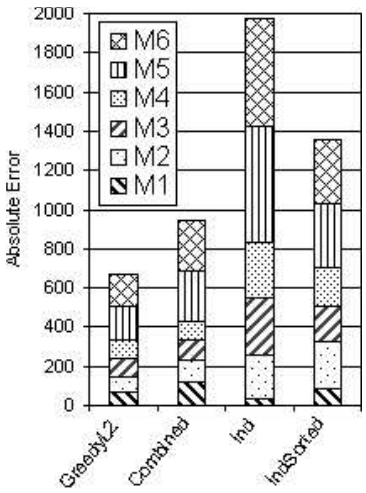


Figure 4.17: Errors for Different Measures

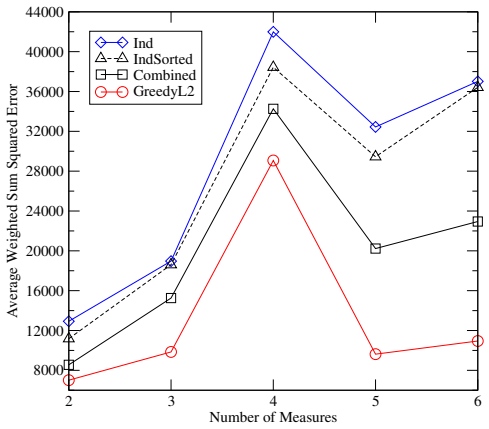


Figure 4.18: Sensitivity to Number of Measures

In particular, the average weighted sum squared error of GreedyL2 was in most cases about 3 times smaller than the error of the closest competitor, and as low as 3.5 times smaller (6105.25 vs 21399.2 for 8% space). For the average weighted absolute error case, the error of GreedyL2 was typically about 1.5 times smaller than the one produced by the closest competitor (51.85 vs 91.19 for 9% space, a ratio of 1.76). Finally, for the average weighted relative error, GreedyL2 produced results that were up to 1.62 times smaller (31.13% vs 50.36% for 2% space) than the ones of the closest competitor. From the remaining methods, the *Combined* algorithm produced the best results.

For the remaining experiments we present only the results for the average weighted sum squared and absolute errors. We will also omit from the graphs the results for the Random Sampling algorithm since its errors were consistently much larger than the ones of the other algorithms.

Skew within Regions. We modified the zipfian parameter controlling the skew of

the measure's data distributions within each region from 0.5 to 4. Figures 4.13 and 4.14 present the obtained results for the weighted average sum squared and absolute errors. As the skew increases, for each distribution the coefficients with large values are limited to an increasingly smaller area. This results on one hand in the reduction of the sum squared error of the results, as the number of coefficient values that greatly influence it becomes smaller. On the other hand, the probability that coefficient values from multiple measures be important simultaneously is decreased. These two factors justify the relative improvement of the performance of IndSorted over the *Combined* algorithm for larger skew values. While the *Combined* algorithm performs closely to the GreedyL2 algorithm for small skews, the difference becomes very large as the skew increases. For large skews, GreedyL2 exhibits about a 3-fold improvement over the closest competitive algorithm for the average weighter absolute error (22.79 vs 62.21 for skew parameter = 4) and up to a 7-fold improvement for the average weighted sum squared error, for the same skew parameter.

Variance in Weights. We varied the weight of the first measure from 0.5 to 4 to identify the impact on the accuracy of the produced result. Figures 4.15 and 4.16 present the results. GreedyL2 exhibits weighted absolute errors that are consistently about 1.5 times smaller than those of the closest competitor (97.36 vs 147.15 for weight = 3.5). For the average weighted sum squared errors, the GreedyL2 algorithm consistently provides a 3-fold improvement, and as high as 3.23 (71530.3 vs 22148.4 for weight = 2). It is interesting to note that the GreedyL2 and the IndSorted methods exhibit the greatest improvement in accuracy when the weights are varied significantly,

both reducing their errors by about 19%.

It is interesting to see for this experiment how well each measure is approximated by the different algorithms. Figure 4.17 presents the average absolute error for each measure, for the case when the first measure is assigned a weight value of 4. To calculate the average weighted error for all measures, the error of Measure 1 ($M1$) needs to be multiplied by a factor of 4, and the resulting quantity to be divided by the value 9, which is the sum of the measures' weights. As Figure 4.17 shows, the Ind algorithm exhibits the smallest error for the measure with the largest weight, about half of the error that GreedyL2 achieves, while the *Combined* algorithm performs the worst for this measure. However, GreedyL2 achieves the lowest errors for the remaining five measures thus displaying that even though it can adjust its choices in cases of measures with large weights, it does so without without severely impacting the accuracy of the remaining measures. Another interesting observation is that the second measure, which follows a distribution similar to the heavily weighted Measure 1, benefited significantly in the GreedyL2 algorithm, a behavior that was not observed in the other algorithms.

Number of Measures. In Figures 4.18 and 4.19 we present the average weighted sum squared and absolute errors as the number of measures is varied from 2 to 6. The initial two measures are the ones with distributions Normal and Middle, and the measures that are later added are: PipeOrgan, Altered-Normal, Altered-PipeOrgan, Altered-Middle. As the number of measures increases, the improvement on accuracy of GreedyL2 over the competitive methods increases.

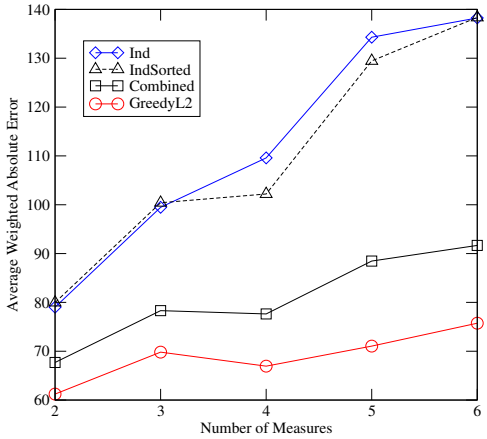


Figure 4.19: Sensitivity to Number of Measures

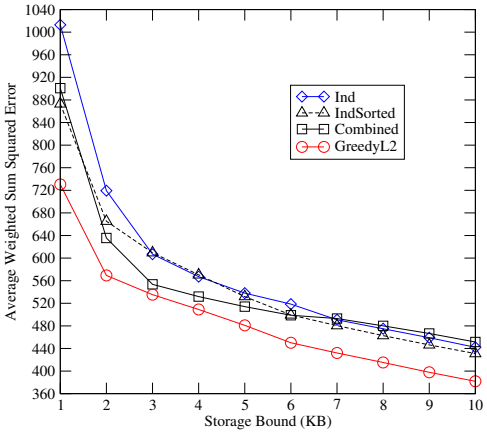


Figure 4.20: Sum Squared Errors for Weather Data Set

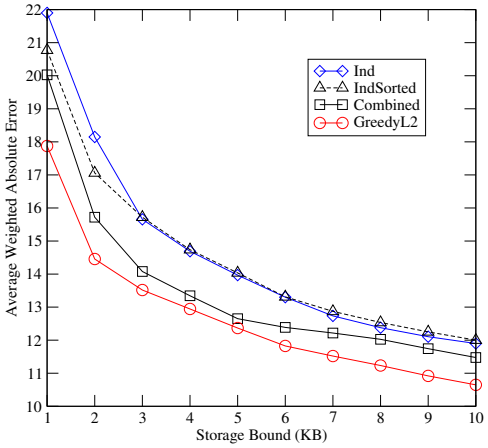


Figure 4.21: Weighted Absolute Errors for Weather Data Set

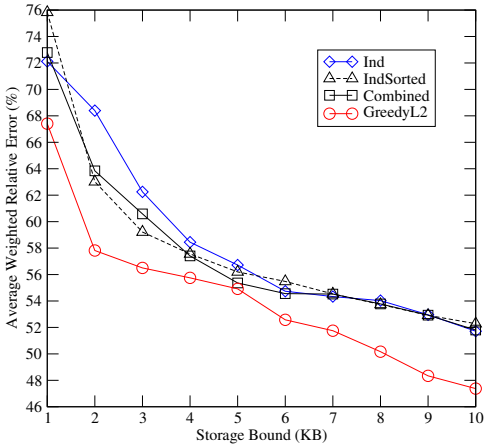


Figure 4.22: Weighted Relative Errors for Weather Data Set

Real Data Set

For our real data set we used the weather data set containing weather measurements from the state of Washington (see Section 4.5.2). To simulate enhanced interest to specific measures, we assigned a weight value of 3 to the first measure, a weight value of 2 to the next two measures, and a weight value of 1 to the remaining measures. We constructed a two-dimensional data set, with the day and time of each measurement as the two dimensions, with a total of 521817 tuples. We performed 1000 range queries,

where each range for the two dimensions was a random number with maximum value 30 and 180, respectively. Thus, the maximum selectivity of a query was about 1%. From the results of each query, we calculated average values for the stored measures over the queried day and time periods. The average values for each query were calculated by using the number of cells that each query accessed. All measures were normalized, with a process described in [DR03], where additional experiments involving different query selectivities and non-normalized measures are also presented. In Figures 4.20, 4.21, 4.22 we present the results for the average weighted sum squared, absolute and relative errors, as the storage bound was varied from 1KB to 10KB. The errors for all wavelet methods decrease with the increase of the space bound. As it can be seen from the three figures, the GreedyL2 algorithm consistently produced the smallest errors for all error metrics. The average weighted sum squared and absolute errors of Random Sampling were 2 and 1 orders of magnitude larger, respectively, than the corresponding errors of GreedyL2, while the average weighted relative errors of Random Sampling were about 3-4 times larger than the errors of GreedyL2. Even though the optimization problem of Section 4.3 is directly linked only to the average weighted sum squared metric, the improved storage utilization of GreedyL2 resulted in improvements in the accuracy of the average weighted absolute and relative error metrics as well.

4.5.4 Maximum Relative Error Algorithms

We now compare the performance and accuracy of our GreedyRel algorithm in comparison to the PODP and IndDP algorithms, described in Section 4.5.1. Our study included several one-dimensional synthetic and real data sets. For the synthetic data sets, our zipfian data generator produced zipfian distributions of various skews (from a low skew of 0.5 to a high skew of 1.5), with the sum of values for each measure set at 200,000. In all types of used zipfian distributions, the centers of the M distributions are shifted and placed in random points of the domain. We also consider several different combinations of used zipfian distributions. In the "AllNoPerm" combination, all M of the zipfian distributions have the "NoPerm" shape. Similarly, in the "AllNormal" combination, all M of the zipfian distributions have the "Normal" shape. Finally, in the "Mixed" combination, 1/3 of the M distributions have the "NoPerm" shape, 1/3 have the "Normal" shape, and the remaining had the "PipeOrgan" shape. The results presented in this section are indicative of the multiple possible combinations of our parameters.

Comparing PODP and GreedyRel. We now evaluate the accuracy and running time of the GreedyRel algorithm in comparison to the PODP algorithm. In Figures 4.23, 4.24 and 4.25 we plot the running time and the maximum and average relative errors, correspondingly, for the two algorithms and for the Weather data set when we vary the synopsis space from 10 to 50 units of space (recall that the unit of space is the size of each data value, i.e., sizeof(float)). In this experiment we only use from the Weather data the three most difficult to approximate measures. The domain size

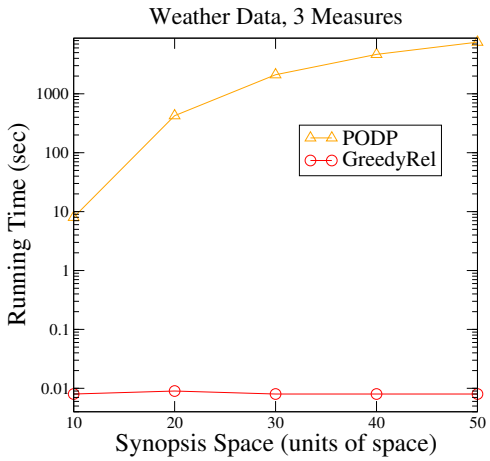


Figure 4.23: Running Time vs Space

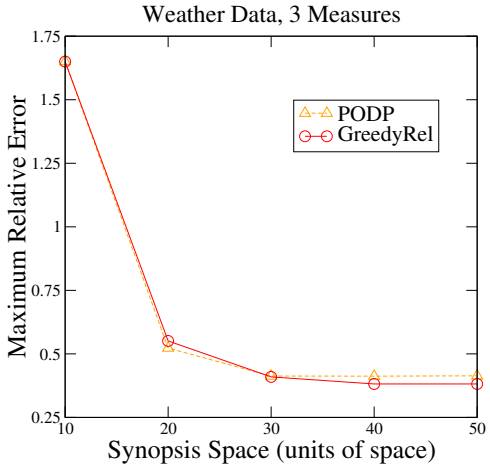


Figure 4.24: Maximum Relative Error

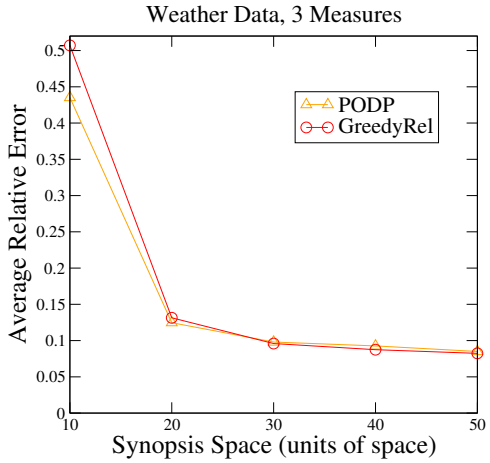


Figure 4.25: Average Relative Error

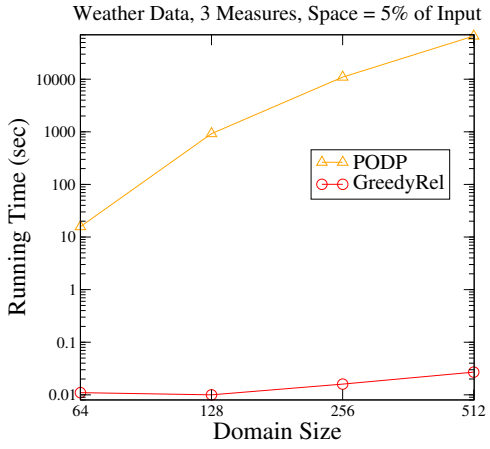


Figure 4.26: Running Time vs Domain

of the data set is set to 128. Note that in all our plots depicting the running time of algorithms, the Y axis is logarithmic. Clearly, the running time of the PODP algorithm does not scale well with the size of the data synopsis, even for such a small data set. For example, for a synopsis size of 50 space units, the PODP algorithm requires more than 2 hours to complete, while the GreedyRel algorithm required just a few milliseconds. However, as Figures 4.24 and 4.25 demonstrate, the GreedyRel algorithm provides near-optimal solutions in all cases.

In Figure 4.26 we present the corresponding running times for both algorithms,

as the domain size is increased from 64 to 512. From the weather data set we again extract just three measures, and set the synopsis space to always be 5% of the size of the input. Again, the running time performance of PODP is disappointing. For a domain size of 512, its running time exceeds 14 hours. Finally, as Figure 4.27 demonstrates, the running time of PODP increases exponentially with the number of the data set measures. Note that for data sets with 4 or more measures, the PODP does not terminate within one day. It is easy to see that the PODP algorithm cannot be used but for toy-like data sets. On the other hand, the GreedyRel algorithm provides near-optimal solutions in all tested cases, while exhibiting small running times.

Running Time Comparison of GreedyRel and IndDP. We now compare GreedyRel and IndDP in terms of their running time. In Figure 4.28 we plot the running times of the IndDP and GreedyRel algorithms for the Weather data set (all 6 measures were included) as the domain size is increased from 128 to 524288. The synopsis size is always set to 5% of the input data. The IndDP algorithm is considerably slower than the GreedyRel algorithm (3 orders of magnitude slower for domain size 131,072), with their difference increasing rapidly with the increase of the domain size. Note that while the GreedyRel algorithm scales linearly with the increase in the domain size (doubling the domain size doubles the running time), the IndDP algorithm grows much faster every time the domain size is doubled. This is of course consistent with the running time complexity of the IndDP algorithm (see Section 4.4.1), since when the domain size is doubled, the synopsis space is doubled as well. Moreover, the large memory requirements ($O(NB\mathsf{q})$) of the IndDP algorithm prevented it from terminating for domain sizes larger than 131,072 (the main memory of our machine

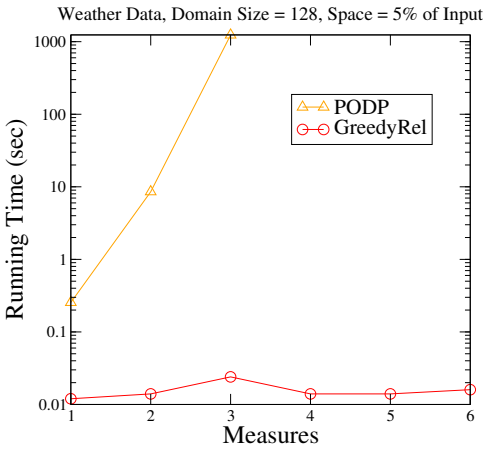


Figure 4.27: Running Time vs Measures

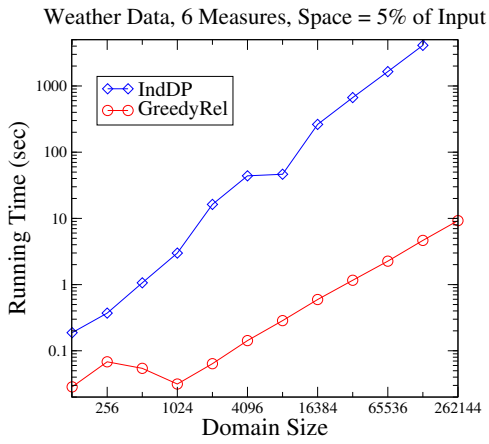


Figure 4.28: Running Time vs Domain Size

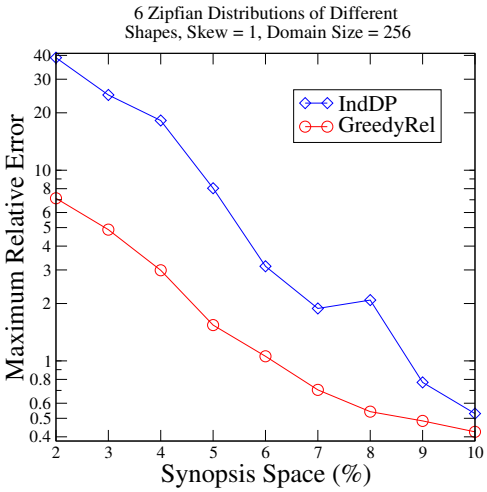


Figure 4.29: Skew 1, "Mixed"

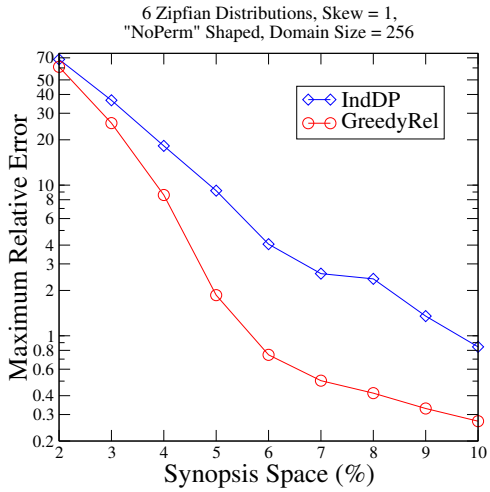


Figure 4.30: Skew 1, "AllNoPerm"

was 512MB). Thus, the linear scalability of the GreedyRel algorithm to the domain size, in terms of both its running time and its memory requirements, constitutes it as the only viable technique (except for small data sets) for providing tight error guarantees, not only on multi-measure data sets, but also on single-measure data sets, since both the GreedyRel and IndDP algorithms scale in a similar way for such data sets. Moreover, as we will demonstrate in this section, the GreedyRel algorithm, which utilizes the extended wavelet coefficients to store the selected coefficient values,

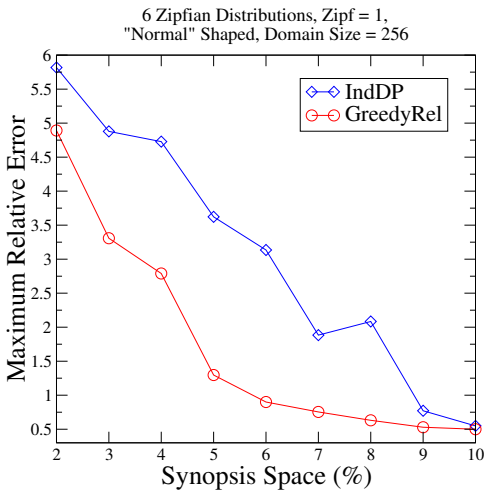


Figure 4.31: Skew 1, "AllNormal"

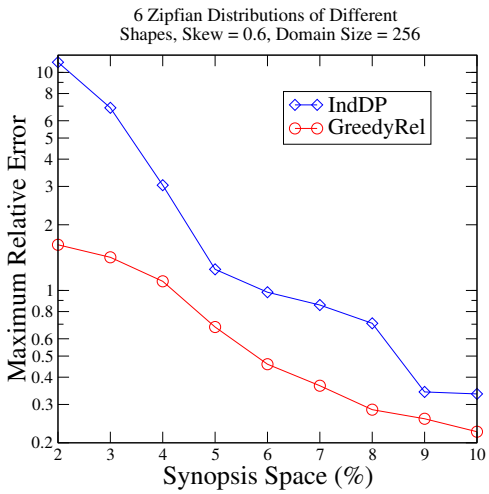


Figure 4.32: Skew 0.6, "Mixed"

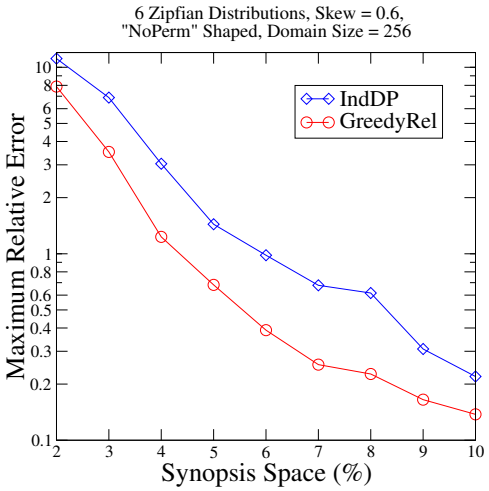


Figure 4.33: Skew 0.6, "AllNoPerm"

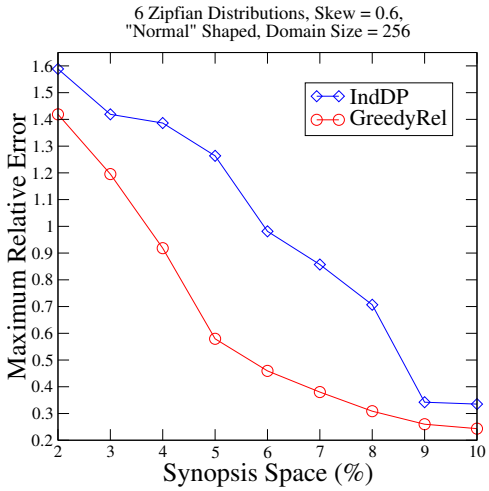


Figure 4.34: Skew 0.6, "AllNormal"

also outperforms the IndDP algorithm in terms of the obtained accuracy of the data synopsis. The improved accuracy is attributed to the improved storage utilization achieved by the use of extended wavelet coefficients, and the ability of our GreedyRel algorithm to exploit the underlying storage dependencies.

Accuracy Comparison of GreedyRel and IndDP in Synthetic Data Sets. For our synthetic data sets, we use a domain size of 256, and present the obtained accuracy in terms of the maximum error of the approximation for the GreedyRel and IndDP

algorithms and six representative combinations of synthetic data sets. These six combinations arise from considering zipfian distributions with skew 0.6 and 1, along with all the other possible combinations of the used zipfian distributions ("AllNoPerm", "AllNormal" and "Mixed"). The synthetic data sets in this section contain 6 measures/distributions.

We first consider the six possible combinations arising from distributions having skew equal to 1. In Figures 4.29, 4.30 and 4.31 we plot the maximum relative errors for the GreedyRel and IndDP algorithms, as the synopsis space is varied from 2% to 10% of the input data size, and for the "Mixed","AllNoPerm" and "AllNormal" (in the specific order) selection of zipfian distribution shapes. Note that the Y axis for the "AllNoPerm" and "Mixed" cases is logarithmic, due to the large maximum errors observed in this case, mainly by the IndDP algorithm. Intuitively, this occurs because the shifting of some distribution centers in this case results in the largest values of the data set being adjacent to the smallest values, thus requiring several coefficient values to capture this large difference of the values. As we can see, the GreedyRel algorithm produces more accurate results than the IndDP algorithm, with the differences being more significant in the "AllNoPerm" and "Mixed" cases (recall that the Y axis is logarithmic in these 2 cases). Even though none of the techniques produces tight error bounds for such a large data skew value and for small data synopses, the improvements achieved by the GreedyRel algorithm are very significant in each combination of used zipfian distributions. For each combination, GreedyRel produces, correspondingly, up to 6.1, 5.7 and 3.5 times smaller maximum relative errors than IndDP.

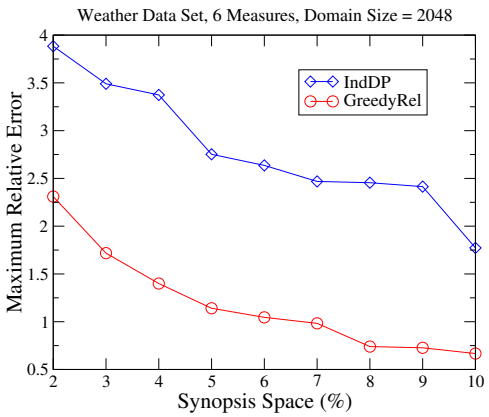


Figure 4.35: Weather Data

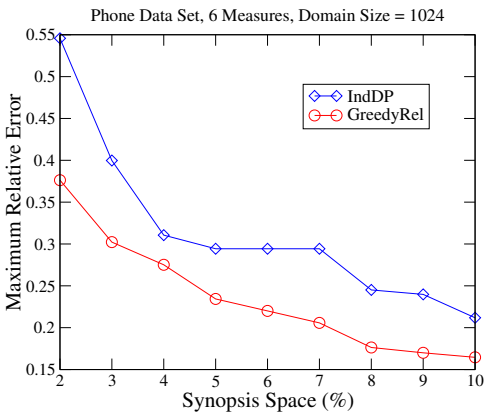


Figure 4.36: Phone Data

Similar results are also observed for the six combinations of synthetic data sets, arising from setting the skew of the distributions to 0.6. In Figures 4.32, 4.33 and 4.34 we show the corresponding results for the "Mixed", "AllNoPerm" and "AllNormal" combinations of used data distributions (note the logarithmic Y axis in the "AllNoPerm" and "Mixed" cases). The maximum relative errors in this case are significantly smaller for all methods. However, the GreedyRel algorithm is still able to provide substantial more tight error bounds, up to 6.9, 2.7 and 2.3 times smaller than IndDP.

Accuracy Comparison of GreedyRel and IndDP in Real Data Sets. In Figures 4.35 and 4.36 we plot the maximum relative errors, respectively, for the Weather and Phone data sets, as we vary the size of the synopsis, and for domain sizes of 2048 and 1024, respectively. As we can see, the benefits of the GreedyRel algorithm continue to be significant in all cases. In the Weather data set, the GreedyRel algorithm provided up to 3.5 times tighter error bounds than the IndDP algorithm (and commonly at least a 2-fold improvement), while in the Phone data the corresponding error bounds were up to 1.75 times tighter.

Chapter 5

Hierarchical in-Network Aggregate Continuous Queries

5.1 Introduction

Processing in sensor networks is often driven by designated monitoring nodes, which usually possess increased processing, storage and energy resources, when compared to the other nodes in the network. These monitoring nodes often evaluate the current state of the network by issuing *continuous queries* [CDTW00, TGNO92] over the data collected by the sensors.

Because of the multi-hop communication between nodes in sensor networks, the broadcast nature of the transmitted messages and the high density of nodes in a typical installation, collecting individual node measurements at the monitoring node is immensely expensive. Aggregation is an effective mean to reduce the data measurements into a much smaller set of comprehensive statistics, like sum, average etc. In order to obtain the full benefits of data aggregation, recent proposals perform the process inside the network [CT00, IEGH02, MFHH02, SWR98]. First, a routing path, which is commonly referred to as the *aggregation tree*, to the monitoring node is established. Then, through the use of a carefully designed transmission schedule,

nodes are programmed to combine measurements that they receive from their descendants in the topology and propagate a single value to their ancestors. With proper synchronization [MFHH02], the number of messages required to update the aggregate at the monitoring node is equal to the number of edges in the aggregation tree.

In-network data aggregation has been shown to reduce the number of messages in the network, often by more than an order of magnitude [MFHH02, SBLC04]. However, in large networks, especially when the monitoring node is several *wireless hops* away, the cost of aggregation may still be significant. In densely distributed networks, proper control on the bandwidth consumed by each running query is essential to guarantee that parts of the network are not overburdened and that all required processing can be performed by the network. For instance, a continuous user query that aggressively computes the total number of moving objects detected by all sensors nodes every second may consume a significant amount of the available bandwidth and essentially block out other significant processing assigned to the nodes.

All the above techniques try to limit the number of transmitted data while always providing accurate answers to posed queries. However, there are many instances where the application is willing to tolerate a specified error in order to reduce the bandwidth consumption and increase the lifetime of the network. Limiting the bandwidth consumed by a posed aggregate query is important to ensure the longevity of the network, since transmission is the biggest source of energy drain in sensor nodes [EGHK99]. This may not be a major concern when sensor nodes are attached to larger devices with ample power supply, but becomes critical when nodes are powered by small batteries. The reduction in bandwidth consumption results

in an equally important reduction in the energy consumption of the sensor nodes, since this is often directly proportional to the number of transmitted and received bits [HCB00, LR02, TK03]. We need to note that, depending on the radio technology used, each transmitted message may drain energy not only from the transmitting node and the intended recipient, but also from other nodes in the vicinity of the transmitting node which also receive the message due to the wireless nature of the communication. As our experimental evaluation demonstrates, often a substantial reduction in the consumed bandwidth during aggregate computation, achieved by suppressing some update messages, only slightly impacts the accuracy of the produced results.

In this chapter we develop new techniques for data dissemination in sensor networks when the monitoring application either is willing to tolerate a specified maximum error threshold, or wants to limit the average bandwidth consumption of the posed query. We refer to the second type of queries as *bandwidth-constrained* queries. Our algorithms initially install an error filter in each node and then modifies these filters in a way that seeks to, depending on the application scenario, either minimize the bandwidth consumption of the query, or provide as strict error guarantees as possible while equating the actual bandwidth consumption to the desired one. The error guarantees and the overall bandwidth consumption are always known to the monitoring node and are appropriately modified periodically, in order to optimize the desired metric. Moreover, we introduce the *residual mode of operation*, during which a parent node may eliminate messages from its children nodes in the aggregation tree, when the cumulative change from these sensor nodes is small. Finally, unlike previous

algorithms that we later detail, our algorithms operate with only local knowledge, where each node simply considers statistics from its children nodes in the aggregation tree. This allows for more flexibility in designing adaptive algorithms, and is a more realistic assumption for sensors nodes with very limited capabilities [MFHH02].

The rest of the chapter is organized as follows. In Section 5.2 we provide an introduction to the data aggregation process. In Sections 5.3 and 5.4 we describe prior techniques that may be suitable for our applications, along with their shortcomings. Section 5.5 presents our extensions and our Potential Gains Adjustment (PGA) algorithm for dynamically adjusting the error thresholds of the sensor nodes when the application specifies a maximum error threshold that it is willing to tolerate, while Section 5.6 contains our experimental evaluation for the same application. Section 5.7 presents our Marginal Gains Adgustment (MGA) algorithm for the problem of bandwidth-constraint queries, while in Section 5.8 we evaluate the performance of our MGA algorithm against prior techniques.

5.2 Basics

In this section we provide some background information needed in our discussion. A description of the characteristics of sensor nodes, focusing on the sources of energy drain in sensor networks, was presented in Section 3.2.1. We first describe the data aggregation process and highlight the major challenges addressed in our algorithms. We then provide an analysis of the expected benefits on the network's lifetime when applying techniques that constrain the bandwidth consumption in these networks.

5.2.1 Data Aggregation Process

We now briefly describe the data aggregation process in sensor networks when 100% accuracy (ignoring network delays, or lost messages) is desired by the querying node and when utilizing the TAG [MFHH02] model to synchronize the transmission of data values by the sensor nodes.

Consider a node *Root*, which initiates a continuous query over the values observed by a set of data sources and requests that the results of this query be reported to it at regular time periods. The time interval between two such consecutive time periods is referred to as the *epoch* of the query. The continuous query is disseminated through the network in search of the sensor nodes that collect data relevant to the posed query. While each such node may have received the announcement of the query through multiple nodes, it only selects one of these nodes as its *parent* node, through which it will propagate its results towards the *Root* node. The flow of the query results forms a tree, rooted at the *Root* node, which is commonly known as the *aggregation tree* [EGHK99, IEGH02, MFHH02]. The query dissemination process and a sample aggregation tree are depicted in Figure 5.1. The nodes in the aggregation tree can be classified as either *active* or *passive*. Active nodes (marked grey in the figure) collect measurements relevant to the query, while passive nodes (marked white in the figure) simply facilitate the propagation of results towards the *Root* node.

At each epoch, each sensor node N_i calculates the partial aggregate corresponding to the query result produced by measurements obtained by sensor nodes in the subtree of N_i. This calculation is performed bottom-up, where each node first waits

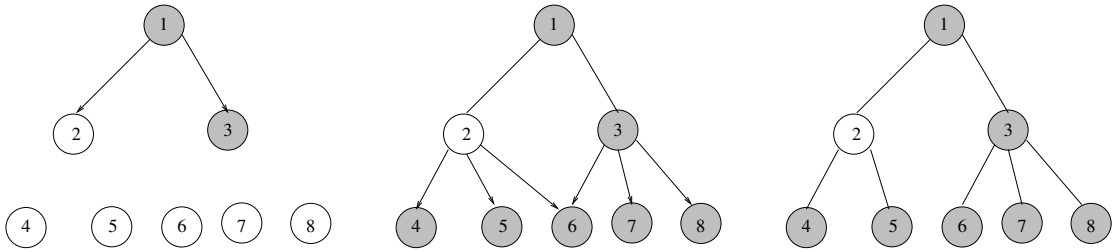


Figure 5.1: Query dissemination process (steps (a) and (b)) and formed aggregation tree (step (c))

to receive any updated partial aggregate values from its children nodes (in the aggregation tree) and then combines these values with its own collected measurements (if this is an *active* node) to produce the partial aggregate for its subtree.

5.2.2 Challenges During In-Network Data Aggregation

We now discuss some challenging characteristics of in-network data aggregation that motivate our techniques.

Hierarchical Structure of Nodes

As we have mentioned above, the sensor nodes that either measure or forward data relevant to a posed continuous query form an aggregation tree, which messages follow on their path to the node that initiated the query. The procedure described in Section 5.2.1 results in a single aggregate value transmitted by each node towards its parent in the aggregation tree.

However, not all kinds of information relevant to the query execution process can be aggregated in the same manner. Consider, for example, a scenario where only a non-predetermined subset of the sensor nodes in the aggregation tree makes

a transmission within each epoch. This scenario is typical, as we will discuss later in this chapter, in the evaluation of approximate aggregate continuous queries. If some application requires that the *Root* node know exactly which nodes made a transmission during each epoch, then each transmitting node needs to piggyback its identifier (id) to each message that it transmits. Note that, because messages are combined in the aggregation tree, in this scenario each message transmitted by a node N_i will contain the node ids of all the transmitting nodes in the subtree of N_i. Obviously, this side information can potentially be excessive, thus resulting in a significant increase in the power consumption by the sensor nodes. Moreover, this information may not even fit within the maximum packet size, thus requiring that it be fragmented and transmitted through multiple messages.

A similar problem occurs whenever a node requires *individual* statistics from the sensors in the aggregation tree. This information cannot be aggregated, since each individual node statistic needs to be accompanied by the node's identifier. Thus, any technique or algorithm that requires individual node statistics will result in the transmission of large amounts of information that may outweigh the benefits of in-network aggregation.

Nodes with Different Characteristics

In a large sensor network, nodes with widely different characteristics may exist. The measurements of some nodes may be either significantly higher or exhibit much larger variance than the measurements of some other nodes. For example, in an application where sensors are used to trace moving objects within their vicinity, some sensor

nodes may detect a large number of moving objects, while others may detect only few, if any. Moreover, the number of detected moving objects over time by each sensor may change either rapidly, if the speed of the objects is significant, or very slowly, if the objects are moving slowly. Throughout this chapter, we refer to sensor nodes that exhibit large variance in their measurements as *volatile* nodes. Proper handling of volatile nodes is crucial, as an ill-designed algorithm may allocate a lot of resources on them at the expense of other nodes in the network.

Negative Correlations in Neighboring Areas

During the data aggregation process, each node calculates the partial aggregate value of its subtree and forwards this new value to its parent node in the aggregation tree. However, there might be cases when changes from nodes belonging to different subtrees of the aggregation tree either cancel out each other, or result in a very small change in the value of the calculated aggregate. This may happen either because of a random behavior of the data, or because of some properties of the measured quantity.

Consider for example the aggregation tree of Figure 5.1(c), and assume that each node observes the number of items moving within the area that it monitors. If some objects move from the area of node 4 to the area of node 5, then the changes that will be propagated to node 2 will cancel out each other. In this case, the partial aggregate value calculated by node 2 does not change and, therefore, there is no need for node 2 to make a transmission. Node 1 may then safely assume that the partial aggregate value of node 2 has not been modified. Even when the overall change of a node's aggregate value is non-zero, but reasonably small, the filtering of transmissions from

this node may result in a large number of saved messages with only minimum effect in the reported aggregate result. In an *approximate* data aggregation application it is crucial to detect and exploit areas where such *negative correlations* occur frequently.

5.2.3 Energy Benefits of Bandwidth-Constrained Queries

We now seek to evaluate how bandwidth-constrained queries reduce the energy drain in sensor nodes. Similar benefits can be also achieved in the dual application scenario, where each aggregate continuous query is associated with a maximum error that the application is willing to tolerate. However, these benefits are easier to quantify using a model for the case of bandwidth-constrained queries.

We construct a simple model that estimates the current total energy of the nodes participating in the evaluation of the continuous query based on the number of transmitted and received messages and the corresponding average transmission and receiving costs (E_T and E_R, respectively) per message. Also, let E_I denote the cost of idle listening, which occurs when a sensor node listens to its channel awaiting for messages. For simplicity, the transmission cost E_T incorporates not only the energy needed to transmit a single message among two nodes, but also any additional energy consumed, which depends on the network protocol being used, in order to reserve the channel for the transmission of this message, or to send/receive acknowledgments for successfully receiving each message. For simplicity, we do not take into account the computational costs, due to the small power consumption required to perform the simple aggregation step in each node. Let $E_T = k \times E_R$, where k denotes the ratio

between the average energy consumed during the transmission of a message and the corresponding energy consumed while receiving a message. Also, let $E_R = p \times E_I$, where p denotes the ratio between the average energy consumed while receiving a message and the corresponding energy consumed during idle listening.

Consider, for simplicity, that the nodes in our model form a full f-ary tree of L levels, meaning that each non-leaf node has exactly f children nodes. Then, the total number of nodes is $T = \sum_{i=0}^{L-1} f^i = \frac{f^L-1}{f-1}$, while the number of non-leaf nodes is $T_{non-leaf} = \sum_{i=0}^{L-2} f^i = \frac{f^{L-1}-1}{f-1} = (T-1)/f$. Using a protocol like TAG [MFHH02], in an unconstrained query evaluation each non-leaf node needs to keep its radio open during an epoch for enough time in order to receive the f messages from its children nodes containing the updated value of the partial aggregate corresponding to their subtrees (see the following sections for details). This listening cost is not incurred by the leaf nodes of the tree. In a full f-ary tree of L levels, the total cost of receiving will thus be exactly:

$$RC_{uncon} = T_{non-leaf} \times (f \times E_R) = (T-1) \times E_R,$$

while the cost of transmitting the aggregate values will be equal to (the *Root* node does not make a transmission):

$$TC_{uncon} = (T-1) \times E_T = k \times RC_{uncon}.$$

On the other hand, a protocol implemented on top of TAG that restricts the band-

width consumed for the execution of the continuous aggregate query to a fraction B_Util of the bandwidth required by an unconstrained evaluation, will result in 1-B_Util of the messages not being transmitted and the transmission energy drain will be:

$$TC_{con} = (T-1) \times B\_Util \times E_T = B\_Util \times k \times RC_{uncon},$$

while the total energy drain spent when either receiving packets or performing idle listening will be:

$$\begin{aligned} RC_{con} &= T_{non-leaf} \times f \times (B\_Util \times E_R + E_I(1 - B\_Util)) \\ &= (T-1) \times E_R \times (B\_Util + (1 - B\_Util)/p) \\ &= (B\_Util + (1 - B\_Util)/p) \times RC_{uncon} \end{aligned}$$

If we consider the time needed for network's energy to reach a value y, when the total initial energy was $C(t_0)$, the unconstrained evaluation will require $t_{uncon} = \frac{C(t_0)-y}{TC_{uncon}+RC_{uncon}} = \frac{C(t_0)-y}{RC_{uncon}(1+k)}$ epochs, while the bandwidth-constrained execution will reach this energy level after $t_{con} = \frac{C(t_0)-y}{RC_{uncon}((1+k-1/p)\times B\_Util+1/p)}$ epochs. Therefore, for any total initial energy level $C(t_0)$, the bandwidth-constrained execution will reach this level y after $\frac{1+k}{(1+k-1/p)\times B\_Util+1/p}$ times more epochs than an unconstrained query execution.

Example 1: . Assuming use of MICA2 nodes at their default and maximum transmission powers (Table 5.1) we get k=1.41 and 3, respectively (ignoring the cost of control messages, retransmissions and acknowledgments), while p=1. For B_Util=6%, the

Characteristic	Value
CPU	7.3828 MHz
Memory	4 KB SRAM, 128 KB FLASH
Additional Storage	32 KB EEPROM
Transmission Range	1000 ft
Battery	2 AA Batteries
Radio Current Draw	25 mA (Transmission max power) 8 mA (receiving) < 1uA (sleep)

Table 5.1: Characteristics of the MICA2 Mote

network lifetime increases by a factor of 2.25 and 3.48, respectively. However, as our experimental evaluation demonstrates, the aggregate computation during this constrained query execution may often incur only a small error (i.e., only 0.1% relative error in Table 5.10). ∎

Depending on the type of protocol (scheduled or contention [YH03]) being used by the sensor nodes, each transmitted message may have many hidden costs associated with it. In many protocols (i.e., the CSMA/CA protocol), prior to the transmission of each message some control messages (RTS/CTS) need to be exchanged in order for the transmitting node to gain access to the channel, or acknowledgments for each received packet often need to be sent. Collisions between transmitted messages may occur and retransmissions (which may result in a significant energy waste) are necessary. By reducing the number of transmitted messages, the number of transmitted control messages and the probability of collisions occurring, and thus the energy consumption by nodes, are greatly reduced. Note that all these costs (including the corresponding receiving costs for all these messages) have been incorporated into the E_T parameter in our model, and thus the actual k value is significantly larger (often by more than

a factor of 5) than simply the power ratio between the transmit and receive modes used in the example above. Finally, we note that in a constrained query execution, each node that decides not to transmit a message may immediately go into a low-duty cycle state and, thus, preserve large amounts of energy.

A point worth mentioning is that our algorithms and analysis for bandwidth-constrained queries focuses on the average energy drain on the network without concern on individual nodes whereabouts. The extensions that we discuss in Section 5.7 allow our techniques to provide for nodes that face severe energy constraints. However, it is our belief that applications are better not utilizing strict controls on the operations of individual nodes. Sensor networks often contain redundant nodes to ensure, for instance, coverage on regions with non-uniform communication density and cope with unexpected node failures. Thus, our focus should not be on extending the lifetime of individual nodes but on ensuring that the network *as a whole* has enough resources to perform the task at hand. Prior work on sensor networks (for instance [CE02, EGHK99, HCB00, Kot05]) has built upon these ideas and has been our inspiration for focusing instead on the average energy drain among all nodes in the aggregation tree.

5.3 Error-Tolerant Applications: Existing Techniques and their Drawbacks

In this section we will demonstrate that straight-forward extensions to the algorithm of [OJW03] for sensor network applications, for applications that seek to minimize the bandwidth consumption of a continuous query given the maximum error that the application is willing to tolerate, result in several shortcomings due to the issues discussed in Section 5.2.2.

The original algorithm of [OJW03] was devised for applications containing a non-hierarchical node setup, where all the nodes in the aggregation tree can be assumed to be direct children of the *Root* node that initiates the query and, therefore, all the messages are aggregated only on that node. Moreover, due to the node setup considered in [OJW03], all the nodes in the aggregation tree collect data relevant to the query (*passive* nodes do not exist).

In our discussion hereafter, we will use the term *Burden-Based Adjustment* (BBA) to refer to the adaptation of the algorithm of [OJW03] for approximate in-network data aggregation, combined with the model of TAG [MFHH02], with the latter being used in order to coalesce messages within the aggregation tree.

5.3.1 Burden-Based Adjustment of Node Filters

Consider a node *Root*, which initiates a continuous query over the values observed by a set of data sources. This continuous query aggregates values observed by the data sources and produces a single aggregate result. For each defined query, a maximum

error threshold, or equivalently a precision constraint *E_Global* that the application is willing to tolerate is specified. The algorithm will install filters at each queried data source, that will help limit the number of transmitted messages from the data source. The selection process for the filters enforces that at any moment after the installation of the query to the data sources, the aggregate value reported at node *Root* will lie within the specified error threshold from the true aggregate value (ignoring network delays, or lost messages).

Initially, a filter F_i is installed in every data source S_i. Each filter F_i is an interval of real values $[L_i, H_i]$ of width $W_i = H_i - L_i$, such that any source S_i whose current observed value $Current_i$ lies outside its filter F_i will need to transmit its newly calculated partial aggregate value, while also taking into account any messages from its children nodes, towards the *Root* node and then re-center its filter around this transmitted value, by setting $L_i = Current_i - W_i/2$ and $H_i = Current_i + W_i/2$. If $Current_i$ lies within the interval specified by the filter F_i, then this value does not need to be transmitted. Note, however, that for any non-leaf node in the aggregation tree, any messages that it receives from its children (unless the resulting aggregate change from these messages is zero) need to be propagated towards the *Root* node, since the node's filter is applied only to the node's observed data value and not on the partial aggregate of its subtree.[1] In this case, the node may include for free in the newly calculated partial aggregate its current observed value and recenter its filter around this value. It is important to emphasize that the initial error guarantees

[1]In the experiments we also investigate the option of applying the error filter to the partial aggregate value of the subtree. However, this modification typically resulted in more transmitted messages than the presented one.

should not be violated by the filter initialization. For example, for the SUM aggregate function the following inequality must be true: $\sum_i \frac{W_i}{2} \leq E\_Global$.

In order for the algorithm to be able to adapt to changes in the characteristics of the data sources, the widths W_i of the filters are periodically adjusted. Every Upd time units, Upd being the *adjustment period*, each filter shrinks its width by a *shrink percentage* ($shrink$). At this point, the *Root* node obtains an *error budget* equal to $(1-shrink) \times E\_Global$, which it can then distribute to the data sources. The decision of which data sources will increase their window W_i is based on the calculation of a *Burden Score* metric B_i for each data source, which is defined as

$$B_i = \frac{C_i}{P_i \times W_i} \tag{5.1}$$

In this formula, C_i is the cost of sending a value from the data source S_i to the *Root* and P_i is the *estimated streamed update period*, defined as the estimated amount of time between consecutive transmissions for S_i over the last period Upd. For a single query over the data sources, it is shown in [OJW03] that the goal would be to try and have all the burden scores be equal.[2] Thus, the *Root* node selects the data sources with the largest deviation from the target burden score (these are the ones with the largest burden scores in the case of a single query) and sends them messages to increase the width of their windows by a given amount. The process is repeated every Upd epochs.

[2]In [OJW03] the case of multiple queries over the data sources is also considered. This is orthogonal to the proposed techniques in our work.

5.3.2 Drawbacks of the BBA algorithm

We now discuss some of the key drawbacks of the BBA algorithm when applied to sensor network applications. Our discussion will be based on the data aggregation characteristics discussed in Section 5.2.2.

Hierarchical Structure of Nodes

In order to calculate the burden score of each sensor N_i, the *Root* node needs to estimate the node's *estimated streamed update period* P_i and the cost C_i of node N_i transmitting values towards the *Root* node. In order for the *Root* to estimate the P_is, each node either needs to transmit at the last epoch of the update period the number of total transmissions that it performed, or piggyback in each message that it transmits its identifier. Obviously, this amount of side information needed is excessive (see Section 5.2.2) and may outweigh the benefits of approximate data aggregation. Note that in a non-hierarchical setup of nodes, this problem would not occur, since the *Root* node would be able to identify from any received packet's header the sender of the message, and accurately compute the number of transmissions by each node.

Calculating the average cost of the transmissions made by a node is more complex. In a non-hierarchical setup of the nodes, this cost could depend on parameters like the bandwidth capacity of the link between each node and the *Root* and could be considered to be either fixed throughout the query execution, or change only occasionally. In a hierarchical setup, this quantity, if measured in the number of messages resulting from each node's transmission, depends on the topology of the other trans-

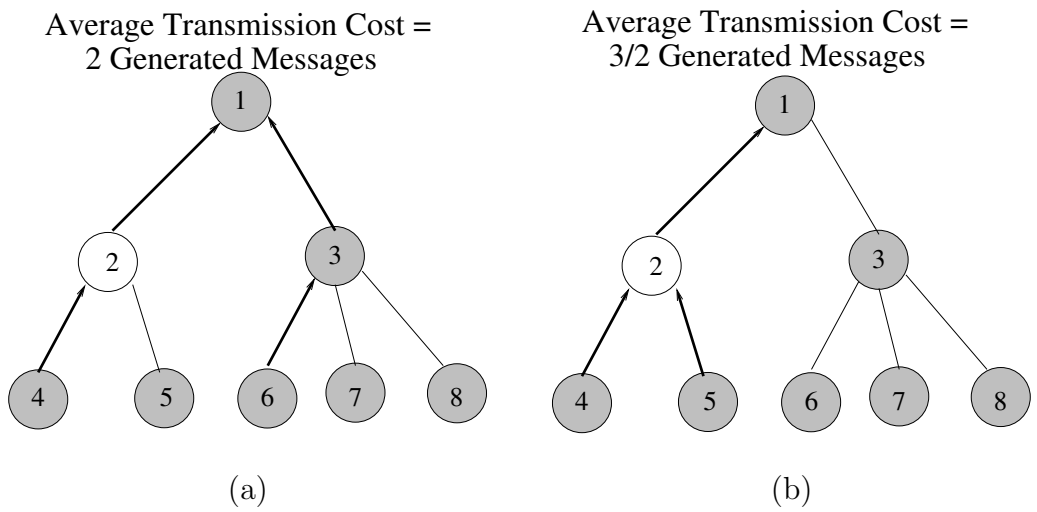


Figure 5.2: Two transmissions scenarios with different costs for each transmission

mitting nodes in the aggregation tree. This point can be more easily understood with an example. Consider the two scenarios depicted in Figure 5.2. In both scenarios, only two nodes make a transmission (the transmitted messages are depicted by bold, thick arrows). However, the transmitted messages are aggregated at different nodes of the aggregation tree. In the first scenario (Figure 5.2(a)), nodes 4 and 6 make a transmission, and nodes 2 and 3 propagate these messages towards the *Root* node. In this case, each transmission from nodes 4 and 6 is responsible for generating 2 messages. On the other hand, in the second scenario, (Figure 5.2(b)) nodes 4 and 5 make a transmission, and node 2 propagates a single message to node 1. Therefore, each transmission is responsible for only 3/2 messages in this case.

To calculate the actual cost C_i (in number of generated messages) for each node transmission (or an average cost over multiple transmissions), the *Root* node requires knowledge of not only which nodes made a transmission within each epoch, but also of the exact topology (parent-child relationships) of these nodes and, furthermore, whether these nodes made a transmission because their monitored value laid outside

the node's filter, or simply made a transmission to forward changes in their calculated partial aggregate because of transmissions by some of their descendants. However, this is a completely unrealistic scenario, since too much information would need to be communicated, namely the exact topology and the root-cause of each transmission. Therefore, the techniques introduced in [OJW03] can be applied in our case only by using a heuristic function to estimate C_i. In Section 5.6 we describe such a heuristic.

Nodes with Different Characteristics

One of the principle ideas behind the adaptive algorithms presented in [OJW03] is that an increase in the width of a filter installed in a node will result in a decrease at the number of transmitted messages by that node. While this is an intuitive idea, there are many cases, even when the underlying distribution of the observed quantity does not change, where an increase in the width of the filter does not have any impact in the number of transmitted messages. To illustrate this, consider a node whose values follow a random step pattern, meaning that the observed value at each epoch differs by the observed value in the previous epoch by either $+\Delta$ or $-\Delta$. In this case, any filter with a window whose width is less than $2 \times \Delta$ will not be able to reduce the number of transmitted messages. A similar behavior may be observed in cases where the measured quantity exhibits a large variance. In such cases, even a filter with considerable width may not be able to reduce but a few, if any, transmissions.

The main reason why this occurs in the BBA algorithm is because the burden score metric being used does not give any indication about the expected benefit that

we will achieve by increasing the width of the installed filter at a node. In this way, a significant amount of the maximum error budget that the application is willing to tolerate may be spent on a few nodes whose measurements exhibit the aforementioned volatile behavior (note that due to the large number of their transmissions these nodes will also exhibit large burden scores), without any real benefit.

Negative Correlations in Neighboring Areas

According to the algorithms in [OJW03], each time the value of a measured quantity at a node N_i lies outside the interval specified by the filter installed at N_i, then the new calculated partial aggregate value of the node is transmitted and propagated to the *Root* node. In this case, negative correlations, such as the ones described in Section 5.2.2 are not exploited and messages cannot be prevented from reaching the *Root* node. Even when we modify the BBA algorithm to take into account negative correlations (see Section 5.6), performance is often worse, because BBA cannot distinguish on the true cause of a transmission (change on local measurement or change in the subtree).

5.4 Bandwidth-Constrained Queries: Existing Techniques and their Drawbacks

We now provide a description of an alternative technique that can potentially be used, for the hierarchical in-network evaluation of bandwidth-constrained continuous aggregate queries in the area of sensor networks and draw direct comparisons to

our algorithms. The *threshold-based adjustment* (TBA) algorithm presented below is motivated by the work of Olston et al. on determining when cached objects should be refreshed based on a defined priority function [OW02]. A detailed description of the algorithm and justification of its decisions can be found in [OW02]. We need to note that we also experimented with some additional techniques that can also be used in our application. These techniques are based on either uniform or biased sampling of the data sources. However, due to the inability of these alternate techniques to provide strong deterministic error guarantees and their poor performance in our experimental evaluation, we omit their discussion.

Recently, novel algorithms that build probabilistic models of the observed data, and then use these models to probe the sensors for their measurements in a limited amount of epochs, depending on the confidence of the constructed model, have been proposed [DGM+04, LM04]. While these techniques may result in a significant reduction in the number of transmitted messages, they present several drawbacks. First of all, these techniques cannot provide strong deterministic guarantees on the quality of the produced result. The provided guarantees are only probabilistic ones, and the accuracy of these guarantees depends on whether the real-time observed data are similar to the training data used for building the model. As the authors of [DGM+04] state, "for models to perform accurate predictions they must be trained in the kind of environment where they will be used". While the acquisition of the appropriate set of training data may be feasible in scenarios of controlled environments, such as temperature monitoring applications within a lab, in applications where the sensors are thrown over hostile environments or disaster areas this is an unrealistic assump-

tion. Moreover, the techniques in [DGM+04] cannot be used for the detection of outliers, meaning events or measurements that deviate significantly from either the corresponding values observed in other sensors, or in the same sensor but over prior time periods. However, the purpose of *monitoring* applications is often to detect such large deviations from the normal behavior, since these deviations may require trigger some alert or require appropriate action to be taken. We thus omit these techniques from our discussion.

5.4.1 Threshold-Based Adjustment (TBA)

We now describe how the TBA algorithm can be adapted for the problem of bandwidth-constrained queries over sensor networks. Each active node i in the aggregation tree maintains the following statistics at each epoch t_i:

- The value $V_{t_{now}}$ of the node's monitored quantity at the current epoch.
- The last transmitted measurement $V_{t_{last}}$ of this node and the time (epoch) of the last transmission t_{last}.
- A threshold value Thr_i that will help determine the time of the node's next transmission.
- A priority value Pr_i calculated as:

$$Pr_i = (t_{now} - t_{last}) * \max_{t \in [1+t_{last}, t_{now}]} |V_t - V_{t_{last}}| - \int_{t_{last}}^{t_{now}} |V_t - V_{t_{last}}| dt$$

We here note that the definition of the priority function has been slightly modified (to include the *max* quantity) from the formula in [OW02], since in our case, the deviation value $|V_t - V_{t_{last}}|$ is **not** a non-decreasing function of t. The *max* quantity is therefore needed to ensure that the calculated priority is always an non-decreasing and non-negative value, as required in [OW02].

Each time a node's priority value Pr_i exceeds the node's threshold value Thr_i, the node performs the following operations:

- Increases its threshold value by a factor θ; that is: $Thr_i = Thr_i \times \theta$.
- Transmits the difference $V_{t_{now}} - V_{t_{last}}$ and its current threshold towards its parent node in the aggregation tree.
- Sets $V_{t_{last}} = V_{t_{now}}$, $t_{last} = t_{now}$ and $Pr_i = 0$.

Each node N_i may also transmit the difference $V_{t_{now}} - V_{t_{last}}$ if one of its descendant nodes makes a transmission. In this case, node N_i can include the above difference at no cost (after aggregating it with the one received by its child node), since any messages received from descendant nodes will need to be propagated towards the *Root* node anyway. In this case, the node performs most of the steps described above, but does not increase its threshold value (neither transmits it), since its transmission was due to another node's measurements.

Note that the TBA algorithm does provide deterministic error guarantees, since at each node i, its current value cannot deviate by more than Thr_i without resulting in a transmission by the node. Therefore, the application error guarantee is equal to the sum of threshold values by all active nodes in the aggregation tree.

The *Root* node continuously monitors the thresholds received from the nodes and, if it detects that the bandwidth is underutilized (the estimation of the bandwidth consumption can be performed in the same way as in our algorithm), then it sends feedback messages to the nodes with the highest thresholds to divide their thresholds by a parameter $\omega > 1$. Trying to keep the thresholds of all active nodes about equal was shown in [OW02] to be the optimal solution, for a different problem though than the one that we tackle in our work, and in a non-hierarchical setup of the nodes.

5.4.2 Drawbacks of the TBA Algorithm

We now discuss some of the key drawbacks of the TBA algorithm when applied to sensor network applications. Our discussion is based on the data aggregation characteristics discussed in Section 5.2.2. We omit the discussion about the drawbacks of the TBA algorithm with respect to exploiting negative correlations in neighboring areas, since they are identical to the corresponding drawbacks of the BBA algorithm (Section 5.3.2).

Nodes with Different Characteristics

The TBA algorithm is only influenced by the variance of the measurements, and not by their magnitude. However, the TBA algorithm fails to take into account that some nodes may be *volatile*, meaning nodes that exhibit large variance in their measurements. Such nodes tend to continuously make transmissions, since their thresholds are usually not sufficiently large to prune any messages, thus significantly raising their thresholds. On the update phase of the TBA algorithm, due to their increased

threshold values, these same nodes are the ones which the *Root* node will ask to lower their thresholds, a process which results in all the other non-volatile nodes to gradually increase their thresholds to large values, thus resulting in large maximum errors for the application. However, it is obvious that the desired behavior would be for the algorithm to eliminate these volatile nodes from consideration (by assigning them a zero, or near-zero threshold). This would still result in a continuous transmission by these nodes, but the thresholds of the remaining nodes would be considerably lower, thus resulting in tighter error guarantees.

Hierarchical Structure of Nodes

One of the significant drawbacks of using the TBA algorithm for evaluating bandwidth-constrained queries over sensor networks is the fact that the TBA algorithm does not exploit or use the hierarchical structure of the aggregation tree. Deciding the set of nodes whose threshold values will be decreased is based solely on the threshold values, and not on the topology of the nodes. However, the tree topology should clearly be taken into account, since the transmission by any node N_i causes the transmission of messages by all the ancestors of N_i in the aggregation tree.

Moreover, in the TBA algorithm each message from node N_i to its parent is accompanied by a potentially large list of identifiers of nodes which increased their thresholds in the subtree of N_i. This information may be quite large, since it cannot be aggregated. Furthermore, the algorithm needs to estimate the amount of consumed bandwidth, which requires some additional, but easy to aggregate, information to be propagated as well. Our MGA algorithm, as we will explain in Section 5.7 requires

Symbol	Description
N_i	Sensor node i
W_i	The width of the filter of sensor N_i
$E_i = W_i/2$	Maximum permitted error in node N_i
E_Sub_i	Maximum permitted error in entire subtree of node N_i
E_Global	Maximum permitted error of the application
V_Curr	The latest measurement obtained by the node (if active)
Upd	Update period of adjusting error filters
$shrink$	Shrinking factor of filter widths
T	Number of nodes in the aggregation tree
$Root$	The node initiating the continuous query
$Gain$	The estimated gain of allocating additional error to the node
$CumGain$	The estimated gain of allocating additional error to the node's entire subtree
$CumGain\_Sub[i]$	The estimated gain of allocating additional error to the node's i-th subtree

Table 5.2: Symbols Used in our Algorithms

the transmission of three quantities which, however, is information that can be easily aggregated.

5.5 Our PGA Algorithm

In this section we first provide a high-level description of our framework and then present the details of our algorithms for dynamically modifying the widths of the filters installed in the sensor nodes, when the application is willing to tolerate a maximum error in its reported aggregate. The notation that we will use in the description of our Potential Gains Adjustment (PGA) algorithm is presented in Table 5.2.

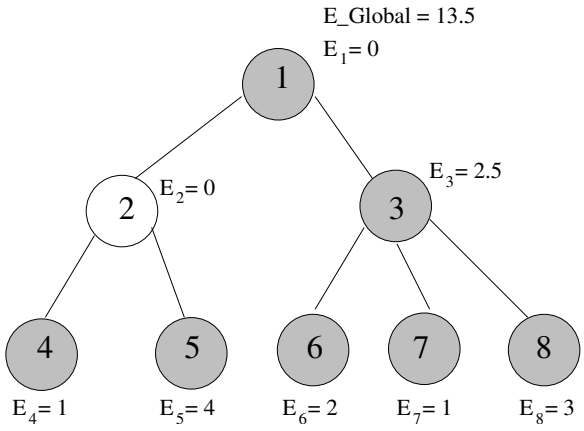


Figure 5.3: Sample Aggregation Tree

5.5.1 Description of our Framework

We assume that the aggregation tree (i.e., Figure 5.3) for computing and propagating the aggregate has already been established. Techniques for discovering and modifying the aggregation tree are illustrated in [MFHH02]. There are two types of nodes in the tree. *Active* nodes, marked grey in the figure, are nodes that collect measurements. *Passive* nodes (for example, node 2 in the figure) are intermediate nodes in the tree that do not record any data for the query but rather aggregate partial results from their descendant nodes. By default all leaf nodes are active, while intermediate nodes may be either active or passive. Our algorithms will install a filter to each node N_i in the aggregation tree, independently on whether the node is an active or passive one. This is a distinct difference from the framework of [OJW03], where filters are assigned only to active nodes.

In our discussion we focus on queries containing the SUM aggregate function. The COUNT function can always be computed exactly as the number of active nodes in the aggregation tree, while the AVG function can be computed by the SUM and

COUNT aggregates. As the work in [OJW03] demonstrated, adaptive filter adjustment algorithms for the MAX and MIN aggregate functions make sense only when considering a multi-query optimization scenario.

Figure 5.3 shows the maximum error of each filter for a query calculating the SUM aggregate over the active nodes of the tree.[3] Notice that the sum of the errors specified is equal to the maximum error that the application is willing to accept (*E_Global*). Moreover, there is no point in placing an error filter in the *Root* node, since this is where the result of the query is being collected.[4]

We now describe the protocol of propagating values in the aggregation tree. In [MFHH02], ways of synchronizing the radios between parent and children nodes in the aggregation tree are described, which try to minimize the amount of time that sensors need to have their radios open in order to receive measurements from their children, aggregate them and transmit them to their parent node. In our work we assume a similar synchronization process. During an epoch duration and within the time intervals specified in [MFHH02] the sensor nodes in our framework operate as follows:

- An active leaf node i obtains a new measurement and forwards it to its parent if the new measurement lies outside the interval $[L_i, H_i]$ specified by its filter.

- A passive (non-leaf) node awaits for messages from its children. If one or more messages are received, they are combined and forwarded to its own parent only

[3]The width of the error filter in node 2 may in general be non-zero in our algorithms.

[4]This can change, when the *Root* node collects and transmits the aggregate to a distant base station. The modifications to all algorithms considered here are straightforward.

if the new partial aggregate value of the node's subtree does not lie within the interval specified by the node's filter. Otherwise, the node remains idle.

- An active non-leaf node obtains a new measurement and waits for messages from its children nodes as specified in [MFHH02]. The node then recomputes the partial aggregate on its subtree (which is the aggregation of its own measurement with the values received by its child-nodes) and forwards it to its parent only if the new partial aggregate lies outside the interval specified by the node's filter.

Along this process, the value sent from a node to its parent is either (i) the node's measurement if the node is a leaf or (ii) the partial aggregate of all measurements in the node's subtree (including itself) if the node is an intermediate node. In both cases, a node remains idle during an epoch if the newly calculated partial aggregate value lies within the interval $[L_i, H_i]$ specified by the node's filter. This is a distinct difference from [OJW03], where the error filters are applied to the values of the data sources, and not on the partial aggregates calculated by each node.

Details on the operation of the sensor nodes will be provided in the following subsection. Compared to the work of [OJW03] we introduce two new ideas for the approximate evaluation of aggregate queries:

1. A new algorithm called *PGA* (Potential Gains Adjustment) for adjusting the widths of filters in the nodes. *PGA* bases its decisions on estimates of the expected gain of allocating additional error to different subtrees. In this way, our algorithm is more robust to the existence of volatile nodes, nodes where the

value of the measured quantity changes significantly in each epoch. Moreover, the estimation of gains is performed based only on local statistics for each node (*that take into account the tree topology*), in contrast to BBA where sources are independent and a significant amount of information needs to be propagated to the *Root* node. Each time a node N_i transmits a message to its parent node in the aggregation tree, then N_i also includes in that message a single value, which is an estimate of the cumulative gain expected when allocating additional error to its entire subtree. These gains are aggregated by the parent of the node, so that only a single value (the cumulative gain) is sent along with the partial aggregate.[5]

2. A hierarchical-based mode of operation: The filters in non-leaf nodes are used in a mode that may potentially filter messages transmitted from their children nodes, and not just from the node itself. We denote this type of operation as *residual-based* operation, and also denote the error assigned to each node in this case as a *residual* error. We show that under the *residual-based* mode nodes may transmit significantly fewer messages than in a *non-residual* operation because of the coalescing of updates that cancel out and are not propagated all the way to the *Root* node. Note that, in contrast, the BBA algorithm, where the filters are applied to each node's measurements and not its calculated partial aggregate, corresponds to the *non-residual* mode of operation of the nodes.

[5]The cumulative gains can also be transmitted only during the last epoch of the update period, to limit the amount of side information transmitted over the network.

5.5.2 Operation of Nodes

The operation of each sensor node is described in Figure 5.4 (notation from Table 5.2). The algorithms consists of four major tasks: *initialization*, *adjustment of filters*, *aggregation* and *transmission of new aggregate*. These tasks are discussed in detail below.

Initialization (Lines 1–3)

A filter is initially installed in each node of the aggregation tree, except from the *Root* node (Line 1). The initial width of each filter is important only for the initial stages of the network's operation, as our dynamic algorithm will later adjust the sizes of the filters appropriately. In our experiments we initialize the widths of the error filters similarly to the *uniform allocation* method. For example, in the case when the aggregate function is the function SUM and there are N_{active} active nodes in the aggregation tree (excluding the *Root* node) then each active node is assigned the same fraction E_Global/N_{active} of the error E_Global that the application is willing to tolerate.

We note that E_i (Line 1) is the maximum permitted error in node N_i, while E_Sub_i is the maximum permitted error in the entire subtree of node N_i. Thus, for the SUM function, E_Sub_i is the sum of E_i and all E_j, where N_j is a descendant of node N_i in the aggregation tree.

procedure NodeOperation(E_Sub)
Input: The maximum permitted error for the subtree of this node E_Sub
begin
// E is the total maximum permitted error of the node itself
// V_Self is the value of the node's measured quantity at its last transmission
// $LastReceived[i]$ is the last received partial aggregate value of the node's i-th subtree
1. In each node initialize E_i using uniform allocation policy and calculate E_Sub_i
2. $NewAggr = 0$ // Current partial aggregate
3. $LTA = 0$ // Last transmitted partial aggregate
// Every Upd epochs the widths of the filters will shrink
4. **for** each epoch ep **do**
5. **if** $ep > 0$ **AND** ep modulo $Upd = 0$ **then**
6. $E\_Sub = shrink * E\_Sub$ // $0 \leq shrink < 1$
7. $E = shrink * E$
8. **if** received message from father to increase error of subtree by E_Additional **then**
9. $E\_Sub+ = E\_Additional$
10. Distribute $E\_Additional$ to self and subtrees and clear all gain related statistics
11. **if** node is active **then**
12. Get current measurement V_Curr
13. Wait for messages from children nodes.
14. ΔChildrenAggr = 0
15. **for** Each Child i **do**
16. **if** i transmitted an aggregate value V_i and its cumulative gain $CumGain_i$ **then**
17. $\Delta ChildrenAggr \mathrel{+}= V_i - LastReceived[i]$ // Needed for non-residual operation
18. $LastReceived[i] = V_i$
19. $CumGain\_Sub[i] = CumGain_i$ // Store the cumulative gain of the node's subtrees
20. **endif**
21. **endfor**
22. $NewAggr = Combine(LastReceived, V\_Curr)$
23. $(Gain, CumGain)$ = UpdateExpectedGain($NewAggr, LTA, E, E\_Sub, Gain, CumGain\_Sub$)
24. **if** (ResidualOperation == false **AND** $((\Delta ChildrenAggr > 0)$ **OR** $|V\_Self - V\_Curr| > E))$ **OR** (ResidualOperation == true **AND** $|NewAggr - LTA| > E$) **then**
25. $V\_Self = V\_Curr$
26. $LTA = NewAggr$
27. Transmit $(NewAggr, CumGain)$ to parent node and re-center the error filter
28. **endif**
29. **endfor**
end

Figure 5.4: Operation of Nodes

Adjustment of Filters (Lines 5–10)

This adjustment phase is performed every Upd epochs. The first step is for all nodes to shrink the widths of their filters by a shrinking factor $shrink$ ($0 \leq shrink < 1$). After this process, the *Root* node has an error budget of size $E\_Global \times (1 - shrink)$, where E_Global is the maximum error of the application, that it can redistribute

Aggregate Function	Implementation of Combine Function
SUM/AVG	$V\_Curr + \sum_i LastReceived[i]$
MAX	$\max\{V\_Curr, \max_i\{LastReceived[i]\}\}$
MIN	$\min\{V\_Curr, \min_i\{LastReceived[i]\}\}$

Table 5.3: Definition of the `Combine` function

recursively to the nodes of the network (Lines 8-10). This redistribution process is done using a statistic called the *cumulative gain* of the node, which is a single value and is the only statistic propagated to the parent node at each transmission. Details of the adjustment process will be given later in this section. At each epoch the node also updates some statistics (Line 23), which will be later used to adjust the widths of the filters.

Aggregation (Lines 11–22)

In each epoch, the node obtains a measurement related to the observed quantity if it is an active node (Lines 11-12), and then waits for messages from its children nodes containing updates to their measured aggregate values (Line 13). We here note that each node computes a partial aggregate based on the values reported by its children nodes in the tree. This is a recursive procedure which ultimately results in the evaluation of the aggregate query at the *Root* node. After waiting for messages from its children nodes, the current node computes the new value of the partial aggregate based on the most current partial aggregate values it has received from its children (Line 22). Variable $LastReceived[i]$ stores the last received partial aggregate value of the root of node's i subtree (Line 18).

Aggregation is performed through a call to the *Combine* function. The specific

implementation depends on the aggregate function specified by the query. In Table 5.3 we provide its implementation for the most common aggregate functions. In the case of the AVG aggregate function, we calculate the sum of the values observed at the active nodes, and then the *Root* node will divide this value with the number of active nodes participating in the query.

Transmission of New Aggregate (Lines 24–27)

After calculating the current partial aggregate, the node must decide whether it needs to transmit a measurement to its parent node or not. This depends on the operation mode being used. In a *non-residual* mode, the node would have to transmit a message either when the value of the measured quantity at the node itself lies outside its filter, or when at least one of the subtrees has transmitted a message and the new changes do not exactly cancel out each other ($\Delta ChildrenAggr > 0$). This happens because in the *non-residual* mode (e.g. the original algorithm of [OJW03]) the error filters are applied to the values measured by each node, and not to the partial aggregates of the subtree. On the contrary, in a *residual* mode of operation, which is the mode used in our algorithms, the node transmits a message only when the value of the new partial aggregate lies outside the node's filter. In both modes of operation the algorithm that distributes the available error enforces that for any node N_i, its calculated partial aggregate will never deviate by more than E_Sub_i from the actual partial aggregate of its subtree (ignoring propagation delays and lost messages). When a node makes a transmission, it caches its current state that includes its latest measurement V_Curr (that is copied to variable V_Self).

Example 2: Consider the aggregation tree of Figure 5.3. Assume that the posed query involves the sum of values in the active nodes of the tree (all nodes except for node 2), and that the maximum error that the application is willing to tolerate is 13.5, as shown in Figure 5.3. We will explain in detail the transmission of messages for both the residual and the non-residual modes of operation, for the sample error filters shown in the figure.

Residual Mode: In the residual mode, the filter of each node is applied to the partial aggregate that it calculates for the subtree rooted at the node. In Table 5.4 we present an example based on the aggregation tree of Figure 5.3. In this table we show the current observed values (V_Curr), the newly calculated partial aggregate value ($NewAggr$) and the last transmitted partial aggregate value of each node (LTA), the difference between these two values ($Diff$) and whether the node makes a transmission or not based on whether the absolute value of this deviation is greater than the maximum permitted error in the node ($|Diff| > E_i$). Notice that whenever a node makes a transmission, then the values of LTA are modified in the next epoch. Moreover, since we are using the model of TAG, each non-leaf node first receives (any) messages from its children nodes and then calculates the new estimate of its partial aggregate.

Non-Residual Mode: In the non-residual mode, the filter of each node is applied to the measurements of the node itself. In Table 5.5 we present an example based on the aggregation tree of Figure 5.3 (same notation as above). In this table we show the current observed values (V_Curr), the measurement of the node at its last transmission (V_Self), the difference between these two values ($Diff$), the last

		Epoch 1					Epoch 2				
Node	E_i	V_Curr	NewAggr	LTA	Diff	Transmit?	V_Curr	NewAggr	LTA	Diff	Transmit?
4	1	20	20	19	1	NO	21	21	19	2	YES
5	4	50	50	45	5	YES	51	51	50	1	NO
6	2	10	10	7	3	YES	9	9	10	-1	NO
7	1	25	25	24	1	NO	23	23	24	-1	NO
8	3	12	12	16	-4	YES	17	17	12	5	YES
2	0	–	69 (19+50)	64	5	YES	–	71 (21+50)	69	2	YES
3	2.5	19	65 (10+24+12+19)	67	-2	NO	17	68 (10+24+17+17)	67	1	NO
1	0	30	166 (69+67+30)	160	6	N/A	28	166 (71+67+28)	166	0	N/A

Table 5.4: Node Operation in Residual Mode

		Epoch 1						Epoch 2					
Node	E_i	V_Curr	V_Self	Diff	LTA	NewAggr	Transmit?	V_Curr	V_Self	Diff	LTA	NewAggr	Transmit?
4	1	20	19	1	19	20	NO	21	19	2	19	21	YES
5	4	50	45	5	45	50	YES	51	50	1	50	51	NO
6	2	10	7	3	7	10	YES	9	10	-1	10	9	NO
7	1	25	24	1	24	25	NO	23	24	-1	24	23	NO
8	3	12	16	-4	16	12	YES	17	12	5	12	17	YES
2	0	–	–	–	64	69 (19+50)	YES	–	–	–	69	71 (21+50)	YES
3	2.5	19	20	-1	67	65 (10+24+ 12+19)	YES	17	19	-2	65	68 (10+24+ 17+17)	YES
1	0	30	29	1	160	164 (69+65+30)	N/A	28	30	-2	164	167 (71+68+28)	N/A

Table 5.5: Node Operation in Non-Residual Mode

transmitted aggregate value of each node (LTA), the newly calculated aggregate value ($NewAggr$), and whether the node makes a transmission or not based on its current measurement and any received messages from its children nodes. Any active node N_i makes a transmission whenever the absolute value of $Diff$ is larger than E_i. Any non-leaf node also makes a transmission whenever it has received an updated value from at least one of its children, and these updates do not exactly cancel out each other. Note that for leaf nodes the mode of operation (residual or non-residual) makes no difference in their functionality. ∎

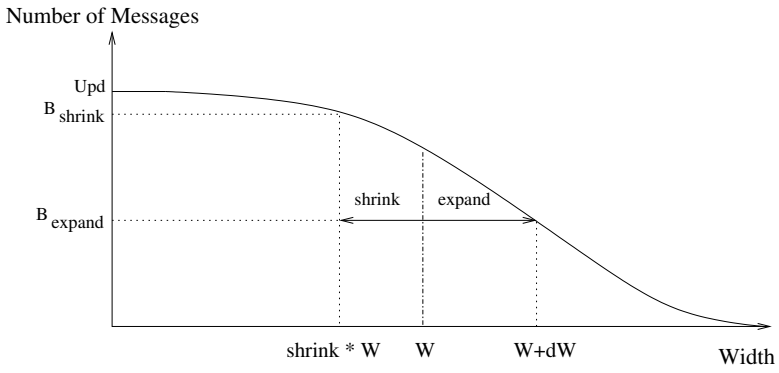


Figure 5.5: Potential Gain of a Node

5.5.3 Calculating the Potential Gain of each Node

Our algorithm updates the width of the filter installed in each node by considering the potential gain of increasing the error threshold at a sensor node, which is defined as the amount of messages that we expect to save by allocating more resources to the node. The result of using this gain-based approach is a robust algorithm that respects the hierarchy imposed by the aggregation tree and, at the same time, is able to identify volatile data sources and eliminate them from consideration. This computation of potential gains, as we will show, requires only local knowledge, where each node simply considers statistics from its children nodes in the aggregation tree.

In Figure 5.5 we show the expected behavior of a sensor node N_i, varying the width of its filter W_i. The y-axis plots the number of messages sent from this node to its parent in the aggregation tree in a period of Upd epochs. Assuming that the measurement on the node is not constant, a zero width filter ($W_i = E_i = 0$) results in one message for each of the Upd epochs. By increasing the width of the filter, the number of messages is reduced, up to the point that no messages are required. Of course, in practice this may never happen as the width of the filter required may

exceed the global error constraint E_Global. Some additional factors that can make a node deviate from the typical behavior of Figure 5.5 also exist. As an example, the measurement of the node may not change for some period of time exceeding Upd. In such a case, the curve becomes a straight line at y=0 and no messages are sent (unless there are changes on the subtree rooted at the node). In such cases of very stable nodes, we would like to be able to detect this behavior and redistribute the error to other, more volatile nodes. At the other extreme, node N_i may be so volatile that even a filter of considerable width will not be able to suppress any messages. Thus, the curve becomes a straight line at y=Upd. Notice that the same may happen because of a highly volatile node N_j that is a descendant of N_i in the aggregation tree.

In principle, we cannot fully predict the behavior of a node N_i unless we take into account its interaction will all the other nodes in its subtree. Of course, a complete knowledge of this interaction is infeasible, due to the potentially large amounts of information that are required, as described in Section 5.2.2. We will thus achieve this by computing the potential gains of adjusting the width of the node's filter W_i, using simple *local* statistics that we collect during the query evaluation.

Let W_i be the width of the filter installed at node N_i at the last update phase. The node also knows the $shrink$ value that is announced when the query is initiated. Unless the adaptive procedure decides to increase the error of the node, its filter's width is scheduled to be reduced to $shrink \times W_i$ in the next update phase, which takes place every Upd epochs. The node can estimate the effects of this change as follows. At the same time that the node uses its filter W_i to decide whether or not to

send a message to its parent, it also keeps track of its decision assuming a filter of a smaller width of $shrink \times W_i$. This requires a single counter B_{shrink} that keeps track of the number of messages that the node would have sent if its filter was reduced. B_{shrink} gives as an estimate of the negative effect of reducing the filter of N_i. Since we would also like the node to have a chance to increase its filter, the node also computes the number of messages B_{expand} in case its filter was increased by a factor dW to be defined later.[6]

Our process is demonstrated in Figure 5.5. Let $DB \geq 0$ be the reduction in the number of messages by changing the width from $shrink \times W_i$ (which is the default in the next update phase) to $W_i + dW$. The *potential gain* for the node is defined as:

$$Gain_i = DB = B_{shrink} - B_{expand}$$

It is significant to note that our definition of the *potential gain* of a node is independent on whether the node is active or not, since the algorithm for deciding whether to transmit a message or not is only based on the value of the partial aggregate calculated for the node's entire subtree. Moreover, the value of dW is not uniquely defined in our algorithms. In our implementation we are using the following heuristics for the computation of gains:

- For leaf nodes, we use $dW = \frac{E\_Global}{N_{active}}$, N_{active} being the number of active nodes

[6]Even though this computation based on two anchor points may seem simplistic, there is little more that can truly be accomplished with only local knowledge, since the node cannot possibly know exactly which partial aggregates it would have received from its children in the case of either a smaller or a larger filter, because these partial aggregates would themselves depend on the corresponding width changes in the filters of the children nodes.

in the aggregation tree.

- For non-leave nodes, in the residual mode, we need a larger value of dW, since the expansion of the node's filter should be large enough to allow the node to coalesce negative correlations in the changes of the aggregates on its children nodes. As a heuristic, we have been using $dW = num\_children_i \times \frac{E\_Global}{N_{active}}$, where $num\_children_i$ is the number of children of node N_i.

These values of dW have been shown to work well in practice on a large variety of tested configurations. We need to emphasize here that these values are used to give the algorithm an estimate on the behavior of the sensor and that the actual change in the widths W_i of the filters will also be based on the amount of "error budget" available and the behavior of all the other nodes in the tree.

Computation of Cumulative Gains

The computation of the potential gains, as explained above, provides us with an idea of the effect that modifying the size of the filter in a node may have, but is by itself inadequate as a metric for the distribution of the available error to the nodes of its subtree. This happens because this metric does not take into account the corresponding gains of descendant nodes in the aggregation tree. Even if a node may have zero potential gain (this may happen, for example, if either the node itself or some of its descendants are very volatile), this does not mean that we cannot reduce the number of transmitted messages in some areas of the subtree rooted at that node.

Because of the top-down redistribution of the errors that our algorithm applies

(using the *AdjRoot* algorithm described below), if no budget is allocated to N_i by its parent node then all nodes in the subtree of N_i will not get a chance to increase their error thresholds, and this will eventually lead to every node in that subtree to send a new message on each epoch, which is clearly an undesirable situation. Thus, we need a way to compute the *cumulative gain* on the subtree of N_i and base the redistribution process on that value. In our framework we define the cumulative gain on a node N_i as:

$$CumGain_i = \begin{cases} Gain_i & N_i \text{ is a leaf node} \\ Gain_i + \sum_{N_j \in children(N_i)} CumGain\_Sub[j] & \text{otherwise} \end{cases} \tag{5.2}$$

This definition of the cumulative gain has the following desirable properties:

1. It is based on the computed gains ($Gain_i$) that is purely a local statistic on a node N_i.

2. The recursive formula can be computed in a bottom-up manner by having nodes piggy-back the value of their cumulative gain in each message that they transmit to their parent along with their partial aggregate value. This is a single number that is being aggregated in a bottom-up manner, and thus poses a minimal overhead. Moreover, transmitting the cumulative gain is necessary only if its value has changed (and in most cases only if this change is significant) since the last transmission of the node.

5.5.4 Adjusting the Filters

We here present two algorithms for adjusting the width of the filters on the nodes. Both algorithms make their decisions using the cumulative gains calculated at each node. They differ in that in the first algorithm, denoted as *AdjRoot*, the *Root* node is the one initiating the process based on the available error budget generated from shrinking the filters. In contrast, in the second algorithm that we denote as *AdjLocal*, this process happens in a localized manner on a level by level basis in the aggregation tree. Below we provide details for both algorithms.

5.5.5 The *AdjRoot* Algorithm

Every *Upd* epochs, all the filters shrink by a factor of *shrink* (see Figure 5.4, Lines 5–7). This results in an error budget of $E\_Global \times (1 - shrink)$ which the *Root* node can distribute to the nodes of the tree. Each node N_i has statistics on the potential gain of allocating error to the node itself ($Gain_i$), and the corresponding cumulative gain of allocating error to each of its subtrees.

Assuming that the node can distribute a total error of $E\_Additional$ to itself and its descendants (Lines 8–10), the allocation of the errors is performed as follows:[7]

- For each subtree j of node N_i, increase E_Sub_j proportionally to its cumulative gain:

$$E\_Additional_j = \frac{E\_Additional \times CumGain\_Sub[j]}{Gain_i + \sum_{N_j \in children(N_i)} CumGain\_Sub[j]} \tag{5.3}$$

[7]For the *Root* node, $E\_Additional = E\_Global \times (1 - shrink)$

This distribution is performed only when this quantity is at least equal to E_Global/N_{active}.

- The remaining error budget is distributed to the node itself.

The fraction of the error budget allocated to the node itself and to each of the subtrees is analogous to the expected benefit of each choice. The use of the computed local gain on the node in comparison to the cumulative gains of its subtrees, allows us to differentiate on the true cause of the transmissions coming out of this node.

The only additional detail is that in case when the error allocated to a subtree of node N_i is less than the E_Global/N_{active} value, then we do not allocate any error in that subtree, and allocate this error to node N_i itself. This is done to avoid sending messages downwards the aggregation tree for adjusting the filters when the error budget is too small.

5.5.6 The *AdjLocal* Algorithm

In the *AdjLocal* algorithm, the nodes negotiate the allocation of the error budget in a localized level-by-level manner, instead of having the whole process initiated by the *Root* node. In particular, each non-leaf node in the tree claims an available error budget equal to:

$$E\_Additional_i = \sum_{N_j \in children(N_i)} E_j \times (1 - shrink) \tag{5.4}$$

This is exactly the available error budget due to the shrinkage of the filters of its immediate descendants. The allocation of this budget among itself and its children nodes in the tree is performed using the potential gain of the node and the gains of its subtrees:

$$E\_Additional_j = \frac{E\_Additional_i \times Gain_j}{Gain_i + \sum_{N_j \in children(N_i)} Gain_j} \tag{5.5}$$

One way to visualize the differences of the two algorithms is to consider how the error budget is being distributed. In the *AdjRoot* algorithm, the whole budget is claimed by the *Root* node. This is possible because all nodes shrink their filters by the same percentage. Then, this error budget is let to flow downwards through the tree, using the accumulated statistics (gains) on the nodes. This process continues until either we reach a leaf node, or when the remaining budget is too small. In the later case the node in consideration claims all the remaining error budget, thus saving downward messages on the corresponding subtree. In contrast, the *AdjLocal* algorithm adjusts the filters in a localized fashion. Any intermediate node in the aggregation tree uses information on the filter widths of its direct descendant nodes to determine its available error budget and then distributes this budget among them and the node itself, without recursively continuing this process on lower levels of the tree.

When comparing the *AdjRoot* and the *AdjLocal* algorithms, one would expect in most cases the *AdjRoot* algorithm to perform better, as it allows broader redistribution of the available error budget. For instance, the *AdjLocal* algorithm will require

more rounds (update periods) to shift a significant amount of error from a subtrees S_1 rooted at a node close to the *Root* to a sibling subtree S_2, because the error-budget will first have to gradually *ascend* towards the root node of the S_1 subtree and then slowly be distributed to the nodes in the S_2 subtree. In *AdjLocal*, whenever some nodes allocate a significant amount of their error budget to themselves, then this results in an increased error budget for the parents of these nodes in the next update period. Using this process, the error of an entire subtree can gradually ascend to (and therefore be distributed by) nodes in higher levels of the aggregation tree.

However, there are occasions when we expect the *AdjLocal* algorithm to be superior. In particular, consider the case when the *Root* node is physically located very far from the nodes that actually collect measurements and that the aggregation tree is tall and narrow in its upper levels. This is a realistic scenario when the aggregate query involves the values observed in just one area of the network. In some extreme cases, the *Root* will be connected to the active nodes through a string of nodes. When the *Root* is several links away from the leaf nodes, the *AdjRoot* algorithm requires a lot of messages to propagate the error budget to the nodes that actually need it. In such cases, the *AdjLocal* algorithm might require fewer messages, since the redistribution process will mostly involve active nodes at (or near) the leaves of the tree. Moreover, due to the minimum additional error that can be distributed to subtrees by the *AdjRoot* algorithm, nodes with modest gains may not receive any budget if they belong to subtrees with small cumulative gains. However, in the *AdjLocal* algorithm, through a local redistribution of errors from their siblings and their parent, these nodes will still be able to increase their filters and, thus, reduce

the number of their transmitted messages.

5.6 PGA Algorithm Experiments

5.6.1 Description of Algorithms

We have developed a simulator for sensor networks that allows us to vary several parameters like the number and configuration of the nodes, the topology of the aggregation tree, the data distribution etc. The synchronization of the sensor nodes is performed as described in TAG [MFHH02]. TAG allows us to reduce the number of messages by combining, whenever possible, messages, on a path to the root within the same epoch. All precision control algorithms are implemented on top of this protocol. In our experiments we compare the following algorithms:

1. *BBA* (Burden-Based Adjustment): This is an implementation of the algorithm presented in [OJW03] for the adaptive precision setting of cached approximate values. As noted above, we use TAG to aggregate messages in the same epoch to further reduce the overall bandwidth consumption.

2. *Uni*: This is a static setting where the error is evenly distributed among all active sensor nodes, and therefore does not incur any communication overhead for adjusting the error thresholds of the nodes.

3. *PGA* (Potential Gains Adjustment): This is our precision control algorithm, based on the potential gains as described in section 5.5. For adjusting the

filters of the sensor nodes we use the *AdjRoot* algorithm; later in this section we also provide an experimental evaluation with the *AdjLocal* method as well.

For the BBA algorithm, we experimented with several heuristics for estimating the cost C_i of each message transmitted by a node N_i, and set it in our experiments to $\frac{dist_i+1}{2}$, where $dist_i$ denotes the distance in number of hops of the node from the *Root* node. Our heuristic is the average of the worst case cost (message not aggregated with any other message until it reaches the *Root*) and the best case cost (message aggregated with others at the parent node of N_i) of messages transmitted by node N_i, and provided the best results in most cases. With this heuristic, each node is able to estimate its burden score and potentially transmit it to the *Root* node at the last epoch of the update period. It is important to emphasize that in our implementation of BBA, we do not account for the additional amount of information needed for the nodes to transmit their burden scores (we do not count the messages needed to transmit them). This is an ideal scenario for BBA and is used to provide a more direct comparison to the PGA algorithm, as to the amount of messages pruned by each method due to the installation of the filters.

For the PGA and BBA algorithms we made a few preliminary runs to choose their internal parameters (*adjustment period*, *shrink percentage*). Notice that the *adjustment period* determines how frequently the precision control algorithm is invoked, while the *shrink percentage* determines how much of the overall error budget is being redistributed. Based on the observed behavior of the algorithms, we have selected the combination of values of Table 5.6 as the most representative ones for reveal-

	Configuration	
Parameters	Conf1	Conf2
Upd	50	20
shrink	0.6	0.95
Invocations	Fewer	Frequent
Error Amount Redistributed	Significant	Smaller

Table 5.6: Used Configurations

ing the "preferences" of each algorithm. The first configuration (Conf1) consistently produced good results, in a variety of tree topologies and data sets, for the PGA algorithm, while the second configuration (Conf2) was typically the best choice for the BBA algorithm. In the BBA algorithm we also determined experimentally that distributing the available error to 10% of the nodes with the highest burden scores was the best choice for the algorithm.

The initial allocation of error thresholds was done using the uniform policy. We then used the first 10% of epochs as a warm-up period for the algorithms to adjust their thresholds and report the number of transmitted messages for the later 90%.

5.6.2 Description of Data Sets

Synthetic Data Sets: We generated synthetic data, similar in spirit to the data used in [OJW03]. For each simulated active node, we generated values following a random walk pattern, each with a randomly assigned step size in the range $(0 \ldots 2]$. We further added in the mix a set of "unstable nodes" whose step size is much larger: $(0 \ldots 200]$. These volatile nodes allow us to investigate how the different algorithms adapt to noisy sensors. Ideally, when the step-size of a node is comparable to the

global error threshold, we would like the precision control algorithm to restrain from giving any of the available budget to that node at the expense of all the other sensor nodes in the tree. We denote with $P_{volatile}$ the probability of an active node being volatile.

$P_{volatile}$ describes the volatility of a node in terms of the magnitude of its data values. Volatility can also be expressed in the orthogonal temporal dimension. For instance some nodes may not update their values frequently, while others might be changing quite often (even by small amounts, depending on their step size). To capture this scenario, we further divide the sensor nodes in two additional classes: *workaholics* and *regulars*. Regular sensors make a random step with a fixed probability of 1% during an epoch.[8] Workaholics, on the other hand, make a random step on every epoch. We denote with $P_{workaholic}$ the probability of an active node being workaholic.

Real Data Sets: We also report results using two real data sets. The first, denoted as `LBL-TCP-3`, is described in [PF95] and was also used in the original paper of [OJW03]. It contains information on all the wide-area TCP traffic between the Lawrence Berkeley Laboratory and the rest of the world for a period of two hours. We have processed this data and created individual time-series (one per sensor node) for each of the 1,540 source IP addresses in the trace. Each time-series describes the number of bytes transmitted from a source IP per second.

The second real data set, denoted as `Weather`, was obtained from IRI/LDEO Climate Data Library and consists of precipitation data from 1,582 weather stations.[9]

[8] We have experimented with many different mixes of configurations but, for brevity, we here present the most characteristic cases.

[9] Data set at http://ingrid.ldeo.columbia.edu/SOURCES/.NOAA/.NCDC/.DAILY/.FSOD/

Again, we created individual time-series (one per sensor node) using precipitation measurements from each weather station. Sensor networks that are used in environmental monitoring are expected to process similar data.

5.6.3 Network Topology

We used three different network topologies denoted as T_{leaves}, T_{all} and T_{random}. In T_{leaves} the aggregation tree was a balanced tree with 5 levels and a fan-out of 4 (341 nodes overall). For this configuration all active nodes were at the leaves of the tree. In T_{all}, for the same tree topology, all nodes (including the *Root*) were active. Finally in T_{random} we used 500 sensor nodes, forming a random tree each time. The maximum fan-out of a node was in that case 8 and the maximum depth of the tree 6.[10] Intermediate nodes in T_{random} were active with probability 20% (all leave nodes are active by default).

In all experiments, we executed the simulator 10 times and present here the averages. In all runs we used the SUM aggregate function (the performance of AVG was similar).

5.6.4 Benefits of Residual Mode of Operation

The three precision control algorithms considered (Uni, PGA, BBA) along with the mode of operation (residual: Res, non-residual: $NoRes$) provide us with six different choices (Uni+Res, Uni+$NoRes$, ...). We note that BBA+$NoRes$ is the original

[10] This configuration resembles the placement of nodes in a 2 dimensional grid, where each node can select its parent from up to 8 different choices.

	T_{leaves}	T_{all}	T_{random}
$PGA{+}Res$	**423** / 978	**479** / 903	**677** / 1,207
$PGA{+}NoRes$	463 / 924	558 / 894	830 / 1,454
$BBA{+}Res$	2,744 / 1,654	2,471 / **1,426**	3,775 / 2,657
$BBA{+}NoRes$	3,203 / **1,394**	2,967 / 1,481	4,229 / **2,474**
$Uni{+}Res$	**2,568**	**2,451**	**3,906**
$Uni{+}NoRes$	**2,568**	2,642	4,044
$(E\_Global{=}0){+}Res$	**4,176**	**4,176**	**5,142**

Table 5.7: First number is total number of messages (in thousands) in the network when using parameters of Conf1, second for Conf2 (see also Table 5.6). Uni does not use these parameters. Best numbers for each algorithm in bold.

algorithm of [OJW03] running over TAG, while $BBA{+}Res$ is our extension of that algorithm using the residual mode of operation. The combination $PGA{+}Res$ denotes our algorithm. We first investigate whether the precision control algorithms benefit from the use of the residual mode of operation. We also seek their preferences in terms of the values of parameters *adjustment period* and *shrink percentage*.

We used a synthetic data set with $P_{volatile}$=0 and $P_{workaholic}$=0.2. We then let the sensors operate for 40,000 epochs using a fixed error constraint E_Global=500. The average value of the SUM aggregate was 25,600, meaning that this E_Global value corresponds to a relative error of about 2%. In Table 5.7 we show the total number of messages in the sensor network for each choice of algorithm and tree topology and each selection of parameters. We also show the number of messages for an exact computation of the SUM aggregate using one more method, entitled as $(E\_Global{=}0){+}Res$, which places a zero width filter in every node and uses our residual mode of operation for propagating changes. Effectively, a node sends a message to its parent only when the partial aggregate on its subtree changes. This is nothing more than a slightly

enhanced version of TAG. The following observations are made:

- Using a modest *E_Global* value of 500 (2% relative error), we reduce the number of messages by 7.6-9.9 times (in PGA+Res) compared to (*E_Global*=0)+Res. Thus, error-tolerate applications can significantly reduce the number of messages in the network resulting in great savings on both bandwidth and energy consumption.
- Algorithm PGA seems to require fewer invocations (larger *adjustment period*) but with a larger percentage of the error to be redistributed (a smaller *shrink percentage* results in a wider reorganization of the error thresholds). In the table we see that the number of messages for the selection of values of Conf1 is always smaller. Intuitively, larger adjustment periods allow for more reliable statistics on the computation of potential gains.
- On the contrary, BBA seems to behave better when filters are adjusted more often by small increments. We also note that BBA results in a lot more messages than PGA, no matter which configuration is used.
- The PGA algorithm, when using the residual operation (PGA+Res), results in substantially fewer messages than all the other alternatives. Even when using the non-residual mode of operation, PGA outperforms, significantly, the competitive algorithms.
- BBA seems to benefit only occasionally from the use of the residual operation. The adjustment of thresholds based on the burden of a node cannot distinguish on the true cause of a transmission (change on local measurement or change in the subtree) and does not seem to provide a good method of adjusting the filters with

respect to the tree hierarchy.

In the rest of the section we investigate in more details the performance of the algorithms based on the network topology and the data distribution. For PGA we used the residual mode of operation. For BBA we tested both the residual and non-residual modes and present the best results for each experiment.[11] We configured PGA using the values of Conf1 and BBA using the values of Conf2 that provided the best results per case.

5.6.5 Sensitivity on Temporal Volatility of Sensor Measurements

We here investigate the performance of the algorithms when varying $P_{workaholic}$ and for $P_{volatile}$=0. We first fixed $P_{workaholic}$ to be 20%, as in the previous experiment. In Figure 5.6 we plot the total number of messages in the network (y-axis) for 40,000 epochs when varying the error constraint *E_Global* from 100 to 2,000 (8% is terms of relative error). Depending on *E_Global*, PGA results in up to 4.8 times fewer messages than BBA and up 6.4 times fewer than Uni. These differences arise from the ability of PGA to place, judiciously, filters on passive intermediate sensor nodes and exploit negative correlations on their subtree based on the computed potential gains. Algorithm BBA may also place filters on the intermediate nodes (when the residual mode is used) but the selection of the widths of the filters based on the burden scores of the nodes was typically not especially successful in our experiments.

Figures 5.7 and 5.8 repeat the experiment for the T_{all} and T_{random} configurations. For the same global error threshold, PGA results in up to 4 times and 6 times fewer

[11] As seen on Table 5.7, the differences were very small.

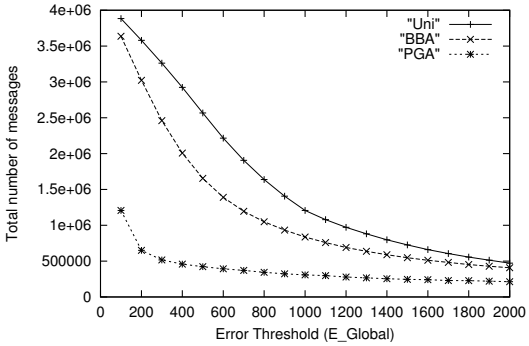


Figure 5.6: Messages varying *E_Global* for T_{leaves} configuration

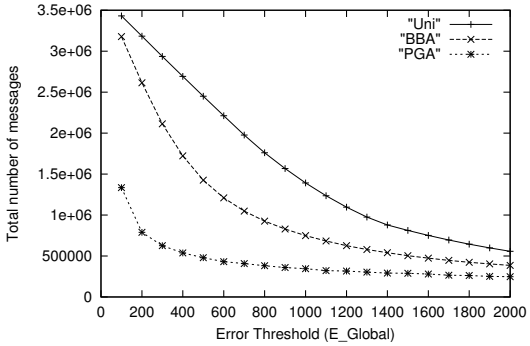


Figure 5.7: Messages varying *E_Global* for T_{all} configuration

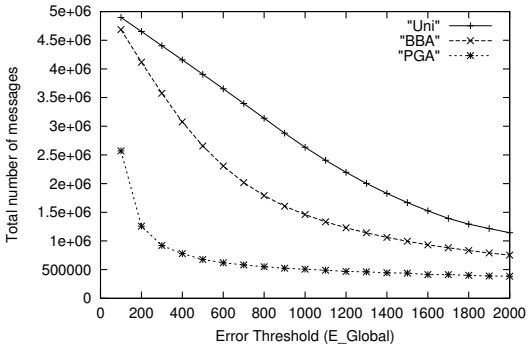


Figure 5.8: Messages varying *E_Global* for T_{random} configuration

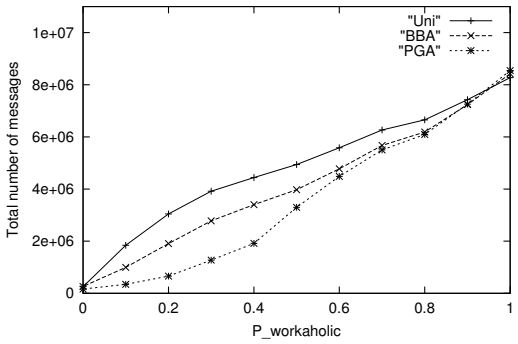


Figure 5.9: Messages varying $P_{workaholic}$ for T_{all} configuration

messages than BBA and Uni respectively. In Figure 5.9 we vary $P_{workaholic}$ between 0 and 1 for T_{all} (best network topology for BBA) and for *E_Global*=500. Again PGA outperforms the other algorithms. An important observation is that when the value of $P_{workaholic}$ is either 0 or 1, all the methods behave similarly. In this case all the nodes in the network have the same characteristics, so it is not surprising that Uni performs so well. The PGA and BBA algorithms managed to filter just a few more messages than Uni for these cases, but due to their overhead for updating the error thresholds of the nodes, the overall number of transmitted messages was about the

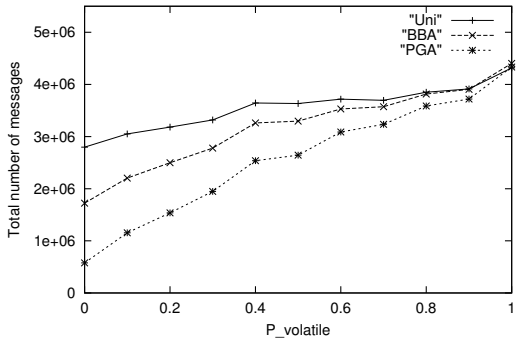


Figure 5.10: Messages varying $P_{volatile}$ for T_{all} configuration

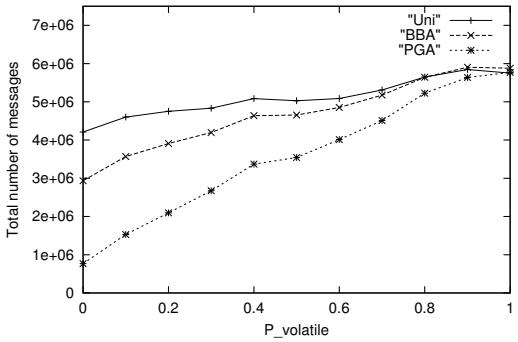


Figure 5.11: Messages varying $P_{volatile}$ for T_{random} configuration

same for all techniques.

5.6.6 Sensitivity in Magnitude of Sensor Measurements

In Figures 5.6.6, 5.11 we vary the percentage of volatile nodes (nodes that make very large steps) from 0 to 100% and plot the total number of messages for T_{all} and T_{random} ($P_{workaholic}$=0.2, E_Global=500). For $P_{volatile}$=1 the error threshold (500) is too small to have an effect on the number of messages and all algorithms have practically the same behavior. For smaller values of $P_{volatile}$, algorithm PGA results in a reduction in the total number of messages by a factor of up to 3.8 and 5.5 compared to BBA and Uni respectively.

5.6.7 Comparison of the *AdjRoot* and *AdjLocal* Algorithms

We started with a balanced aggregation tree with a fan-out of 4 and 6 levels (1,365 nodes overall) having all active nodes at the leaves of the tree (i.e., similar to T_{leaves}). We then gradually augmented the tree by injecting *transport* nodes between levels

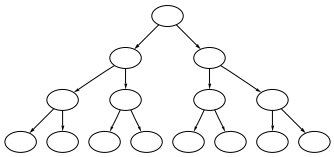


Figure 5.12: Original aggregation tree

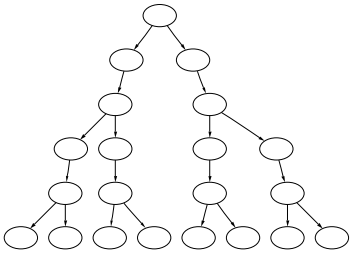


Figure 5.13: After adding transport nodes between layers 0-1 and 1-2

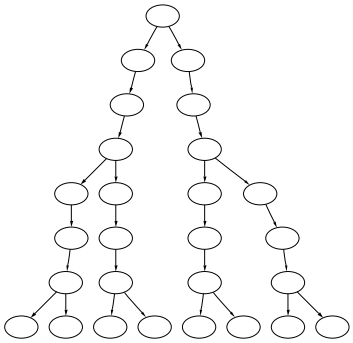


Figure 5.14: After adding second set of transport nodes between layers 0-1 and 1-2

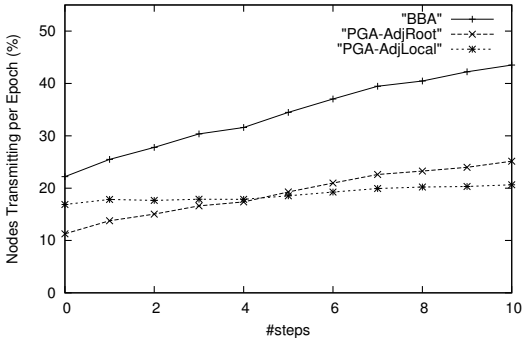


Figure 5.15: Algorithm performance varying the number of transport layers

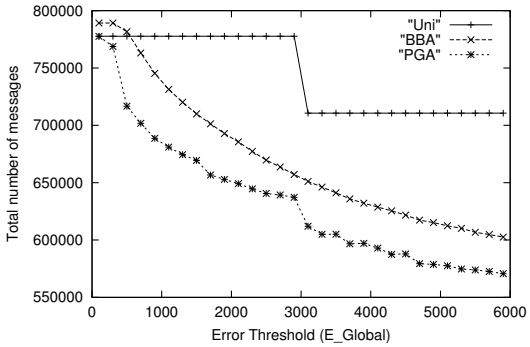


Figure 5.16: Messages, `LBL-TCP-3` data set

0-1, 1-2 and 2-3 in the tree. This process is illustrated in Figures 5.12, 5.13 and 5.14. For presentation purposes in these figures we use an initial tree with fan-out 2 and only 4 levels. In Figure 5.13 we show the resulting tree of adding transport nodes between levels 0-1 and 1-2, while Figure 5.14 shows the tree after adding another set of transport nodes between these levels. Essentially each step makes the top-level nodes of the tree lay further away for the leaf nodes that collect the measurements.

In Figure 5.15 we compare the performance of the BBA algorithm against PGA

(residual mode) with the later using (i) the *AdjRoot* algorithm for adjusting the filters (the default choice) and (ii) the *AdjLocal* algorithm. The y-axis is the figure shows the percentage of nodes in the tree transmitting on an epoch, averaged over 1,000 epochs and 10 repetitions of the experiment (E_Global=1,500). The x-axis shows the number of successive steps of adding transport nodes. As more nodes are added between the *Root* and the leaves of the tree, the number of messages increases in both *BBA* and *PGA+AdjRoot* algorithms. This is due to both the increased number of nodes in the tree and because both algorithms adjust the filters in a top-down manner, thus resulting in a larger reorganization overhead, since the average distance of the *Root* node from the nodes of the aggregation tree that ultimately received most of the error budget increases. In contrast, when using the *AdjLocal* algorithm for adjusting the filters, the performance is practically unaffected by the addition of the transport nodes. We note that both *PGA+AdjLocal* and *PGA+AdjRoot* operate on the same set of statistics collected at the nodes and, in principle, one can alternate between the two algorithms; i.e., use *PGA+AdjLocal* when the data distribution appears to be relatively static and switch to *PGA+AdjRoot* when a quick large-scale redistribution of the budget is required.

5.6.8 Experiments with Real Data

In Figure 5.16 we summarize our results for the `LBL-TCP-3` data set and the T_{random} topology. This data set has the unique feature that many IP-sources show long periods of inactivity (number of bytes sent is zero) followed by short, bursty transmissions.

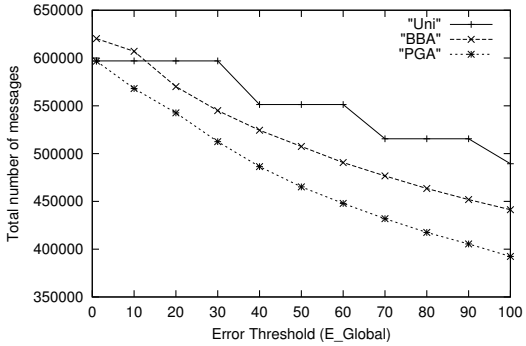


Figure 5.17: Messages, `Weather` data set, T_{all} configuration

We include this data, as a "hard" case for our algorithm, since this property makes it hard to predict future behavior based on past statistics. However, we can see that *PGA* still outperforms the other alternatives. We also note that, for very small values of *E_Global*, *Uni* is very competitive, as in that case the available thresholds are not enough to prune transmissions of active IP sources. We remind that *Uni* has a static allocation of filters, and has no overhead of adjusting them, unlike the other two algorithms.

We also provide results using precipitation readings from the `Weather` data set. In Figure 5.17 we show the total number of messages, varying *E_Global*, for the three algorithms, when nodes are organized in the T_{all} configuration.

5.7 Our MGA Algorithm

In this section we first describe the notion of a *marginal gain* and how it can be calculated at each node. We then provide details on which statistics need to be maintained at each node by our MGA algorithm, how each node calculates the cumulative bandwidth consumption within its subtree, and how our MGA algorithm dynami-

cally adjusts the filter widths of the sensor nodes. Some basic information on the node operation and the use of the error filters was presented in Section 5.5.1. The notation used throughout this section is consistent with the notation introduced in Table 5.2. Additional notation is introduced in Table 5.8. A detailed description of these symbols is presented in appropriate parts of this section. We finally discuss some interesting extensions to our algorithm, including modifications when node movement occurs or when local bandwidth constraints exist.

5.7.1 Algorithm Description

Intuition of our Algorithm

Assume a simple schedule in which the *Root* node decides how to adjust the nodes' error filters every Upd epochs. Consider a snapshot of the network at the epoch when the *Root* node makes such a decision. Let B_{actual} denote the overall bandwidth consumption since the last update epoch, and $B\_Global \times Upd = B\_Util \times \mathrm{T} \times Upd$ denote the targeted bandwidth consumption. If the network bandwidth is underutilized ($B_{actual} < B\_Global \times Upd$), the *Root* node may instruct some nodes to increase their bandwidth consumption by decreasing their error thresholds. This necessitates the

Symbol	**Description**
B_Cum	Total bandwidth consumption in node's subtree
DE	Difference of filter widths between the two anchor points used when calculating the node's marginal gain
DB	Expected decrease in bandwidth when increasing filter width by DE
$CumDE$	Sum of DE values among all nodes with $DB \neq 0$ in subtree
$CumDB$	Sum of DB values among all nodes in subtree
$budget$	Bandwidth (positive or negative) assigned to the node's subtree in the update process

Table 5.8: Notation Used in the MGA Algorithm

existence of a method to translate the additional bandwidth units in each node to changes in the node's error filter. An important question that is also being raised is which nodes should receive the most additional bandwidth. Since we want to provide tight error guarantees, it is evident that nodes which are expected to exhibit the largest reduction in their error filters per additional bandwidth unit should be ordered to increase their bandwidth consumption by the largest amount. Inversely, if the bandwidth is overutilized, then some nodes will be ordered to decrease their bandwidth utilization. In this case, the nodes which will exhibit the smallest increase in their error filters per reduced bandwidth unit should be ordered to have the largest decrease in their bandwidth consumption.

Marginal Gains

In Figure 5.18 we depict the expected width of a sensor node's error filter as we vary the desired number of transmitted messages by the node to its parent in the aggregation tree, within a period of Upd epochs. This is similar to Figure 5.5, but augmented with more information needed in our algorithm. The maximum number of transmitted messages within an update period is obviously equal to Upd, and this may occur, for example, when the filter has zero width and the partial aggregate value calculated by the node changes at each epoch. As the desired bandwidth consumption increases, the width of the filter that is expected to result in this bandwidth consumption gradually decreases.

Now, consider a randomly chosen node N_i in the aggregation tree and let W_i denote the node's error filter width. At each epoch, N_i decides whether to make a

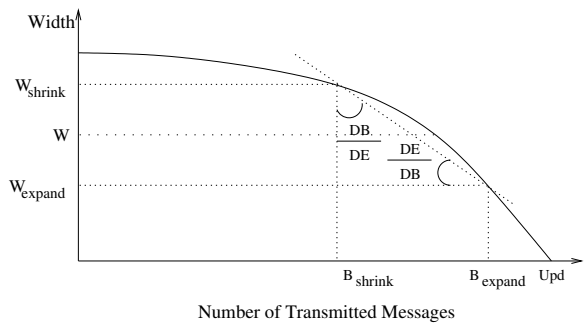


Figure 5.18: Marginal Gains of a Node

transmission, based on the value of the current partial aggregate calculated at N_i and its deviation from the previously transmitted aggregate value. At the same time, the node also keeps track of the number of transmissions it would have performed had its filter width been either a smaller W_{shrink} or a larger W_{expand} value. We refer to these filter width values as the two *anchor* points, and defer the discussion on how to determine their values for Section 5.7.2. Let B_{shrink} and B_{expand} denote the calculated number of transmissions in each case, correspondingly, and also let $DB = B_{shrink} - B_{expand}$ and $DE = W_{expand} - W_{shrink}$. Then, the ratio $\frac{DE}{DB}$ is an indication of the decrease (increase) on the node's maximum error per additional (reduced) bandwidth unit assigned to the node.

Statistics Maintained at each Node

Besides the values of DB and DE that we described above, each node also needs to maintain some additional statistics. We denote as $CumDE_i$ the sum of the DE values among all nodes in N_i's subtree that have nonzero DB values; that is:

$$CumDE_i = \sum_{\substack{j\,:\,N_j \in subtree(N_i) \\ and\ DB_j > 0}} DE_j = \begin{cases} DE_i + \sum_{j\,:\,N_j \in children(N_i)} CumDE_j & DB_i > 0 \\ \sum_{j\,:\,N_j \in children(N_i)} CumDE_j & DB_i = 0 \end{cases}$$

The reason why we exclude from the calculation of the $CumDE$ values those nodes which have zero DB values will be made clear later in this section. We also denote as $CumDB_i$ the corresponding sum of the DB quantities among all nodes in N_i's subtree. Each node can, therefore, perform the calculation of the $CumDB_i$ and $CumDE_i$ values using local statistics that its children nodes can piggyback to messages transmitted by them. This is a minimal amount of information needed that is aggregated at each node. These two quantities will be used, as we will explain later, by our algorithm to dynamically adjust the widths of the error filters. We here note that each node should not discard the latest individual $CumDB$ and $CumDE$ statistics transmitted by its children, as these quantities will also be used by our MGA algorithm in the allocation of bandwidth among the nodes in the aggregation tree.

Computing the Bandwidth Consumption

Because of the hierarchical topology and the limited transmission ranges of nodes, the *Root* node has no way of determining by itself the actual bandwidth consumption in the entire network. To resolve this, we use a simple intuitive idea. Each node maintains an estimate B_Cum of the overall bandwidth consumed by nodes in its

subtree (including the node itself) during the last update period. When a node transmits a message to its parent node, it increments its calculated B_Cum value for its subtree by one (to account for the new message) and piggybacks this estimate in its message.[12] The parent in turn uses the last received bandwidth estimates from its children to calculate its own B_Cum value. Think of these values as "bubbles" that ascend the hierarchy when nodes transmit. The *Root* node sums-up all the values it receives from its children.

A small complication arises because some nodes in the middle of the hierarchy, due to their error filters, may have pruned messages. Thus, some statistics on the bandwidth consumption of their descendant nodes may not have been propagated towards the *Root* node. To solve this problem, at the epoch immediately before the invocation of the algorithm for adjusting the filters (discussed below), the statistics (bubbles) that still remain in the network are transmitted towards the *Root* node. Every node whose last transmitted value of B_Cum differs from the corresponding current value performs a transmission, even if this is not required by its latest measurement, and this process goes on recursively until all bubbles reach the *Root* node. Note that the above procedure may only occur for non-leaf nodes of the aggregation tree.

[12] Actually, when using the cost model used by the LEACH [HCB00] and Pegasis [LR02] protocols, where the energy drain during the transmission or reception of messages is proportional to the number of transmitted/received bits, it is often optimal, in terms of energy consumption, to transmit the B_Cum, $CumDE$ and $CumDB$ statistics only at the last epoch of each update period.

Adjusting the Error Filters

We now present the complete MGA algorithm for dynamically adjusting the error filters installed in sensor nodes.

The algorithm starts at the *Root* node and progressively distributes additional bandwidth (which is positive in case of bandwidth underutilization, or negative in case of bandwidth overutilization) to subtrees and nodes in a top-down fashion. Each node N_i awaits a message containing the additional bandwidth $budget_i$ (positive or negative) to be distributed to the nodes in its subtree. If such a message arrives and $budget_i > 0$, then this budget is distributed among the node itself and the node's subtrees proportionally to the expected per bandwidth unit benefit of each choice, which is in turn equal to $\frac{DE_i}{DB_i}$ for the node itself (if $DB_i > 0$) and $\frac{CumDE_j}{CumDB_j}$ for each child subtree with $CumDB_j > 0$. Subtrees (nodes) having $CumDB = 0$ ($DB = 0$) receive no bandwidth and no message is being sent to them. For the remaining subtrees, their budget is calculated as (assuming $DB > 0$):

$$budget_j = \frac{budget_i \times \frac{CumDE_j}{CumDB_j}}{\frac{DE_i}{DB_i} + \sum_{\substack{N_k \in children(N_i) \\ and\ CumDB_k > 0}} \frac{CumDE_k}{CumDB_k}} \tag{5.6}$$

and a message is transmitted to them with this value. The formula for calculating the budget allocated for the filter of node N_i itself is similar, but instead uses the quantity $\frac{DE_i}{DB_i}$ on the nominator. The budget allocated to the node is then multiplied by $\frac{DE_i}{DB_i}$ to determine the appropriate increase in the error filter's width. If the budget given to some subtree is very small (for example, less than 1) then there is no real

benefit in such a budget assignment, since the update message itself will outweigh any possible benefits of error filter adjustments in the subtree. These small bandwidth budgets can be redistributed to subtrees which are programmed to receive additional bandwidth.

We now consider the case when either some node does not receive any update message, or has $DB = 0$. If $DB = 0$, then the node can decrease its error filter's width to W_{shrink} without this having an impact on the expected bandwidth consumed by the node. If $DB > 0$ and no bandwidth is allocated to the node, then the node does not modify its error filter.

Finally, the case when the bandwidth is overutilized (and therefore the disseminated budget is negative) is almost symmetric to our above discussion. Based on our discussion at the beginning of this section, the nodes which are expected to exhibit the smallest increase in their error filters per reduced bandwidth unit should be ordered to have the largest decrease in their bandwidth consumption. Therefore, the distribution of the negative budget should be performed proportionally to the $\frac{DB}{DE}$ quantity in this case. For any node N_i, the equation of allocating bandwidth to its children subtrees therefore becomes:

$$budget_j = \frac{budget_i \times \frac{CumDB_j}{CumDE_j}}{\frac{DB_i}{DE_i} + \sum_{N_k \in children(N_i)} \frac{CumDB_k}{CumDE_k}} \tag{5.7}$$

Finally, the case when $DB = 0$ is handled almost identically to the case of bandwidth underutilization (a small difference is discussed in Section 5.7.2). Therefore, independently of whether the dispensed budget is positive or negative, the behavior

Node	W	DB	CumDB	DE	CumDE
1	0	0	260	0	8
2	2.5	90	120	2	4
3	1.25	0	180	1	**4**
4	2.5	30	30	2	2
5	5	0	0	4	**0**
6	2.5	60	60	2	2
7	1.25	50	50	1	1
8	1.25	70	70	1	1

Table 5.9: Sample Statistics

of nodes having $DB = 0$ remains the same. This is the reason why they are not taken into account when calculating the $CumDE$ values at each node. It is interesting to note that in the case of bandwidth overutilization, the width of a node's error filter may either increase (if negative budget is assigned to the node), remain intact (if it receives zero budget), or even decrease (if $DB = 0$).

It is important to emphasize that in either case (positive or negative bandwidth budget), each node N_i in the aggregation tree decides how to distribute its budget to the node itself and to its children subtrees by using only statistics received by its children nodes. The tree topology is taken into account when calculating the cumulative statistics ($CumDE$, $CumDB$ and B_Cum) of each subtree. Finally, we need to note that in the first epoch after the reorganization, each node needs to transmit the new error of its entire subtree (calculated bottom-up) so that the *Root* node will be able to know the error guarantees of its estimated aggregate value.

Example 3: We now present a simple example to demonstrate the error filter adjustment process. Consider the aggregation tree of Figure 5.3 and assume that the budget of node 1 is 30 (the bandwidth was therefore underutilized in our example

in the previous update period) and that the W, DE, DB, $CumDE$ and $CumDB$ values of each node are the ones presented in Table 5.9. Note that in two cases where $DB = 0$, the $CumDE$ values (marked in bold) have omitted from their calculations the DE values of some nodes. At the beginning, node 1 disseminates its budget based on the values $\frac{CumDE_2}{CumDB_2}$ and $\frac{CumDE_3}{CumDB_3}$. Using Equation 5.6, the node dispenses budget 18 and 12 to nodes 2 and 3, respectively. The budget left for node 1 is 0, because there is no point in using a non-zero filter at the *Root* node. Now, considering just the case of node 2, the node will dispense budget 13.5 to node 4, 0 budget to node 5 and keep the remaining budget (4.5) for itself. Node 5 has $DB_5 = 0$ and, therefore, decreases its error filter to W_{shrink_5}. Node 4 will decrease its error filter by $0.9 = 13.5 \times \frac{2}{30}$ and set its new width to 1.6. Similarly, node 2 will decrease its filter by $0.1 = 4.5 \times \frac{2}{90}$. ∎

5.7.2 Algorithm Details

We now present some details of our algorithm that are either not covered by the above discussion, or have been omitted to this point for ease of presentation.

Selecting the Anchor Points

The selection of the two anchor points W_{shrink} and W_{expand} should enable the sensor nodes to calculate useful statistics on the expected impact on their error filters by a desired increase or decrease of their bandwidth consumption. While there is no optimal way to select these two anchor points, there are some useful guidelines that help determine their values. A simple technique would be to select filters with widths smaller and larger than W_i by a factor of a $(0 < a < 1)$, i.e. setting $W_{shrink} = (1-a)W_i$

and $W_{expand} = (1+a)W_i$. However, using this setting, for nodes with small error filters the distance of the two anchor points will be small. This prevents the collection of useful statistics and often results in estimated DB values of zero. A more robust approach would utilize the standard deviation σ (or equivalently the variance σ^2) of the measurements collected at the node during the *previous* update period[13] thus setting $W_{shrink} = \max\{W_i - \sigma, 0\}$ and $W_{expand} = W_i + \sigma$. This technique, however, is slow to react to nodes that exhibited a small variance in their measurements during their previous update period, thus resulting in a small distance of the two anchor points in the current update period, and which suddenly become more volatile in their measurements. What is, thus, needed is a combination of the aforementioned techniques.

In our MGA algorithm we use the following values for the two anchor points: $W_{shrink} = \max\{0, \min\{W_i - \sigma, (1-a)W_i\}\}$ and $W_{expand} = \max\{W_i + \sigma,\ (1+a)W_i,\ W_{shrink} + minDE\}$. The $minDE$ value specifies a minimum distance of the two anchor points and is needed in the case of nodes with both small filters and small variance in their measurements. Consider for example the case where the readings in a sensor node are integer values. Then, if the node's filter is centered around an integer value, any distance of the two anchor points smaller than 2 will always result in $DB = 0$. Thus, the $minDE$ value can be determined by the granularity of the node observations.

[13]The standard deviation at the current update period cannot be used, since this would result in moving anchor points, based on the observations during each, current, epoch.

Filter Modification Based on Assigned Budget

Let $DE_{shrink} = W - W_{shrink}$ and $DE_{expand} = W_{expand} - W$. Also, let DB_{shrink} (DB_{expand}) denote the expected increase (decrease) in the number of messages transmitted by the node by decreasing (increasing) its error filter to W_{shrink} (W_{expand}). Obviously, $DB = DB_{shrink} + DB_{expand}$. After determining the budget (positive or negative) assigned to each node during the update process, the desired modification of the error filter is more accurately calculated if, instead of using the node's DE and DB values calculated by both anchor points, the node utilizes just the statistics of the anchor point in the direction of the filter modification. Therefore, when zero budget is assigned to a node, the MGA algorithm shrinks the node's filter to W_{shrink} if $DB_{shrink} = 0$. If negative bandwidth is allocated to the node, then the ratio $\frac{DE_{expand}}{DB_{expand}}$ is used to determine the increase in the filter's width. If positive bandwidth is allocated to the node and $DB_{shrink} > 0$, then the ratio $\frac{DE_{shrink}}{DB_{shrink}}$ is used to determine the decrease in the filter's width. Finally, if positive bandwidth is allocated to the node and $DB_{shrink} = 0$ (but $DB > 0$), then the MGA algorithm shrinks the node's filter to $\min\{W_{shrink}, \max\{W - \frac{|budget| \times DE}{DB}, 0\}\}$. We here need to emphasize that the DE_{shrink}, DE_{expand}, DB_{shrink} and DB_{expand} statistics are not transmitted to other nodes (to limit the size of the transmitted information), and that the bandwidth dissemination is based solely on the DE and DB values, and the corresponding cumulative statistics for the subtree.

Additional Details

The MGA algorithm also imposes a set of restrictions concerning the budget dissemination process.

- There is no point in assigning to a subtree negative budget larger (in absolute value) than the bandwidth B_Cum it consumed during the previous update period. Moreover, we cannot assign to a node itself more budget than what is necessary to drop its error to 0.
- For passive nodes with a single child node, the error filter is always set to zero, since one can easily demonstrate that it is always more beneficial to "push" the error budget of that node to its child.

5.7.3 Extensions

We now describe interesting extensions to our framework, such as dealing with node movement and imposing strict bandwidth constraints in areas of the aggregation tree.

Node Movement

In sensor networks the aggregation tree often changes during the lifetime of a continuous query. This may happen because of node and link failures or, for instance, when nodes are attached to moving objects. When the aggregation tree gets reorganized, each node which experiences changes in the set of its children nodes needs to:

1. Receive the partial aggregate and collected statistics from its new children nodes.

2. Calculate the new partial aggregate and statistics of its subtree by considering the newly acquired values received from its new children nodes and by subtracting the corresponding values of its children nodes that were removed.

3. Make a transmission depending on the value of the new calculated partial aggregate.

Strict Bandwidth Constraints in Local Areas

While our MGA algorithm limits, on the average, the targeted overall bandwidth consumption, it can be adapted to provide a strict bandwidth guarantee in all or parts of the aggregation tree. This will be useful when parts of the network exhibit severe bandwidth or energy constraints. As mentioned in the previous section, each node maintains an estimate of the bandwidth consumption B_Cum in its subtree. If this bandwidth consumption exceeds the given strict constraint for the area, then sufficient negative budget should be assigned to this node's subtree in the next update epoch, independently of the overall bandwidth consumption.

Let B_Lim_i denote the bandwidth constraint in the subtree of node N_i. If we do not wish to impose a bandwidth constraint at certain subtrees, then $B\_Lim_i = \infty$. Each node N_i needs to maintain an additional statistic B_Need_i which denotes the minimum amount, in absolute value, of negative budget needed to be disseminated to descendant nodes of N_i. The value of B_Need_i is calculated as follows:

$$B\_Need_i = \sum_{k:N_k \in children(N_i)} B\_Need_k + \max\{0, B\_Cum_i - \sum_{k:N_k \in children(N_i)} B\_Need_k - B\_Lim_i\}$$

The first summand in the equation above represents the budget needed by nodes in subtrees rooted at children nodes of N_i. The second summand represents the budget needed by node N_i itself and is more complicated. To properly calculate this second summand, we need to take into account not only the bandwidth consumption and the bandwidth constraint in the entire subtree of N_i, but also consider how much of the difference between these two values will be offset by limiting the bandwidth consumed by the subtrees of N_i. The MGA algorithm now requires the following modifications:

- The *Root* node adds (since B_Need represents needed negative budget, in absolute value) its computed B_Need value to the budget that it will disseminate to the sensor nodes. Therefore, the disseminated budget is decided only after the *Root* node takes into account the minimum negative bandwidth needed by areas of the aggregation tree that have exceeded their bandwidth limits.
- Each subtree with a B_Need value greater than zero automatically receives at least this amount of negative budget. We here note that this negative budget will not be assigned to the subtree by its parent node, but is acquired automatically. Additional negative budget may be assigned to this subtree if the budget disseminated by the *Root* node is negative. Positive budget is disseminated only to subtrees that have not exceeded their bandwidth limit.

5.8 MGA Algorithm Experiments

We have developed a simulator for testing the TBA and MGA algorithms that we discussed in this chapter under various conditions. The experiments that we present are split in two parts. We first use synthetic data sets to test the effect of various data characteristics. In the second part we experiment with real sensor data.

5.8.1 Configuration Parameter Selection

We first ran a set of preliminaries experiments for setting up the configuration parameters of the algorithms. Both algorithms had a large range of values for their configuration parameters that provided near-optimal results. As an example we show in Figures 5.19 and 5.20 the average error guarantee when varying the update period (Upd) from 10 to 100 epochs for the light and temperature measurements of the lab data set discussed in Section 5.8.3. For TBA we found out that the parameters that were originally proposed in [OW02] provided the best results in most cases. We thus used θ=1.1 and ω=10. The update period Upd was 50. For MGA we used α=40% and Upd=40.

5.8.2 Sensitivity Analysis

There are two orthogonal dimensions that affect the evaluation of a bandwidth constrained query. The first is the hierarchical organization of the nodes and the second is the data distribution. We have experimented with several topologies for the aggregation tree. For brevity we present results for the following two configurations.

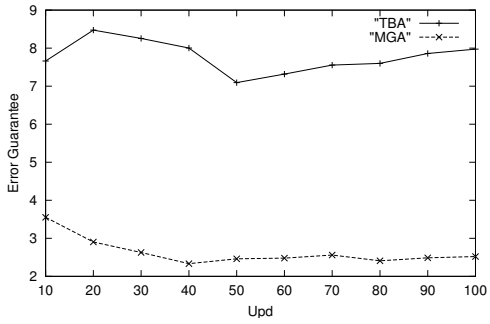


Figure 5.19: Error Guarantee varying Upd, temperature measurements (lab data set)

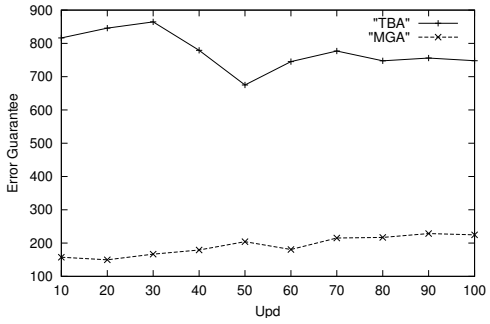


Figure 5.20: Error Guarantee varying Upd, light measurements (lab data set)

- $T1$: In this configuration the sensor nodes form a balanced tree with fanout = 3 and 6 levels (364 nodes overall). Only leaf nodes in the tree are active. Intermediate nodes do not collect measurements but rather aggregate results from their subtrees.
- $T2$: The nodes form a random tree with the fanout of each node being randomly chosen between zero (leaves) and 8 and with the maximum distance of a leaf from the *Root* node equal to 6. The tree is not balanced and leaf nodes are in different distances from the *Root* node. The tree that we used had 644 nodes. Intermediate nodes are active with probability 20%. All leaf nodes are active by default.

The synchronization of the sensor nodes in the tree is performed as described in TAG [MFHH02]. TAG reduces the number of messages by combining, whenever possible, messages on a path to the *Root* node within the same epoch. All algorithms are implemented on top of this protocol.

In the synthetic data sets, we had the values of each active node follow a random walk pattern, similar to the discussion in Section 5.6.2. The probabilies $P_{volatile}$ and $P_{workaholic}$ are defined in a similar way. In the following experiments, unless specified

	MGA			*TBA*		
B_Global	B_used	Error Guarantee	Abs. Error	B_used	Error Guarantee	Abs. Error
20	19.5	3,667.0	279.2	19.0	35,013.7	245.9
30	28.9	2,506.8	195.7	28.5	9,527.5	203.7
40	38.6	1,914.2	155.4	38.1	5,221.6	217.1
50	48.4	993.1	75.7	46.8	4,255.7	218.6
60	58.0	442.2	46.9	55.7	3,439.0	208.1
70	67.5	217.6	21.4	62.2	2,438.8	161.5
80	77.7	121.5	10.9	72.7	1,417.7	101.7
90	87.8	71.5	5.7	77.4	1,463.2	107.2
100	97.6	26.1	1.8	89.9	675.0	54.1

Table 5.10: Avg Error Guarantee, Avg Abs Error and B_used for $T1$

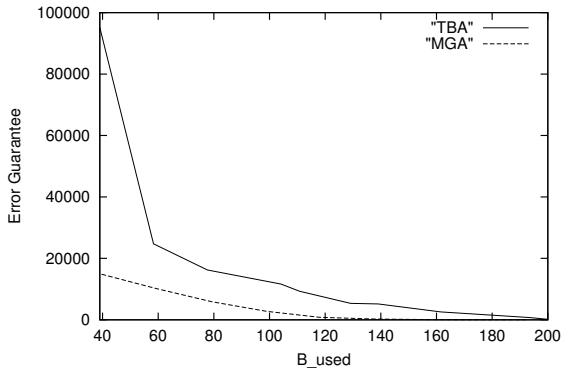


Figure 5.21: Error Guarantee for $T2$

otherwise, we used the SUM function. Performance with the AVG function is similar.

Effect of Bandwidth Constraint. In Table 5.10 we show the average error guarantee and the average absolute error provided by the algorithms over 10,000 epochs for different values of *B_Global* (bandwidth constraint) and for the configuration T1. The corresponding values of *B_Util* can be easily derived in each case by considering the number of nodes in each tested configuration. In this data, we used $P_{volatile} = 0.2$ and $P_{workaholic} = 0.2$.

In this table, the first column shows the constraint used, while the column *B_used* shows the average bandwidth (number of messages per epoch) achieved per

algorithm.[14] The numbers also include any control messages required by the algorithms. We notice that both algorithms achieve a bandwidth consumption very close to the input value. However, there are huge differences in the error guarantee provided per algorithm. The error guarantee of algorithm MGA is up to 26 times smaller than the one of TBA for the same bandwidth constraint. In this table we also show the average absolute deviation of the reported aggregate to the *Root* node from the true aggregate value (corresponding to an unconstrained execution). We notice that both algorithms are significantly more accurate than their reported error guarantee; the (real) absolute error is typically an order of magnitude smaller than the reported error guarantee. This is a trend consistent in all our experiments. For brevity, in the remaining of this section we will only be reporting the error guarantees.

In Figure 5.21 we plot the average error guarantee versus B_Global for $T2$. The error guarantee of MGA, for the same bandwidth constraint is smaller by a factor of up to 75, depending on the used bandwidth constraint.

Overall, the error guarantees provided by the algorithms are very tight. The average value of the SUM aggregate was 254,429 and 531,718 for $T1$ and $T2$ ($T1$ has fewer active nodes than $T2$). For B_Global=20 in $T1$, MGA provides an average error guarantee of 3,667. In relative terms this is just 1.5% of the aggregate value and is obtained with just 19.5 messages per epoch, while an unconstrained execution of the query, as in [MFHH02], would require 363 messages per epoch. Thus, we obtain a 1.5% error guarantee using about 5% of the bandwidth. The actual error is just

[14] As mentioned, TBA does not aggregate the statistics sent to the *Root* node and, thus, the size of each message is substantially larger. We do not account for this overhead of TBA here.

	$T1$		$T2$	
$P_{volatile}$	MGA	TBA	MGA	TBA
0.1	275.1	2,396.9	1,279.5	4,988.6
0.2	442.2	3,439.0	10,174.9	24,729.4
0.3	907.3	4,718.7	16,531.9	43,443.6
0.4	4,036.0	11,051.7	22,526.8	86,970.6
0.5	9,176.5	18,352.0	28,456.8	112,490.0
0.6	10,983.9	21,409.3	41,829.7	259,386.0
0.7	16,950.3	33,328.0	58,972.8	346,293.0
0.8	19,578.4	42,094.5	63,997.1	407,582.0
0.9	27,507.6	64,161.8	79,285.0	591,617.0

Table 5.11: Varying $P_{volatile}$

	$T1$		$T2$	
$P_{workaholic}$	MGA	TBA	MGA	TBA
0.1	413.5	2,873.6	4,162.3	10,428.4
0.2	442.2	3,439.0	10,174.9	24,729.4
0.3	1,774.4	7,239.1	12,487.9	34,428.8
0.4	2,594.7	8,534.7	16,841.0	70,116.4
0.5	4,070.6	10,258.9	18,370.5	91,735.1
0.6	4,220.3	10,947.1	21,356.7	125,393.6
0.7	6,291.4	13,681.1	27,911.5	171,527.2
0.8	8,431.1	16,521.5	25,152.0	172,036.2
0.9	10,281.7	23,102.3	36,684.1	276,168.8

Table 5.12: Varying $P_{workaholic}$

0.1% for the same bandwidth.

Varying Mix of Nodes. In Table 5.11 we vary the probability $P_{volatile}$ that a node is volatile. We set the probability $P_{workaholic}$ to 0.2 and used B_Global=60 in all cases. As the number of volatile nodes increases, the error guarantees increase as well. Clearly, the error guarantee provided by MGA are significantly smaller than the ones provided by TBA.

In Table 5.12 we repeat the experiment varying this time the probability $P_{workaholic}$. In this experiment we set $P_{volatile}$ to 0.2 and B_Global=60.

Measure	Mean	Variance	
		Data Measurements	AVG aggregate
Lab-Temperature (Celsius)	21.81	11.55	1.72
Lab-Light (Lux)	378.61	254,668.00	11,718.55
Lab-Humidity (0-100%)	38.53	40.96	5.63
Weather-Temperature (Fahrenheit)	70.25	225.93	2.54

Table 5.13: Characteristics of Real Data Sets

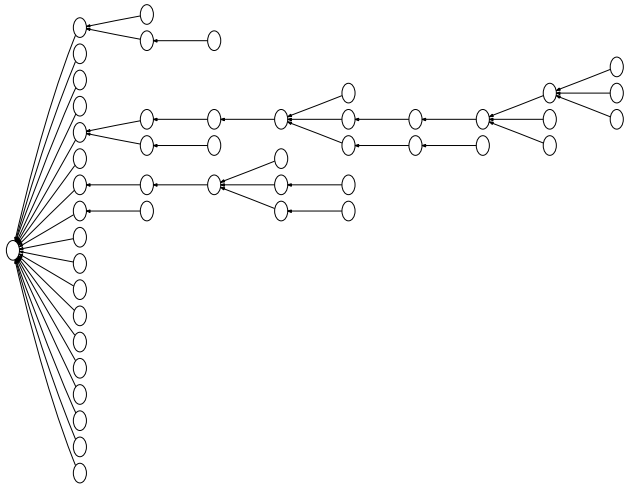


Figure 5.22: Aggregation tree used in *lab* data set

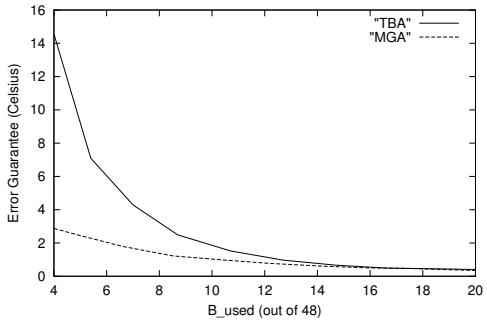


Figure 5.23: Error Guarantee for temperature readings (lab data set)

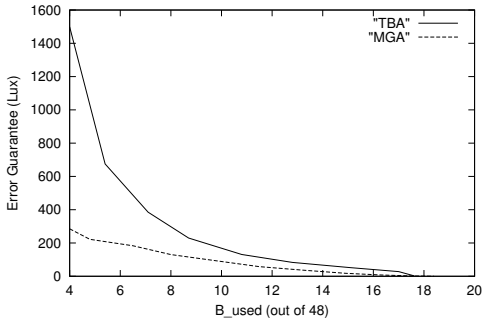


Figure 5.24: Error Guarantee for light readings (lab data set)

5.8.3 Experiments with Real Data

Sensor networks are frequently used in environmental monitoring. We here present results on running experiments over two real-world data sets. The first data set, *lab*,

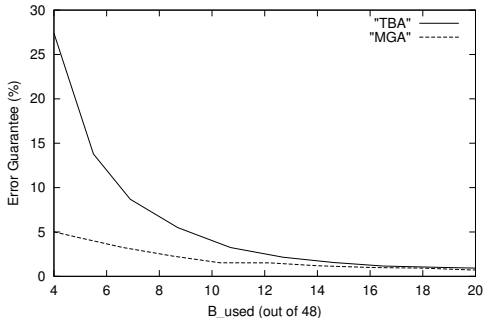


Figure 5.25: Error Guarantee for humidity readings (lab data set)

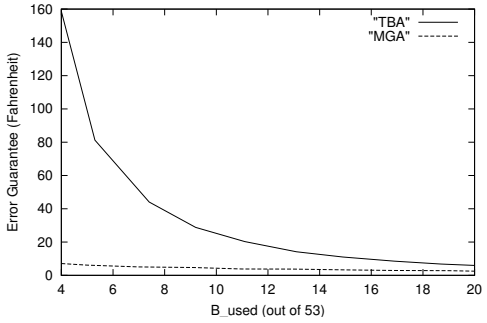


Figure 5.26: Error Guarantee for *weather* data set

is a trace of readings from sensors in the Intel Research, Berkeley lab [DGM+04], collecting light, humidity and temperature readings.[15] We used trace data of 48 sensors for a period of one week. For setting up the aggregation tree, we used the aggregate connectivity data available with this trace. The sensors formed a tree through a simple protocol that prioritized the choice of a parent node based on the quality of the upload link. The final aggregation tree is shown in Figure 5.22.

The second data set, *weather*, provides temperature measurements at a resolution of one minute for the year 2002 (a total of 525K measurements). The data was collected at the weather station of the university of Washington (http://www-k12.atmos.washington.edu/k12). We split the data into 53 non-overlapping sets. We tested this set using a random aggregate tree consisting of 80 nodes. Each intermediate node in the tree had between one and four children while the maximum distance of a leaf from the *Root* node was 5 (i.e. 6 levels). Intermediate nodes were active with probability 20%.

[15]The data set is available at http://berkeley.intel-research.net/labdata/. We would like to thank the owners for making their trace data publicly available.

For the interpretation of the performance of the algorithms we provide in Table 5.13 the mean value and variance of the sensor measurements as well as the variance of the AVG function used in all the aggregate queries in this subsection. In Figures 5.23, 5.24 and 5.25 we show the average error guarantee for the lab data set, while Figure 5.26 depicts the performance for the weather data set. We notice that with about 10% of the readings, MGA provides strong deterministic guarantees that are below the variance of the aggregate in each case.

Chapter 6

Conclusions

In this dissertation we presented several data reduction techniques that can be applied in a variety of applications in constrained environments. We focused our discussion on the area of sensor networks, where the severe energy and bandwidth constraints require the development of novel data reduction techniques for the transmission of measurements collected in these networks.

We first presented a new data compression technique, termed *SBR*, designed for historical data collected in sensor networks, which however can also be applied in compressing multiple time series in general. Our SBR algorithm 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. While the encoding of the data using the base signal is performed using linear regression techniques, our method does not only apply to linear data sets; in fact none of the data that we used in our experimental evaluation are linear in nature. Linearity is exploited when encoding the correlations of the data values and the base signal. The algorithm 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 SBR algorithm 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.

We then developed algorithm that enable the use of wavelet-based algorithms in multi-measure data sets. These techniques can be used either when the processing capabilities of the sensor nodes are prohibitively limited for the application of the SBR algorithm, or when the collected data is multi-dimensional. For this task, we introduced the notion of an *extended* wavelet coefficient as a flexible storage method for maintaining wavelet coefficients for data sets containing multiple measures. This flexible storage method bridges the gap between the two extreme storage hypotheses that the existing algorithms represent, and achieves better storage utilization, which results in improved accuracy to queries. We proposed both optimal and greedy, near-optimal algorithms for selecting which *extended* wavelet coefficients to retain under a storage constraint such that either the weighted sum of the squared L^2 error norms or the maximum relative error is minimized. The results from our extensive experimental study validate the effectiveness of our approach, demonstrating that our greedy GreedyL2 and GreedyRel algorithms constitute highly scalable solutions that provide near optimal results in all cases and achieve significantly more accurate data synopses than those of previously proposed algorithms.

Finally, we proposed a new framework for in-network data aggregation over

sensor networks that supports the evaluation of aggregate queries in error-tolerant applications. Our framework supports a dual scenario, where the application can specify either the maximum error that it is willing to tolerate, or the average bandwidth consumption of the continuous query. Unlike previous approaches, our *PGA* and *MGA* algorithms exploit the tree hierarchy that messages follow in such applications to significantly reduce the number of transmitted messages and, therefore, increase the lifetime of the network. These algorithms are based on two key ideas. Firstly, the residual mode of operation for nodes in the aggregation tree allows nodes to apply their error filters to the partial aggregates of their subtrees and, therefore, potentially suppress messages from being transmitted towards the root node of the tree. A second key idea is the use of simple and local statistics to estimate the gain of allocating additional error to nodes in a subtree. This is a significant improvement over straightforward extensions for the hierarchical setting of previous approaches that require a large amount of information to be transmitted to the root node of the tree. Through an extensive set of experiments, we have shown that while the distribution of the error based on the computed gains is the major factor for the effectiveness of our techniques compared to other approaches, the fusion of the two ideas provides the best improvements.

BIBLIOGRAPHY

[AFF+03] Anastassia Ailamaki, Christos Faloutsos, Paul S. Fischbeck, Mitchell J. Small, and Jeanne VanBriesen. An environmental sensor network to determine drinking water quality and security. *SIGMOD Record*, 32(4):47–52, December 2003.

[AMS96] N. Alon, Y. Mattias, and M. Szegedy. The Space Complexity of Approximating the Frequency Moments. In *Proc. of the 28th Annual ACM Symp. on the Theory of Computing*, 1996.

[ANR74] N. Ahmed, T. Natarajan, and K.R. Rao. Discrete cosine transform. In *IEEE Trans. on Computers, C-23*, 1974.

[BF95] A. Belussi and C. Faloutsos. Estimating the Selectivity of Spatial Queries Using the 'Correlation' Fractal Dimension. In *Proceedings of 21th International Conference on Very Large Data Bases*, pages 299–310, 1995.

[BGM92] D. Barbará and H. Garcia-Molina. The Demarcation Protocol: A Technique for Maintaining Linear Arithmetic Constraints in Distributed Database Systems. In *Proceedings of EDBT*, pages 23–27, 1992.

[BGMGM03] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani. Estimating Aggregates on a Peer-to-Peer Network. Technical report, Stanford, 2003.

[BSG00] T. Barclay, D. Slutz, and J. Gray. Terraserver: A spatial data warehouse, 2000.

[CDH+02] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-Dimensional Regression Analysis of Time-Series Data Streams. In *Proc. of VLDB*, 2002.

[CDTW00] J. Chen, D.J. Dewitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *ACM SIGMOD*, 2000.

[CE02] A. Cerpa and D. Estrin. ASCENT: Adaptive Self-Configuring sEnsor Network Topologies. In *INFOCOM*, 2002.

[CFZ01] M. Cherniack, M. J. Franklin, and S. B. Zdonik. Data Management for Pervasive Computing. In *VLDB*, 2001.

[CGRS00] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate Query Processing Using Wavelets. In *Proc. of the 26th VLDB Conf.*, 2000.

[CGRS01] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *The VLDB Journal*, 10(2-3), September 2001.

[CKP03] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating Probabilistic Queries over Imprecise Data. In *ACM SIGMOD*, 2003.

[CLKB04] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate Aggregation Techniques for Sensor Databases. In *Proceedings of ICDE (to appear)*, 2004.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[CP03] R. Cheng and S. Prabhakar. Managing Uncertainty in Sensor Databases. *SIGMOD Record*, 32(4):41–46, 2003.

[cro] Stargate gateway by Crossbow. http://www.xbow.com/products/xscale.htm.

[CT00] Jae-Hwan Chang and Leandros Tassiulas. Energy Conserving Routing in Wireless Ad-hoc Networks. In *INFOCOM*, 2000.

[DGM+04] A. Deshpande, C. Guestrin, S. Madden, J.M. Hellerstein, and W. Hong. Model-Driven Data Acquisition in Sensor Networks. In *VLDB*, 2004.

[DGR01] A. Deshpande, M. Garofalakis, and R. Rastogi. Independence is Good: Dependency-Based Histogram Synopses for High-Dimensional Data. In *ACM SIGMOD*, 2001.

[DGR+03] Alan Demers, Johannes Gehrke, Rajmohan Rajaraman, Niki Trigoni, and Yong Yao. The Cougar Project: A Work In Progress Report. *SIGMOD Record*, 32(4):53–59, 2003.

[DKR04] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical in-Network Data Aggregation with Quality Guarantees. In *Proceedings of EDBT*, 2004.

[DR03] A. Deligiannakis and N. Roussopoulos. Extended Wavelets for Multiple Measures. Technical Report CS-TR-4462, University of Maryland, March 2003.

[EGHK99] D. Estrin, R. Govindan, J. Heidermann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *MobiCOM*, 1999.

[FvDFH90] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, second edition, 1990.

[GEH02] D. Ganesan, D. Estrin, and J. Heidermann. DIMENSIONS: Why do we need a new Data Handling architecture for Sensor Networks? In *HotNets-I*, 2002.

[GG02] M. Garofalakis and P. B. Gibbons. Wavelet Synopses with Error Guarantees. In *ACM SIGMOD*, 2002.

[GG04] M. Garofalakis and P. B. Gibbons. Probabilistic Wavelet Synopses. In *ACM Transactions on Database Systems, 29(1)*, 2004.

[GGC03] Abhishek Ghose, Jens Grossklags, and John Chuang. Resilient Data-Centric Storage in Wireless Ad-Hoc Sensor Networks. In *Mobile Data Management*, 2003.

[GHK92] S. Ganguly, W. Hasan, and R. Krishnamurthy. Query Optimization for Parallel Execution. In *ACM SIGMOD*, 1992.

[Gib01] P. B. Gibbons. Distinct Sampling for Highly-Accurate answers to Distinct Values Queries and Event Reports. In *Proc. of the 27th VLDB Conf.*, 2001.

[GK04] M. Garofalakis and A. Kumar. Deterministic Wavelet Thresholding for Maximum-Error Metrics. In *ACM PODS*, 2004.

[GKMS01] A.C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M.J. Strauss. Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries. In *VLDB*, 2001.

[GTK01] L. Getoor, B. Taskar, and D. Koller. Selectivity Estimation using Probabilistic Models. In *Proceedings of ACM SIGMOD*, 2001.

[HCB00] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *HICSS*, 2000.

[HFC+00] J.M. Hellerstein, M.J. Franklin, S. Chandrasekaran, A. Descpande, K.Hildrum, S. Madden, V. Raman, and M.A. Shah. Adaptive Query

Processing: Technology in Evolution. In *IEEE Data Engineering Bulletin 23(2)*, 2000.

[HS92] P. Haas and A. Swami. Sequential sampling procedures for query size estimation. In *ACM SIGMOD*, 1992.

[HSI+01] J. Heidermann, F. Silva, C. Intanagonwiwat, R. Govindanand D. Estrin, and D. Ganesan. Building Efficient Wireless Sensor Networks with Low-Level Naming. In *SOSP*, 2001.

[IEGH02] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidermann. Impact of Network Density on Data Aggregation in Wireless Sensor Networks. In *ICDCS*, 2002.

[IP00] Y. E. Ioannidis and V. Poosala. Histogram-Based Approximation of Set-Valued Query Answers. In *Proc. of the 25th VLDB Conf*, 2000.

[JS94] B. Jawerth and W. Sweldens. An Overview of Wavelet Based Multiresolution Analyses. In *SIAM Review, 36(3):377–412*, 1994.

[KDG03] D. Kempe, A. Dobra, and J. Gehrke. Gossip-Based Computation of Aggregate Information. In *FOCS*, 2003.

[KLKF00] F. Korn, A. Labrinidis, Y. Kotidis, and C. Faloutsos. Quantifiable Data Mining Using Ratio Rules. *VLDB Journal*, 8(3-4):254–266, 2000.

[Kot05] Y. Kotidis. Snapshot Queries: Towards Data-Centric Sensor Networks. In *Proceedings of ICDE*, 2005.

[KT01] S. Khanna and W. C. Tan. On Computing Functions with Uncertainty. In *ACM PODS Conference*, 2001.

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

[LKC99] J. Lee, D. Kim, and C. Chung. Multi-dimensional Selectivity Estimation Using Compressed Histogram Information. In *ACM SIGMOD*, 1999.

[LM04] I. Lazaridis and S. Mehrotra. Approximate Selection Queries over Imprecise Data. In *ICDE*, 2004.

[LR02] S. Lindsey and C.S. Raghavendra. Pegasis: Power-Efficient Gathering in Sensor Information Systems. In *IEEE Aerospace Conference*, 2002.

[MFHH02] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: A Tiny Aggregation Service for ad hoc Sensor Networks. In *OSDI Conf.*, 2002.

[MFHH03] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The Design of an Acquisitional Query processor for Sensor Networks. In *ACM SIGMOD Conf*, June 2003.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[MVW98] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-Based Histograms for Selectivity Estimation. In *ACM SIGMOD*, 1998.

[MVW00] Y. Matias, J.S Vitter, and M. Wang. Dynamic Maintenance of Wavelet-Based Histograms. In *Proc. of the 26th VLDB Conf.*, 2000.

[MWA+03] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query Processing, Resource Management, and Approximation in a Data Stream Management System. In *CIDR*, 2003.

[net] Netflow Services and Applications. http://www.cisco.com/.

[OJW03] C. Olston, J. Jiang, and J. Widom. Adaptive Filters for Continuous Queries over Distributed Data Streams. In *ACM SIGMOD Conference*, pages 563–574, 2003.

[OLW01] C. Olston, B. T. Loo, and J. Widom. Adaptive Precision Setting for Cached Approximate Values. In *ACM SIGMOD*, 2001.

[OW00] C. Olston and J. Widom. Offering a Precision-Performance Tradeoff for Aggregation Queries over Replicated Data. In *VLDB Conference*, pages 144–155, 2000.

[OW02] C. Olston and J. Widom. Best-effort Cache Synchronization with Source Cooperation. In *ACM SIGMOD Conference*, pages 73–84, 2002.

[PF95] V. Paxson and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[PG99] V. Poosala and V. Ganti. Fast Approximate Answers to Aggregate Queries on a Data Cube. In *Proc. of the 1999 Intl. Conf. on Scientific and Statistical Database Management*, 1999.

[PI97] V. Poosala and Y. E. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *Proc. of the 23th VLDB Conf.*, 1997.

[PIHS96] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. In *ACM SIGMOD*, 1996.

[PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition edition, 1992.

[RKY+02] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A Geographic Hash Table for Data-Centric Storage. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[SBLC04] A. Sharaf, J. Beaver, A. Labrinidis, and P. Chrysanthis. Balancing Energy Efficiency and Quality of Aggregate Data in Sensor Networks. *VLDB Journal*, 2004.

[SCI+01] E. Shih, S.H. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan. Physical Layer Driven Protocol and Algorithm Design for Energy-Efficient Wireless Sensor Networks. In *MOBICOM*, 2001.

[SDS96] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. Wavelets for Computer Graphics - Theory and Applications. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1996.

[SS90] N. Soparkar and A. Silberschatz. Data-value Partitioning and Virtual Messages. In *Proceedings of PODS*, pages 357–367, Nashville, Tennessee, April 1990.

[SS02] R. R. Schmidt and C. Shahabi. Propolyne: A Fast Wavelet-Based Algorithm for Progressive Evaluation of Polynomial Range-Sum Queries. In *EDBT*, 2002.

[SWR98] S. Singh, M. Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *ACM/IEEE International Conference on Mobile Computing and Networking*, 1998.

[TGIK02] N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic Multidimensional Histograms. In *ACM SIGMOD*, 2002.

[TGNO92] D.B. Terry, D. Goldberg, D. Nichols, and B.M. Oki. Continuous Queries over Append-Only Databases. In *ACM SIGMOD*, 1992.

[TK03] H.O. Tan and I. Korpeoglu. Power Efficient Data Gathering and Aggregation in Wireless Sensor Networks. *SIGMOD Record*, 32(4), 2003.

[Vit85] J. S. Vitter. Random Sampling with a Reservoir. In *ACM TOMS*, 1985.

[VN02] S. D. Viglas and J. F. Naughton. Rate-based Query Optimization for Streaming Information Sources. In *ACM SIGMOD Conference*, pages 37–48, 2002.

[VW99] J.S Vitter and M. Wang. Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets. In *Proceedings of ACM SIGMOD*, 1999.

[YF04] O. Younis and S. Fahmy. HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks. *IEEE Transactions on Mobile Computing*, 3(4), 2004.

[YG02] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *SIGMOD Record*, 31(3):9–18, 2002.

[YH03] W. Ye and J. Heidermann. Medium Access Control in Wireless Sensor Networks. Technical report, USC/ISI, 2003.

[ZSC+03] S. B. Zdonik, M. Stonebraker, M. Cherniack, U. Cetintemel, M. Balazinska, and H. Balakrishnan. The Aurora and Medusa Projects. *IEEE Data Engineering Bulletin*, 2003.