Fast Data Streaming in Resource Constrained Wireless Sensor Networks

by

Emad Soroush

BSc, Sharif University of Technology, 2006

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

University of Victoria

Fast Data Streaming in Resource Constrained Wireless Sensor Networks

by

Emad Soroush

Supervisory Committee

Dr. Kui Wu, Supervisor (Department of Computer Science)

Dr. Jianping Pan, Member (Department of Computer Science)

Dr. Alex Thomo, Outside Member (Department of Computer Science)

Supervisory Committee

Dr. Kui Wu, Supervisor (Department of Computer Science)

Dr. Jianping Pan, Member (Department of Computer Science)

Dr. Alex Thomo, Outside Member (Department of Computer Science)

Abstract

In many emerging applications, data streams are monitored in a network environment. Due to limited communication bandwidth and other resource constraints, a critical and practical demand is to online compress data streams continuously with quality guarantee. Although many data compression and digital signal processing methods have been developed to reduce data volume, their super-linear time and more-than-constant space complexity prevents them from being applied directly on data streams, particularly over resource-constrained sensor networks. In this thesis, we tackle the problem of online quality guaranteed compression of data streams using fast linear approximation (i.e., using line segments to approximate a time series). Technically, we address two versions of the problem which explore quality guarantees in different forms. We develop online algorithms with linear time complexity and constant cost in space. Our algorithms are optimal in the sense that they generate the minimum number of segments that approximate a time series with the required quality guarantee. To meet the resource constraints in sensor networks, we also develop a fast algorithm which creates connecting segments with very simple computation.

The low cost nature of our methods leads to a unique edge on the applications of massive and high speed streaming environment, low bandwidth networks, and heavily constrained nodes in computational power (e.g., tiny sensor nodes). We implement and evaluate our methods in the application of an acoustic wireless sensor network.

Table of Contents

Supervisory Committee **ii**

Abstract **iii**

Table of Contents **v**

List of Tables **vii**

List of Figures **viii**

Acknowledgements **x**

1 Introduction **1**

1.1 Motivation 1

1.2 Contribution 4

2 Problem Definition and Related Work **6**

2.1 Problem Formulation 6

2.2 Related Work 10

3 Online Algorithms **13**

3.1 The Framework 13

3.2 Solving the PLA-PointBound Problem 15

3.3 Solving the PLA-SegmentBound Problem 19

4 Optimality **23**

5 PLAZA for Tiny Sensors **26**

5.1 PLAZA 26

5.2 Benchmarking PLAZA 29

6 Multidimensional PLA Problems **32**

6.1 Problem Formulation 32

6.2 Multidimensional PLA-SegmentBound Problem 33

6.3 Multidimensional PLA-PointBound Problem 35

7 Experimental Evaluation **38**

7.1 Existing Work 38

7.2 Experimental Setting 39

7.3 Results on Quality 41

7.4 Benchmarking PLAZA 45

7.5 Results on Real Sensors 46

7.6 Evaluation in Other Applications 48

7.7 Experimental Results of Multidimensional PLA Problems 49

8 Conclusion **51**

Bibliography **54**

List of Tables

7.1 3-dimensional accelerometer data from a sony ERS-210 aibo robot . . 49

List of Figures

2.1 Piecewise linear approximation. 7

3.1 Ranges of possible line segments. 15
3.2 Polygon $poly(i, j)$. 16
3.3 PointBound, an online algorithm for the PLA-PointBound problem. . 17
3.4 Using up to 4 edges to represent the intersection polygon. 18
3.5 SegmentBound, an online algorithm for the PLA-SegmentBound problem. 21

5.1 An example of zoning angle 27
5.2 Algorithm PLAZA. 28
5.3 Optimal PLAZA benchmark algorithm 31

6.1 M-SegmentBound, an online algorithm for the multidimensional PLA-SegmentBound problem. 34
6.2 ErrorBoundAdjustment, the algorithm to adjust error bound among different dimensions for the PLA-PointBound problem. 37

7.1 The waveform of the human voice data (the lower part is in a smaller time scale). 39
7.2 The waveform of the piano music data (the lower part is in a smaller time scale). 39
7.3 The sample reduction ratio of PointBound with respect to error bounds. 40

7.4 The sample reduction ratio of SegmentBound with respect to error bounds. 40
7.5 The sample reduction ratio of PLAZA with respect to error bounds. . 42
7.6 Comparison of the three algorithms on the human voice data set. . . 43
7.7 The distortion on the human voice dataset. 44
7.8 The distortion on the piano music voice dataset. 44
7.9 The comparison between algorithm PLAZA and the PLAZA benchmark method. 46
7.10 The test bed using real sensors. 47
7.11 Results on an ECG data set. 48
7.12 Multi-Dimensional PLA Problem 50

Acknowledgements

I would like to thank all people who have helped and inspired me during my Master study.

I would like to express my deep and sincere gratitude to my supervisor, Dr. Kui Wu. His wide area of knowledge and his logical way of thinking have been of great value for me. His understanding, encouraging and personal guidance have laid a good basis for this thesis. I thank him for his constructive comments that triggered my thinking and gave me inspiration to improve this thesis work further.

I want to thank Dr. Jian Pei for his informative comments and his great efforts in pinpointing technical errors and improving the presentation of the thesis.

I am also grateful to Dr. Jianping Pan and Dr. Alex Thomo for their willingness to serve in my committee and to Dr. T. Aaron Gulliver for the constructive feedbacks he provided in the preparation of the thesis.

I warmly thank Dr. Valerie King for her valuable advice and friendly help. It was truly my honour to be her student. I wish to thank her because of her endless support and the insights she provided throughout my Master study.

I would also like to acknowledge my office and lab mates: Yuehhua Lee, Zhang Xi, Steven Lonergan, Marco Yuen, Rebeca Dunn-Krahn, Celina Gibbs, Onat Yazir, Dr. Yvonne Coady and all the other people who provided an enjoyable and memorable atmosphere during my study in UVic.

I owe my most sincere gratitude to my dear friends Bahar Javan, Saloome Motavas, Solmaz Khezerloo, Alireza Hamidi, and my dear cousin Sahar Karimi for helping me get through the difficult times and for their emotional supports throughout the ups and downs of the last two years.

Finally, I cannot finish without saying how grateful I am with my family. It is difficult to overstate my gratitude to my brothers Ali and Milad, my sister's family Niloofar, Babak, and dear Nika, and most importantly I wish to thank my mother and father. They have always supported and encouraged me to do my best in all matters of life. To them I dedicate this thesis.

This research was supported by Natural Sciences and Engineering Research Council of Canada (NSERC), Canada Foundation for Innovation (CFI), and the British Columbia Knowledge Development Fund (BCKDF). We also thank Dr. E. Keogh for his informative comments and for providing the ECG dataset.

Chapter 1

Introduction

1.1 Motivation

Massive data streams are the large quantities of data that arrive in a continuous manner. The application of massive date streams is becoming increasingly common in many fields of science and technology. For example, large sensor networks are extensively used in wildlife monitoring, road traffic monitoring, and environment surveillance. Each sensor generates a data stream where new data items keep arriving in a continuous manner. Large sensor networks usually consist of tiny devices with very limited resources in computation and storage. In order to aggregate and analyze the massive streaming data under monitoring, it is often required to store them in a node with high computational and storage capability called base station. As a result, massive streaming data has to be transmitted from tiny sensor nodes to the base station. Due to often limited communication bandwidth and other resource constraints, online compressing data streams continuously with quality guarantee comes as a natural, critical, and practical demand in those applications.

We use several concrete examples to emphasize the importance of this problem:

An acoustic monitoring system has many applications. An appealing scenario is towards "smart conference hall." By analyzing the data collected from an acoustic monitoring system deployed in a large conference place, we can identify and locate

speakers as well as some of their activities. The information can be used to adjust the equipment such as the light system, the microphone system, the video monitoring system, and the air conditioning system.

The other potential application is bird surveillance in wildness. By analyzing the bird sound collected using such a sensor network, ornithologists can study the distribution of birds and their behavior patterns.

Also acoustic monitoring system can be used in air control system. Sensors can be installed on the aircraft and provide wide area detection of acoustic sources (heavy vehicles, other aircrafts, etc.). Underwater monitoring is another interesting application of acoustic monitoring systems [3]. Sensor nodes customized for ocean environment can be used for oceanographic data collection, pollution monitoring, offshore exploration, and disaster prevention. Autonomous underwater vehicles equipped with underwater sensors can explore very deep part of the ocean and gather scientific data for monitoring missions. The traditional approach for underwater monitoring is to deploy custom-built sensors that record data during the monitoring mission, and then recover the data after the mission. This approach has many disadvantages and is inapplicable in many emerging situations: First, real-time monitoring that is of great importance in surveillance becomes impossible. The data needs to be recorded and recovered at a later time because it is usually hard to access the offshore vehicles after deployment. Second, there is no interaction between onshore control systems and the monitoring instruments, impeding any interactive reconfiguration after specific events. Also the amount of data that can be recorded by the sensors is limited by their storage capacity. To build real-time interactive underwater monitoring systems, there is a need to enable underwater communications among underwater devices. Underwater sensor nodes and vehicles must be able to coordinate their operation by exchanging configuration, location, and movement data, and to relay monitored information to an onshore station. In many cases, we use acous-

tic wireless communication for underwater sensor networks, which suffers from very limited bandwidth.

We are building an acoustic monitoring system using wireless sensor networks. Sensor nodes are deployed in a target area, while each node contains an acoustic sensor that samples sound signals continuously. Wireless sensor nodes that integrate sensors, processors, memory, and wireless transceivers are small and have only very limited computational power and communication bandwidth [33]. For instance, we select the Chipcon radio chip in the broadly-used MICA2 motes [16] from Crossbow Technology Inc [1] that has the maximum transmission power of 27 mA and the maximum bandwidth of 38 kbps. The model of the motes that we use is MPR400CB with the CC1000 900 Mhz data radio. The sensor board, MTS300CA, enables the mote to measure temperature, sound, and light in addition to the voltage of the batteries in the node. The base station interface unit, MIB510CA, is based on RS232.

One technical challenge is that the acoustic data stream cannot be sent out in time due to the low bandwidth radio channel. Specifically, in order to make the data analysis useful, we need to sample human voice with the normal sampling rate of 8 kHZ and 16 bits per sample. This sampling mode requires a bandwidth of 128 kbps for 1 channel (mono) voice, which greatly exceeds the maximum bandwidth of 38 kbps that a MICA2 mote can support (In underwater wireless sensor networks, no matter using sound or radio waves for communication, the challenge still exists). In addition, we cannot temporarily store a large number of samples since the memory size of MICA2 motes is only 512 kb. The only technical solution to the bottleneck is to compress data streams on the fly with the minimum required quality, and then send out the compressed streams instead of the original streams through the network. Sending compressed streams can also reduce the power consumption of sensors on communication, and thus extend the lifetime of sensors. In large environmental

[1] Crossbow Technology Inc is a manufacturer of RF Sensor Network hardware and software WEB: www.xbox.com

surveillance sensor networks, recharging or replacing batteries of sensor nodes is often very difficult or even impossible after the sensors are deployed.

Many data compression and digital signal processing methods have been developed to reduce data volume, such as Fourier transform [1, 12, 18, 34], discrete cosine transform [29], Wavelets [5, 7, 30, 13, 25], linear predictive coding (LPC) [2], etc. However, these methods cannot be applied to data stream compression in sensor networks due to the high cost of those methods in time and space. Moreover, sensor nodes like MICA2 motes only have very limited computational power. For example, only simple arithmetic operations are supported by TinyOS [6], the operating system for MICA2 motes. Although it is possible to implement a mathematical module to calculate essential functions like sinusoid and exponential functions or use dedicated DSP chips for audio processing and compression, such complex modules are highly undesirable due to the limited memory size and computational capacity of MICA2 motes as well as the extra energy cost of dedicated DSP chips.

1.2 Contribution

In this thesis, we tackle the problem of online compression of data streams in the application context of sensor networks. Particularly, we aim at the fast linear approximation methods (i.e., using line segments to approximate a time series) with quality guarantee. We make the following contributions.

1. We model the piecewise linear approximation problem properly for data streams. Different from the conventional situations where the whole time series to be compressed and the required compression rate can be specified, a data stream is potentially unlimited, and the distribution is often unpredictable. We propose the error-bounded piecewise linear approximation problem to tackle those challenges.

2. We present fast online solutions with linear time complexity and constant cost

in space. Our algorithms are optimal in the number of segments used to approximate a (potentially unlimited) time series. Moreover, our algorithms have an approximation factor of 2 to the maximum compression ratio using piecewise linear approximation.

3. To address the computational challenges in sensor nodes, we develop another online approximation algorithm that is particularly tailored for tiny sensor devices by requiring only very simple computation. The low cost nature of our methods leads to a unique edge on the applications of massive and high speed streaming environment, low bandwidth networks, and heavily constrained nodes in computational power (e.g., tiny sensor nodes).

4. We implement and evaluate our methods in the application of an acoustic wireless sensor network. Our empirical evaluation clearly shows that our methods are highly feasible for resource-constrained wireless sensor networks.

The rest of the thesis is organized as follows. In Chapter 2, we formulate and analyze the problem and review the related work. Two online algorithms are developed in Chapter 3, and their optimality is studied in Chapter 4. In Chapter 5, we design an online approximation algorithm which is more economic in computation for tiny sensors. Multidimensional PLA problems are studied in Chapter 6. We report our implementation and evaluation of the proposed methods in an acoustic wireless sensor network in Chapter 7. The thesis is concluded in Chapter 8.

Chapter 2

Problem Definition and Related Work

In this chapter, we propose the error-bounded piecewise linear approximation problem for data streams. We also review the related work.

2.1 Problem Formulation

A very popular method for approximation of a time-series is by combination of polynomials. The simplest approach is to use first degree polynomials, i.e. linear approximation. Piecewise linear approximation (PLA) is an effective method to compress a time series. A numeric data stream can be treated as a potentially unlimited time series. Thus, it is natural to explore whether we can compress a numeric data stream using the piecewise linear approximation method. PLA algorithms can also be classified as static or online algorithms. In the static or conventional PLA algorithm, the entire time series (numeric data stream) is accessible at the beginning, but in the online PLA problem data stream is treated as an infinite time series. This is an important distinction because many data mining algorithms are required to work with infinite stream [31, 21]. PLA for static time series has been well studied (e.g., [10, 14, 20, 26]), But there is not substantial effort in the literature to cover online PLA problem [31, 21, 19].

Let $X = x_1 \cdots x_n$ be a time series of n points, and x_i $(1 \leq i \leq n)$ be the value of the i-th point of X. A *(line) segment* is a tuple $s = ((i, y_i), (j, y_j))$ where $i < j$ and

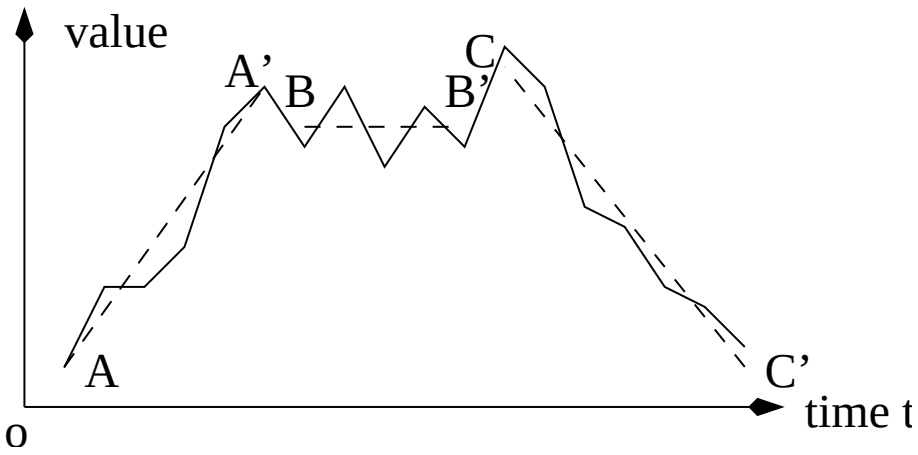

Figure 2.1: Piecewise linear approximation.

(i, y_i) and (j, y_j) are two endpoints. $[i, j]$ is called the *range* of s.

Given a time series X, PLA uses a set of line segments as the approximation of the time series. Figure 2.1 elaborates the general idea, where three line segments, AA', BB', and CC', are used to approximate a time series. A line segment $s = ((i, y_i), (j, y_j))$ approximates the k-th point $(i \leq k \leq j)$ of the time series by value

$$\tilde{x}_k = y_i + \frac{k - i}{j - i} y_j$$

The compression comes from that the number of line segments used for approximation can be much smaller than the number of points in the time series. In the figure, the time series has 18 points. 3 segments are used to approximate the time series, and each segments has 2 endpoints. Thus, the 3 line segments only need 6 points to represent. A compression ratio of 3 is achieved.

Regarding the endpoints in the segments, there are at least two ways to classify the PLA algorithms. In the first way, the endpoint of each segment must belong to the original time series. In the second way, the endpoints in the segments are not necessarily from the original points. In the first way, we tend to closely align the endpoint of consecutive segments, and give the piecewise approximation a smooth

look. In contrast, the selection of the second way can produce a very disconnected look on some datasets. The aesthetic superiority of the connected segments , together with its low computational complexity has made it a suitable technique in computer graphic applications [15]. However, the quality of the approximating line while they are required to be connected is generally inferior to the second approach. Here we assume that the endpoints in the segments are not necessarily positioned at some points in the time series (e.g., B, B', and C' in the figure).

Formally, a set of segments $\tilde{X} = \{s_1, \ldots, s_m\}$ is a piecewise linear approximation of X if (1) $s_1, \ldots, s_m$ are segments; and (2) for each index i $(1 \leq i \leq n)$, i is either in the range of exactly one segment in $\tilde{X}$, or there exist two segments $s, s' \in \tilde{X}$ such that s and s' share the same endpoint at index i. Clearly, using the segments, for every index i, $\tilde{X}$ can give a value $\tilde{x}_i$ to approximate x_i.

Most of the previous studies on static time series address an optimization problem as follows.

Problem 1 (Conventional PLA problem) *Given a time series X of n points and a number $m < n$, find a set of m segments as a piecewise linear approximation of X such that the approximation error is minimized.* ■

Unfortunately, solutions to the conventional PLA problem are not applicable to data streams. A data stream is potentially unlimited. It is impossible to know in advance the number of points in the stream or to specify the number of segments to be used for approximation. To tackle the stream compression problem we turn to the *error-bounded PLA problem.*

Problem 2 (Error-bounded PLA problem) *Given an error measurement function $err()$ such that $err(X, \tilde{X})$ gives the error that a PLA $\tilde{X}$ approximates X. Let ϵ be a user-specified error bound. $\tilde{X}$ is called an ϵ-PLA of X if $err(X, \tilde{X}) \leq \epsilon$. An*

ϵ-PLA $\tilde{X}$ of X is optimal *if $|\tilde{X}|$ (i.e., the number of segments in $\tilde{X}$) is minimized.*

■

We propose two error measurement functions meaningful for data streams.

First, the *max-err* function captures the maximal error between X and $\tilde{X}$ at any index. That is,

$$maxerr(X, \tilde{X}) = \max_{i=1}^{n}\{|x_i - \tilde{x}_i|\}$$

With potentially unlimited streams, using the max-err function, we can make sure the approximation quality is consistently bounded at every point.

Second, the *seg-err* function checks the error introduced by each segment, and captures the maximal error. That is,

$$segerr(X, \tilde{X}) = \max_{s \in \tilde{X}}\{\sum_{i \in range(s)} (x_i - \tilde{x}_i)^2\}$$

Using the seg-err function, we can make sure that the error introduced by every segment is bounded.

Using the two error measurement functions, we have two versions of the error-bounded PLA problem.

Problem 3 (PLA-PointBound problem) *Given an error-bound ϵ, the* PLA-PointBound problem *is to find an ϵ-PLA $\tilde{X}$ such that $maxerr(X, \tilde{X}) \leq \epsilon$ and $|\tilde{X}|$ is minimized.*

■

Problem 4 (PLA-SegmentBound problem) *Given an error-bound ϵ, the* PLA-SegmentBound problem *is to find an ϵ-PLA $\tilde{X}$ such that $segerr(X, \tilde{X}) \leq \epsilon$ and $|\tilde{X}|$ is minimized.*

■

It is worth noting that different versions of the ϵ-PLA problem require different solutions. Generally, the PLA-PointBound problem composes a stronger quality

requirement than the PLA-SegmentBound problem, and thus the PLA-PointBound problem is more challenging.

A good solution to one problem does not necessarily result in a good solution to the other. For instance, suppose that we have an effective algorithm to solve the PLA-PointBound problem. One may wonder whether we can translate the PLA-SegmentBound problem to the PLA-PointBound problem by setting the error-bound (of each point) to $\sqrt{\frac{\epsilon}{n}}$, where n is the number of points approximated by a line segment.

Unfortunately, it is hard to obtain the number of points in a segment. The segments may have different lengths. Moreover, a segment in the PLA-SegmentBound problem might approximate most points well, but permits several exceptional points to have large errors. A naïve translation of the PLA-SegmentBound problem to the PLA-PointBound problem using a uniform error-bound on all points excludes such segments. Similarly, we cannot easily reduce the PLA-PointBound problem to the PLA-SegmentBound problem.

2.2 Related Work

Piecewise linear approximation (PLA) is a well-investigated research area [9, 19, 20, 26, 27, 32]. The idea is to use a sequence of line segments to represent the time series with a low approximation error. Since a line segment can be determined by only two endpoints, PLA can make the storage, transmission and computation of time series more efficient. In data mining, most existing piecewise linear approximation algorithms use standard linear regression techniques to calculate a line segment fitting the original data with the minimum mean squared error. Many of them target at solving the conventional PLA problem and are not applicable to streaming data [10, 14, 20, 26].

Besides data mining, PLA algorithms are also used in other fields like cartogra-

phy [8] and computer graphics [15]. In [8], the authors use 2D linear segments in order to approximate map contours. In [15], the authors use linear segments for the simplification of polygonal surfaces. In both applications, the aesthetic feature of the approximation is the objective, and linear interpolation (connected segments) has been utilized.

Despite the substantial research efforts in PLA techniques [10, 14, 19, 23, 20, 26], existing solutions are not tailored for data streams over resource-constrained sensor networks. They either require complex computation or have high cost in space. To the best of my knowledge, there has no implementation of these algorithms in realistic sensor devices.

In [19], Keogh et al. give a comprehensive review of existing techniques for segmenting time series. They categorize the solutions into three different groups, namely sliding window methods, top-down methods, and bottom-up methods. In sliding window, a segment is extending as large as possible until it exceeds some previously defined error bound. Sliding window is a broadly used algorithm in different areas. That is mainly because of its simplicity to implement and online characteristic of the algorithm. Top-down and bottom-up approaches are both working with static dataset. Top-down approach recursively partitions the segments until it exceeds some threshold. So an initial segmentation is required to start the algorithm. Note that the first segmentation can be the whole time series. In bottom-up, we start with the finest possible approximation and then segments are merged until it exceeds some threshold. In [19], they take advantage of both sliding window and bottom-up methods and design a Sliding-Window-And-Bottom-up (SWAB) algorithm. The SWAB algorithm uses a moving window to constrain a time period in consideration.

Sliding window has been broadly used in different areas especially with medical applications [21, 31, 17]. The main reason of using this type of PLA algorithm backs to its simplicity and online characteristic of the algorithm. Although different

variations of sliding window are proposed in the literature [21, 24, 32], none covers a large range of dataset with different characteristics, and it is shown that sliding window can give poor results under some circumstances [28].

In [23], an amnesic function is introduced to give weights to different points in the time series. The PLA-SegmentBound problem is discussed in the context of Unrestricted Window with Absolute Amnesic (UAA) problem, but complete solutions to this problem are not provided in [23].

A solution to the PLA-PointBound problem is addressed in [22] with a different definition of point error bound. The algorithm is claimed to be optimal, but the time complexity is $O(n^3)$ where n is the number of points in the time series. Moreover, no performance evaluation of the solution is presented in the paper.

In [4], a solution is introduced for the PLA-PointBound problem. The proposed algorithm extends the sliding window model without an asymptotic increase in memory space. The memory space complexity of this algorithm is linear in the size of the bucket histogram

In summary, although the error-bounded PLA problem has been touched slightly in several very recent studies, the problem has not been studied systematically. No solution applicable to data streams has been developed, let alone solutions for resource-constrained sensor networks.

Chapter 3

Online Algorithms

In this chapter, we develop two online algorithms for the PLA-PointBound and the PLA-SegmentBound problems, respectively. The two algorithms share the same framework.

3.1 The Framework

The framework of our algorithms works in a greedy manner. When x_1, the first point in the stream, arrives, we store x_1. When x_2 arrives, we also store x_2 since x_1 and x_2 can be compressed by a segment exactly. When x_3 arrives, we check whether x_3 can be compressed together with x_1 and x_2 by a line segment satisfying the error-bound requirement. If so, we store x_3. Otherwise, we output a line segment compressing x_1 and x_2, remove x_1 and x_2 from the main memory, and store x_3.

Generally, imagine we have a buffer in main memory storing points $x_i, x_{i+1}, \ldots, x_j$ such that the points in the buffer can be compressed by a line segment satisfying the error-bound requirement. When a new point x_{j+1} arrives, we check whether x_{j+1} can be compressed together with $x_i, \ldots, x_j$ by a line segment satisfying the error-bound requirement. If so, we add x_{j+1} to the buffer and move on to the next point. Otherwise, we output a segment compressing $x_1, \ldots, x_j$ satisfying the error-bound requirement, and remove them from the buffer. x_{j+1} is then stored in the buffer.

Although the framework is simple, there are two critical issues that need to be

solved carefully in order to make sure that the runtime of the algorithms is linear with respect to the number of points in the streams, and the space size needed by the algorithms is bounded by a constant.

First, how can we store the information about the points we have seen but have not compressed? In the worst case, there can be an unlimited number of such points (e.g., a times series where all points take the same value). How can we summarize them using only constant size memory?

Second, how can we determine whether a newly arrived point can be compressed together with the points already in the buffer that have been seen but have not been compressed? Revisiting those points one by one leads to the runtime quadratic with respect to the number of such points. As explained before, there can be an unlimited number of such points. The overall time complexity is quadratic if those points are revisited one by one.

Our central idea to tackle the above two challenges is the following. Instead of storing the points explicitly, we monitor the range of all possible line segments that can be used to compress the points that have been seen but have not been compressed in a concise way. When a new point arrives, we can check whether the point can be compressed using some line segment in the range. If so, it means that the new point can be compressed together with the points accumulated. We only need to adjust the range of the possible line segments to make sure the new point is also compressed. If not, it means that the new point cannot be compressed together with the points accumulated. A segment should be output.

In the rest of this chapter, we develop the techniques to implement the above idea for the PLA-PointBound problem and the PLA-SegmentBound problem.

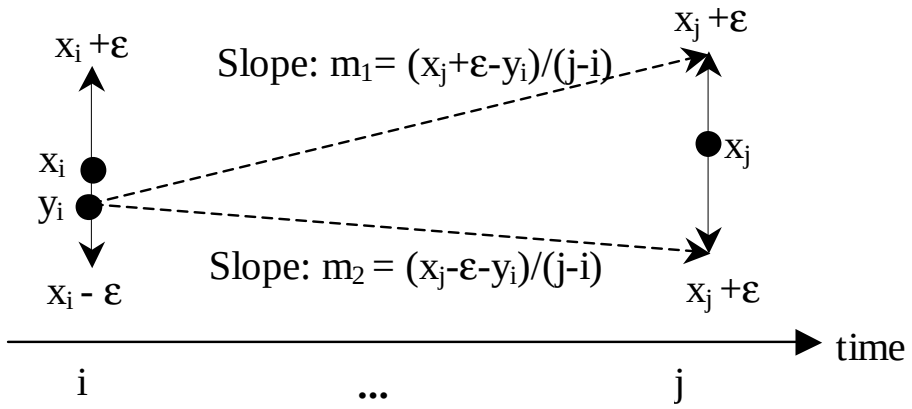

Figure 3.1: Ranges of possible line segments.

3.2 Solving the PLA-PointBound Problem

A segment $s = ((i, y_i), (j, y_j))$ can also be represented by the left endpoint (i, y_i), the slope $m = \frac{y_j - y_i}{j-i}$, and the index of the right endpoint j.

For two points x_i and x_j in a data stream, if a line segment $s = ((i, y_i), (j, y_j))$ with slope $m = \frac{y_j - y_i}{j-i}$ can approximate x_i and x_j, i.e., $|x_i - \tilde{x}_i| \leq \epsilon$ and $|x_j - \tilde{x}_j| \leq \epsilon$ where ϵ is the error-bound, s must satisfy the following two conditions.

$$(x_i - \epsilon) \leq y_i \leq (x_i + \epsilon) \tag{3.1}$$

$$m_2 \leq m \leq m_1 \tag{3.2}$$

Where $m_1 = \frac{(x_j+\epsilon)-y_i}{j-i}$ and $m_2 = \frac{(x_j-\epsilon)-y_i}{j-i}$. Figure 3.1 illustrates the conditions and their relations. Particularly, m_1 and m_2 are the slopes of the two lines shown in the figure.

Since the line segments are determined by the value of the left endpoint y_i and slope m, we examine the distribution of points (y_i, m) that satisfy Equations 3.1 to 3.2. As illustrated in Figure 3.2, the possible line segments form a polygon $poly(i, j)$. We have the following important result.

Lemma 1 (PLA-PointBound) *A line segment of left endpoint y_i and slope m can*

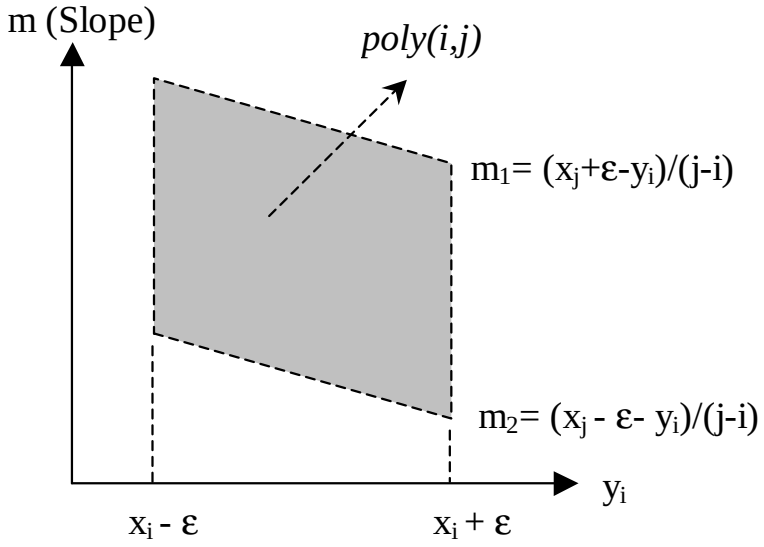

Figure 3.2: Polygon $poly(i, j)$.

approximate points $x_i, \ldots, x_j$ with max-err at most ϵ if and only if (y_i, m) is in polygon $poly(i, i+1) \cap poly(i, i+2) \cap \cdots \cap poly(i, j)$.

Proof. The necessity follows with the definition of $poly(i, j)$. For any line segment $s \notin poly(i, i+1) \cap poly(i, i+2) \cap \cdots \cap poly(i, j)$, there exists an index k $(i \leq k \leq j)$ such that $s \notin poly(i, k)$, i.e., s cannot approximate either x_i or x_k.

We prove the sufficiency by contradiction. Suppose a segment $s \in poly(i, i+1) \cap poly(i, i+2) \cap \cdots \cap poly(i, j)$ but s cannot approximate x_k $(i \leq k \leq j)$. Two situations may arise. First, $k = i$. Then, $s \notin poly(i, i+1)$ since $|x_i - y_i| > \epsilon$ where y_i is the value of s on index i. Second, $k \neq i$. Then, $s \notin poly(i, k)$. In both cases, we have contradictions. ∎

Using Lemma 1, we have algorithm PointBound, an online algorithm as shown in Figure 3.3. We maintain the intersection of polygons $poly(i, i+1)$, $\ldots$, $poly(i, j)$, where x_i is the first point that has not been compressed yet in the data stream, and x_j is the last point arrived such that $poly(i, i+1) \cap \ldots \cap poly(i, j) \neq \emptyset$.

When a new point x_{j+1} arrives, we compute $poly(i, j+1)$ and $poly(i, i+1) \cap \ldots \cap poly(i, j) \cap poly(i, j+1)$. If it is $\emptyset$, then a line segment s is randomly chosen to

Input: a data stream $X = x_1, x_2, \ldots$ and error-bound ϵ;
Output: a list of line segments $\tilde{X}$ approximating X
such that $maxerr(X, \tilde{X})) \leq \epsilon$;
Method:

```
1:  P = poly(1,2); i = 1; j = 3;
2:  WHILE (1) DO {
3:    P' = P ∩ poly(i,j);
4:    IF P' ≠ ∅ THEN P = P', j = j + 1;
5:    ELSE {
6:      randomly choose a point (y,m) in P;
7:      output a line segment
          ((i,y),(j − 1, y + (j − 1 − i) ∗ m));
8:      P = poly(j,j + 1); i = j; j = j + 2;
      }
    }
```

Figure 3.3: PointBound, an online algorithm for the PLA-PointBound problem.

approximate $x_i, \ldots, x_j$ such that (y_i, m) is in $poly(i, i+1) \cap \ldots \cap poly(i, j)$, where y_i is the value of s on index i, and m is the slope of s. s is output, and the intersection of polygon is removed. x_{j+1} and x_{j+2} are used to generate a new polygon $poly(j + 1, j + 2)$.

If $poly(i, i+1) \cap \ldots \cap poly(i, j) \cap poly(i, j+1) \neq \emptyset$, then the intersection is kept, and the algorithm moves on to the next point in the stream.

For any i and j, $poly(i, j)$ is a parallelogram where there are two edges parallel to the slope axis. It is easy to show that for any i and j, $\cap_{k=i}^{j} poly(i, k)$ is a convex polygon. In the worst case, the edges of the intersection of parallelograms could be up to $2(j-i+1)$, i.e., twice the number of parallelograms intersected. A straightforward method keeping all edges of the intersection area still has quadratic time complexity and linear space complexity, which are not applicable to data streams.

Fortunately, we do not need to record all edges of the intersection polygon. Instead, we need to record only up to 4 edges to determine whether a new point can be compressed together with the points seen but not compressed.

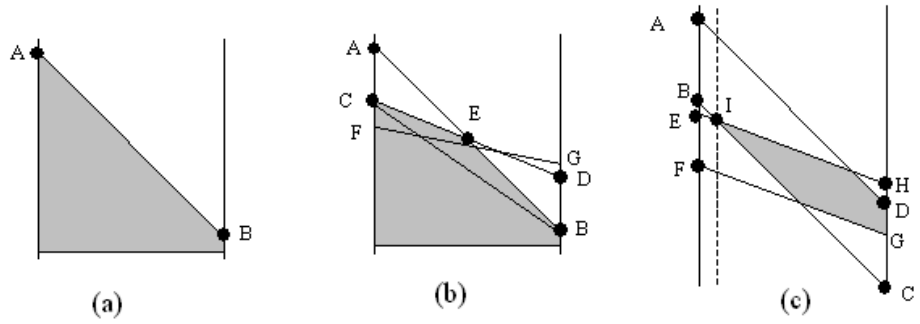

Figure 3.4: Using up to 4 edges to represent the intersection polygon.

Using Equations 3.1 to 3.2, it is easy to see that each parallelogram has two properties.

First, each parallelogram has two vertical edges and two sloping edges with a negative slope value, as shown in Figure 3.2. The range of y_i is the same for all parallelograms (i.e., $x_i - \epsilon \leq y_i \leq x_i + \epsilon$).

Second, for $j_2 > j_1 > i$, the absolute slope value of the two sloping edges in $poly(i, j_2)$ is strictly smaller than the absolute slope value of the two sloping edges in $poly(i, j_1)$.

To keep our discussion simple, let us focus on the intersection points of the upper sloping edge of parallelograms. The case for the lower sloping edges can be analyzed similarly.

The situations are illustrated in Figure 3.4. Suppose that the first parallelogram gives the upper sloping edge AB with slope value m_{AB} as in Figure 3.4(a). When a new data point arrives, a new parallelogram is formed. In the worst case, the upper sloping edge of the parallelogram CD cuts AB into two parts. Let E be the intersection point between AB and CD, as shown in Figure 3.4(b).

By the second property, we have $|m_{CD}| < |m_{AB}|$. Moreover, the upper sloping edge FG of any future parallelogram cannot cut both CE and EB due to the smaller absolute slope value of FG than m_{CD}. In other words, if a future parallelogram intersects with the current intersection polygon, the upper sloping edge of the paral-

lelogram can only cut either CE, EB or the right vertical edge. Instead of keeping CE and EB, we can keep line segment CB. Then, a future parallelogram intersects with the current intersection polygon if and only if it cuts CB.

Generally, we only need to keep the line segment connecting the left-most upper corner and the right-most upper corner for the upper sloping edges. Similarly, we only need to keep the line segment connecting the left-most lower corner and the right-most lower corner for the lower sloping edges.

In addition to this two line segments, we need to keep the two vertical edges in the intersection polygon. The reason is that the intersection of two parallelograms may shrink the range of the intersection, as illustrated in Figure 3.4(c), where parallelogram $ABCD$ intersects with parallelogram $EFGH$. The left vertical edge is shrunk into a point I right to the original edge.

In summary, we need to record only up to 4 edges to determine whether a new point can be compressed together with the points seen but not compressed. This immediately leads to the following result.

Theorem 1 (Complexity – PointBound) *The algorithm PointBound for the PLA-PointBound problem has time complexity $O(n)$ and space complexity $O(1)$, where n is the number of points in a time series to be compressed.* ■

Since algorithm PointBound only looks ahead for one point in the data stream to output a line segment whenever necessary in the piecewise linear approximation, it is an online algorithm and can be applied to data streams.

3.3 Solving the PLA-SegmentBound Problem

To tackle the PLA-SegmentBound problem, we first obtain the following useful observation.

Lemma 2 *Suppose that a line segment s approximates a fragment X of n points $x_1, \ldots, x_n$ in a time series. Then, s minimizes $segerr(s, X)$ if the slope of s is*

$$m = \frac{(2\sum_{i=1}^{n} ix_i) - \frac{2}{n}\sum_{i=1}^{n} i \sum_{i=1}^{n} x_i}{(2\sum_{i=1}^{n} i^2) - \frac{2}{n}(\sum_{i=1}^{n} i)^2} \tag{3.3}$$

and the left endpoint of s has value

$$m + \frac{\sum_{i=1}^{n}(x_i - i \cdot m)}{n}$$

Proof. Consider a line segment s approximating fragment X. Let the left endpoint of s be $(1, y_1)$ and the slope be m.

For each point x_i $(1 \leq i \leq n)$, the error is $|x_i - \tilde{x}_i| = |x_i - y_1 - m(i-1)|$. Thus,

$$segerr = \sum_{i=1}^{n}(x_i - y_1 - m(i-1))^2 \tag{3.4}$$

Clearly, when

$$y_1 = m + \frac{\sum_{i=1}^{n}(x_i - i \cdot m)}{n}$$

segerr reaches the minimum value

$$segerr = \sum_{i=1}^{n} x_i^2 + m^2 \sum_{i=1}^{n} i^2 - 2m \sum_{i=1}^{n} x_i i - \frac{(\sum_{i=1}^{n}(x_i - i * m))^2}{n} \tag{3.5}$$

From Equation (3.5), when

$$m = \frac{(\sum_{i=1}^{n} ix_i) - \frac{1}{n}\sum_{i=1}^{n} i \sum_{i=1}^{n} x_i}{(\sum_{i=1}^{n} i^2) - \frac{1}{n}(\sum_{i=1}^{n} i)^2}$$

segerr is minimized. ∎

Lemma 2 leads to algorithm SegmentBound, an online algorithm for the PLA-SegmentBound problem as shown in Figure 3.5. Suppose $x_1, \ldots, x_n$ are the points

Input: a data stream $X = x_1, x_2, \ldots$ and error-bound ϵ;
Output: a list of line segments $\tilde{X}$ approximating X
such that $maxerr(X, \tilde{X})) \leq \epsilon$;
Method:

```
1:  i = 1; j = 3
2:  s = the line segment ((1, x_1), (2, x_2));
3:  WHILE (1) DO {
4:    s' = the line segment identified in Lemma 2 to
      compress x_i, ..., x_j;
5:    IF segerr(s', x_i ··· x_j) ≤ ε THEN
6:      s = s'; j = j + 1;
7:    ELSE {
8:      output s;
9:      i = j; j = j + 2;
10:     s = the line segment ((i, x_i), (i + 1, x_{i+1}));
      }
    }
```

Figure 3.5: SegmentBound, an online algorithm for the PLA-SegmentBound problem.

that have not been compressed yet. When a new point x_{n+1} arrives, we check whether the line segment identified by Lemma 2 can achieve the segment error bound. If so, then x_{n+1} is added into the buffer, and the algorithm moves on to the next point in the stream. Otherwise, the line segment suggested by Lemma 2 for points $x_1, \ldots, x_n$ is output, and $x_1, \ldots, x_n$ are considered compressed. x_{i+n} is added into the buffer.

The time cost is constant to apply Lemma 2 to check whether a newly arrived point can be compressed together with the points that have been seen but have not been compressed. When a new data point x_{n+1} arrives, the left endpoint and the slope of the line segment suggested by Lemma 2 can be calculated quickly. Technically, Equation (3.3) indicates that we need to calculate $\sum_{i=1}^{n+1} i$, $\sum_{i=1}^{n+1} x_i$, $\sum_{i=1}^{n+1} x_i i$, and $\sum_{i=1}^{n+1} i^2$. Since we already have $\sum_{i=1}^{n} i$, $\sum_{i=1}^{n} x_i$, $\sum_{i=1}^{n} x_i i$, and $\sum_{i=1}^{n} i^2$, the addition of the new point only incurs a constant cost to update the values of m and the left endpoint.

Since each point in the data stream is processed in constant time, and we only maintain the values of $\sum_{i=1}^{n+1} i$, $\sum_{i=1}^{n+1} x_i$, $\sum_{i=1}^{n+1} x_i i$, and $\sum_{i=1}^{n+1} i^2$, we have the following claim.

Theorem 2 (Complexity – SegmentBound) *The algorithm SegmentBound for the PLA-SegmentBound problem has the time complexity $O(n)$ and space complexity $O(1)$, where n is the number of points in a time series to be compressed.* ■

Similar to algorithm PointBound, algorithm SegmentBound only looks ahead for one point in the data stream to output a line segment whenever necessary in the piecewise linear approximation. Thus, it is an online algorithm and can be applied on data streams.

Chapter 4

Optimality

In this chapter, we address the quality of algorithms PointBound and SegmentBound. Recall that, as defined in Chapter 2, in the error-bounded PLA problem, we want to minimize the number of segments used to compress a time series.

Theorem 3 (PLA-PointBound quality) *The PointBound algorithm in Section 3.2 produces a minimum number of segments to compress a time series.*

Proof. For a time series $X = x_1, \ldots, x_n$, let $l = \min\{|\tilde{X}|\}$, where $\tilde{X}$ is an ϵ-PLA approximating X (i.e., $maxerr(X, \tilde{X}) \leq \epsilon$). We conduct an induction on l to show that algorithm PointBound outputs an ϵ-PLA of l line segments.

(Base case) Consider $l = 1$, i.e., there exists a line segment that approximates the whole time series. According to Lemma 1, $poly(1,2) \cap \cdots \cap poly(1,n) \neq \emptyset$. Thus, algorithm PointBound finds a line segment s approximating $x_1, \ldots, x_n$ and $maxerr(s, X) \leq \epsilon$.

(Induction) Assume that, when $l \leq k$, algorithm PointBound finds an ϵ-PLA $\tilde{X}$ of l line segments to approximate X. Now, consider the case of $l = (k+1)$, i.e., there exists an optimal ϵ-PLA $\tilde{Y} = \{s_1, \ldots, s_{k+1}\}$ that approximates X.

Suppose that s_1 approximates $x_1, \ldots, x_m$. Let us assume that s_1' output by algorithm PointBound approximates points $x_1, \ldots, x_{m'}$. Due to Lemma 1, $poly(1,2) \cap \cdots \cap poly(1,m) \neq \emptyset$. Thus, s_1' must approximate $x_1, \ldots, x_m$ with the quality guarantee,

i.e., $maxerr(s'_1, x_1 \cdots x_m) \leq \epsilon$. In other words, $m' \geq m$.

If $m = m'$, then points $x_{m+1}, \ldots, x_n$ in X can be approximated by an ϵ-PLA of $(l-1) = k$ line segments. According to the assumption, algorithm PointBound finds an ϵ-PLA of $(l-1)$ line segments approximating $x_{m+1}, \ldots, x_n$.

Suppose that $m' > m$. Since $x_{m+1}, \ldots, x_n$ can be approximated by an ϵ-PLA of $(l-1)$ line segments, a proper subset $x_{m'+1}, \ldots, x_n$ must also be approximated by an ϵ-PLA of at most $(l-1) = k$ line segments. We only need to drop the segments approximating $x_{m+1}, \ldots, x_{m'}$. According to the assumption, algorithm PointBound finds an ϵ-PLA of the minimum number of line segments to approximate points $x_{m'+1}, \ldots, x_n$.

In summary, algorithm PointBound finds an ϵ-PLA of $l = (k+1)$ line segments approximating X. ■

Similarly, we can also show the optimality of the SegmentBound algorithm.

Theorem 4 (PLA-SegmentBound quality) *The SegmentBound algorithm in Section 3.3 produces a minimum number of segments to compress a time series.*

Although the number of line segments used to approximate a time series is a good measure on the compression quality, it is not directly translated to compression ratio. For example, in our methods, the endpoints of segments are not constrained. Thus, two points are needed to represent a segment. On the other hand, a PLA using connecting segments (i.e., two consecutive segments share the same endpoint) may use more segments but achieve a better compression ratio since only one point is needed to represent a segment except for the first segment.

How good compression can algorithms PointBound and SegmentBound achieve? We have the following result.

Theorem 5 (Approximation factor) *Algorithms PointBound and SegmentBound*

have an approximation factor of 2 *to the optimum compression factor that an* ϵ*-PLA can achieve.*

Proof. We only show the case for the PointBound algorithm. The same argument applies to the SegmentBound algorithm.

For any time series X of m points, suppose that the PointBound algorithm approximates X using n line segments. Then, according to Theorem 3, any PLA cannot have less than n line segments. To represent n line segments, at least $(n+1)$ points are needed. Thus, the optimum compression ratio using PLA is at most $\alpha_{opt} = \frac{m}{n+1}$.

The line segments generated by the PointBound algorithm may not be connecting. Thus, at most $2n$ points are needed to represent the n line segments. The worst case compression ratio of the PointBound algorithm is $\alpha_{PointBound} = \frac{m}{2n}$.

Clearly, $\frac{\alpha_{opt}}{\alpha_{PointBound}} = \frac{2n}{n+1} < 2$. ■

Chapter 5

PLAZA for Tiny Sensors

Although algorithm PointBound is optimal for the PLA-PointBound problem, it still may be too computation intensive for tiny, resource-constrained sensors due to two reasons.

First, algorithm PointBound may generate non-connecting segments such that each segment requires the transmission of two endpoints. As analyzed before, connecting line segments reduce the data transmission volume since each segment (except the first one) requires the transmission of only one endpoint. Second, algorithm PointBound has to calculate intersection of parallelograms. The computation may be too heavy for tiny, resource-constrained sensor nodes.

In this chapter, we design a simple, fast online algorithm PLAZA (Piecewise Linear Approximation with Zoning Angle) for the PLA-PointBound problem. PLAZA generates connecting line segments. Although PLAZA is not optimal in the number of line segments used for approximation, it is light in computation and very effective in compression ratio, as will be verified by our experiments.

5.1 PLAZA

PLAZA builds on the concept of zoning angle. Given an error bound ϵ and two points (i, x_i) and (k, x_k) $(i < k)$, the *zoning angle* from (i, x_i) to (k, x_k), denoted by $\theta^{\epsilon}_{(i,k)}$, is defined as the angle that has (i, x_i) as the endpoint, $((i, x_i), (k, x_k))$ as the bisector,

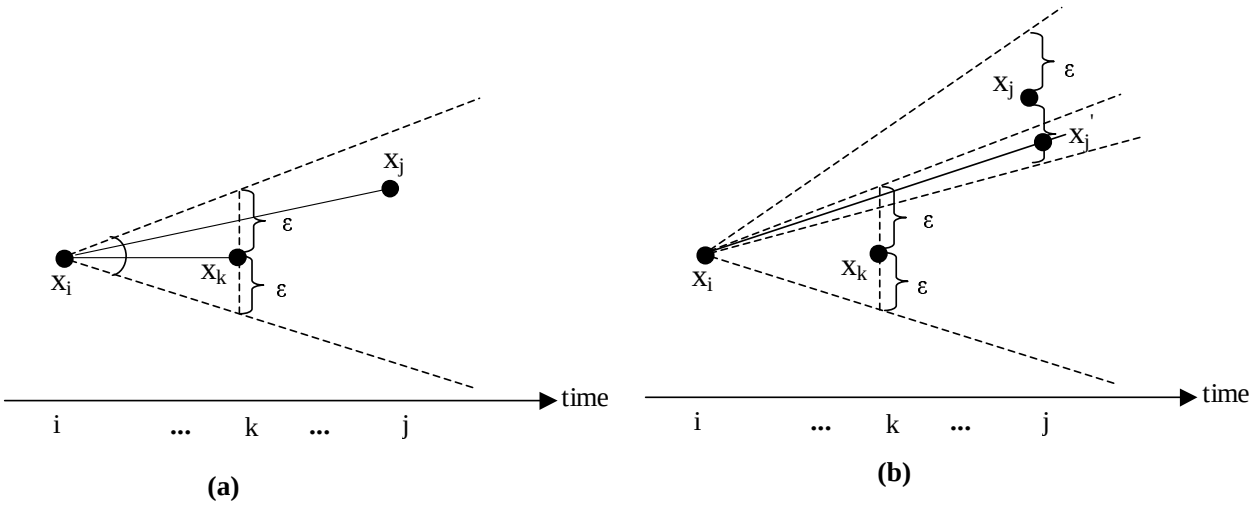

Figure 5.1: An example of zoning angle

and has a degree of $2\arctan\frac{\epsilon}{|x_i x_k|}$, where $|x_i x_k| = \sqrt{(k-i)^2 + (x_k - x_i)^2}$.

Figure 5.1(a) shows an example of zoning angle $\theta^{\epsilon}_{(i,k)}$. The zoning angle defines a zone to include any potential line segments that can be used to compress x_i and x_k.

We observe the following important results.

Lemma 3 *For three points $x_i, x_k, x_j (i < k < j)$ in a time series, the line segment $((i, x_i), (j, x_j))$ approximates x_k with error up to ϵ if and only if the line segment $((i, x_i), (j, x_j))$ falls in the zoning angle $\theta^{\epsilon}_{(i,k)}$.*

Proof: (Necessity) Without loss of generality, we assume that (j, x_j) is above the line $((i, x_i), (k, x_k))$, as shown in Figure 5.1. If $((i, x_i), (j, x_j))$ approximates x_k with error up to ϵ, the vertical distance between point (k, x_k) and line $((i, x_i), (j, x_j))$ must be smaller than ϵ. That is, the line segment $((i, x_i), (j, x_j))$ must fall in the zoning angle $\theta^{\epsilon}_{(i,k)}$.

(Sufficiency) If the line segment $((i, x_i), (j, x_j))$ falls in the zoning angle $\theta^{\epsilon}_{(i,k)}$, the degree of angle $\angle x_j x_i x_k$ is smaller than $\arctan\frac{\epsilon}{|x_i x_k|}$. Therefore, the vertical distance between point (k, x_k) and line $((i, x_i), (j, x_j))$ is smaller than ϵ, which means, line segment $((i, x_i), (j, x_j))$ approximates x_k with error up to ϵ. ■

Input: a data stream $X = x_1, x_2, \ldots$ and error-bound ϵ;
Output: an ϵ-PLA $\tilde{X}$ of a list of *connecting* line segments, i.e., $maxerr(X, \tilde{X})) \leq \epsilon$;
Method:

```
1:  i = 1; angle = θ^ε_(1,2);
2:  s = line segment ((1, x_1), (2, x_2)); j = 3;
3:  WHILE (1) DO {
4:    angle = angle ∩ θ^ε_(i,j);
5:    IF angle ≠ 0 THEN {
6:      IF segment ((i, x_i), (j, x_j)) falls in angle
7:      THEN s = line segment ((i, x_i), (j, x_j));
8:      ELSE {
9:        x'_j = the value of the bisector line of
          angle at index j as shown in Figure 5.1(b);
10:       s = the line segment ((i, x_i), (j, x'_j));
11:       x_j = x'_j;
12:     }
13:     j = j + 1;
14:   }
15:   ELSE {
16:     output s;
17:     i = j − 1; x_i = x_{j−1}; j = j + 1;
18:     angle = θ^ε_(i,i+1);
19:     s = line segment ((i, x_i), (i + 1, x_{i+1}));
20:   }
21: }
```

Figure 5.2: Algorithm PLAZA.

Lemma 4 *For three points $x_i, x_k, x_j (i < k < j)$ in a time series, if zoning angle $\theta^{\epsilon}_{(i,j)}$ has no overlap with zoning angle $\theta^{\epsilon}_{(i,k)}$, there does not exist a line segment s with (i, x_i) as the left endpoint such that $maxerr(s, x_i \cdots x_k \cdots x_j) \leq \epsilon$.*

Proof: We prove by contradiction. Assume that there is a line segment s with (i, x_i) as the left endpoint such that s approximates both x_k and x_j with an error up to ϵ. Based on Lemma 3, this line segment must fall in both $\theta^{\epsilon}_{(i,k)}$ and $\theta^{\epsilon}_{(i,j)}$, which means $\theta^{\epsilon}_{(i,k)}$ and $\theta^{\epsilon}_{(i,j)}$ overlap. Contradiction. ■.

Algorithm PLAZA works as follows. Starting from a point x_i, Lemma 3 is used to

check if there is a line segment approximating points between indexes i and $j(i < j)$. Moreover, Lemma 4 is used to check if searching further in the time series is futile. The pseudocode of PLAZA is shown in Figure 5.2.

Algorithm PLAZA scans each point in a data stream only once and stores only the zoning angle and the current approximating segment in main memory, the algorithm clearly has linear time complexity and constant space complexity.

Compared to algorithm PointBound, PLAZA requires much simpler computation. The major operation is to construct the zoning angle and to calculate the intersection of zoning angles. The exact degree of each angle is not required. Instead, we only need to record the endpoint and the two edges of the angle. Similar to algorithm PointBound, PLAZA returns endpoints of segments that may not necessarily belong to the original time series (i.e., the value of x_j may be changed as shown in line 11 in the pseudo code).

5.2 Benchmarking PLAZA

PLAZA creates connecting line segments. Only transmission of one point is needed for each line segment except for the first line segment. This feature distinguishes PLAZA from algorithms PointBound and SegmentBound. What is the optimal compression that can be achieved by an ϵ-PLA consisting of only connecting line segments? In this subsection, we give such an optimal algorithm, optimal PLAZA benchmark.

The idea behind the optimal PLAZA benchmark algorithm is similar to that of algorithm PointBound. The main difference is that, unlike the PointBound algorithm, we do not start the new segment with the initial condition $x_i - \epsilon \leq y_i \leq x_i + \epsilon$, where y_i is the value of the left endpoint of the new segment. Instead we set a smaller range on y_i to guarantee the connectivity of two consecutive segments. Specifically, to decide the range of y_i, we use the last non-empty polygon intersection in the previous point.

Suppose points $x_1, \ldots, x_{j-1}$ are checked but have not been compressed yet, i.e., $poly(1,2) \cap \cdots \cap poly(1, j-1) \neq \emptyset$. We keep the exact intersection of the parallelograms. The intersection is used to confine the range of y_j. Apparently, as the intersection is a convex polygon, only the corners of the polygon need to be checked when looking for the minimum and maximum values. This fact helps the search of y_j.

We find the optimal solution by a exhaustive search. Starting from x_1, we try all values of j such that $x_1, \ldots, x_j$ can be approximated by a line segment with maximal error ϵ. For each such a subset $x_1, \ldots, x_j$, we compute the intersection of parallelograms $poly(1,2) \cap \cdots \cap poly(1, j)$, and try to find a line segment with left endpoint (j, y_j) that can approximate some points $x_{j+1}, \ldots, x_i$ where $j + 1 < i$ and y_j is in the range confined by $poly(1,2) \cap \cdots \cap poly(1, j)$. By doing so, the first and the second line segments are connecting. We conduct a depth-first search to find an ϵ-PLA consisting of the minimum number of connecting line segments.

We search for the optimal solution as follows. Starting from x_1, we add new points step by step until the intersection of polygons becomes empty. We then use the last non-empty intersection polygon to define the initial range of y_i and start a new search. The above procedure continues until all points are covered. Since a new search depends on the previous endpoint, we need to test all possible segmentation scenarios to obtain the optimal solution. To do so, we perform back-tracking search (i.e., we perform the above search from all intermediate points along a segment.). The segmentation scenario with the minimum number of connecting segments is returned as the optimal solution. The back-tracking search is done with the recursive function defined in Figure 5.3. To obtain the optimal solution for the whole time series, we execute the recursive function with the start point as x_1.

The optimal PLAZA benchmark is an offline algorithm: it assumes the time series is given and can be scanned multiple times. Its complexity is far above linear due to

Input: a data stream $X = x_1, x_2, \ldots, x_n$ and error-bound ϵ;
Output: the minimum number of *connecting* segments
approximating X within point error-bound ϵ;
Function: PLAZABenchmark() {
1: return $recursiveBacktracking(1, x_1 + \epsilon, x_1 - \epsilon)$;
}

Function: $recursiveBacktracking(index, y_i^{max}, y_i^{min})$ {
2: $minSegments = \infty$;
3: `IF` $(index == n)$ `THEN` return 1;
4: $end = index$; $P = \infty$;
5: `WHILE` $(P \neq \emptyset)$ `DO` {
6: `IF` $(end - index > 0)$ `THEN` {
7: update y_i^{max} and y_i^{min} based on P;
8: $segNum = 1 + recursiveBacktracking(end, y_i^{max}, y_i^{min})$;
9: $miniSegments = min(miniSegments, segNum)$;
}
10: $end = end + 1$;
11: $P' =$ polygon defined by y_i^{max}, y_i^{min}
and slopes calculated with Equations **??** to 3.2;
12: $P = P \cap P'$;
}
13: return $miniSegments$;
}

Figure 5.3: Optimal PLAZA benchmark algorithm

the exhaustive search. This algorithm is obviously not suitable for online compression of data streams. It is for comparison purpose only.

Chapter 6

Multidimensional PLA Problems

6.1 Problem Formulation

In this section, we take a step forward to investigate the PLA problems in multidimensional space, where each point of the time series consists of an array of values. One example application of multidimensional PLA problem is to use line segments to approximate the trace of a moving object in the three-dimensional space. In this case, the endpoints of line segments are points in the three-dimensional space.

Let X be a time series of n points and $x_i (1 \leq i \leq n)$ be the value of the i-th point of X in a multidimensional space. Assume that the dimension of the space is M. We denote each point x_i as $(x_i^1, x_i^2, \ldots, x_i^M)$. Let $\tilde{X}$ be a PLA of X. We revise the error measurements *maxerr* and *segerr* in Chapter 2 to the M-dimensional case and denote them as $Mmaxerr$ and $Msegerr$, respectively.

$$Mmaxerr(X, \tilde{X}) = \max_{i=1}^{n}\{|x_i - \tilde{x}_i|\} \tag{6.1}$$

where $|x_i - \tilde{x}_i| = \sqrt{\sum_{j=1}^{M}(x_i^j - \tilde{x}_i^j)^2}$.

$$Msegerr(X, \tilde{X}) = \max_{s \in \tilde{X}}\{\sum_{i \in range(s)}(x_i - \tilde{x}_i)^2\} \tag{6.2}$$

where $(x_i - \tilde{x}_i)^2 = \sum_{j=1}^{M}(x_i^j - \tilde{x}_i^j)^2$.

Similar to the PLA-PointBound problem and the PLA-SegmentBound problem, the multidimensional PLA problems could be defined as follows.

Problem 5 (Multidimensional PLA-PointBound problem) *Given an error-bound ϵ, the* Multidimensional PLA-PointBound problem *is to find an ϵ-PLA $\tilde{X}$ such that $Mmaxerr(X, \tilde{X}) \leq \epsilon$ and $|\tilde{X}|$ is minimized.* ■

Problem 6 (Multidimensional PLA-SegmentBound problem) *Given an error-bound ϵ, the* Multidimensional PLA-SegmentBound problem *is to find an ϵ-PLA $\tilde{X}$ such that $Msegerr(X, \tilde{X}) \leq \epsilon$ and $|\tilde{X}|$ is minimized.* ■

6.2 Multidimensional PLA-SegmentBound Problem

We have shown in Lemma 2 that in one-dimensional case we can easily find a line segment s that minimizes $segerr(s, X)$ for a time series X of n points. With this Lemma, we can solve an M-dimensional PLA-SegmentBound problem by breaking it to M one-dimensional problems.

Problem 7 (k-th PLA-SegmentBound problem) *Given M-dimensional PLA-SegmentBound problem with $Msegerr$ of ϵ, the k-th PLA-SegmentBound problem $(1 \leq k \leq M)$ is defined as the one-dimensional PLA-SegmentBound problem in the k-th dimension with segerr of ϵ_k, where $\epsilon = \sum_{k=1}^{M} \epsilon_k^2$.*

Algorithm M-SegmentBound to solve the multidimensional PLA-SegmentBound problem is shown in Figure 6.1. The basic idea of this algorithm is to calculate the minimum *segerr* in each individual dimension based on Lemma 2, and then check if $Msegerr$ calculated with the *segerr* values is within the given bound. If it is, we can move ahead to check the next point without starting a new line segment. Otherwise, we output the current segment and start a new line segment. According to Lemma 2,

Input: a data stream $X = x_1, x_2, \ldots$ in an M-dimensional space and error-bound ϵ;
Output: a list of line segments $\tilde{X}$ approximating X in the M-dimensional space such that $Msegerr(X, \tilde{X})) \leq \epsilon$;
Method:

```
1:  i = 1; j = 3
2:  s = the line segment ((1, x_1), (2, x_2));
3:  WHILE (1) DO {
4:    FOR (d = 1; d++; d ≤ M) DO
5:      s'_d = the line segment in the d-th dimension
        identified in Lemma 2 to compress x_i^d, ..., x_j^d;
6:    IF Σ_{d=1}^{M} segerr(s'_d, x_i^d ··· x_j^d) ≤ ε THEN
7:      Construct line segment s' in the M-dimensional
        space with s'_1, ..., s'_M;
8:      s = s'; j = j + 1;
9:    ELSE {
10:     output s;
11:     i = j; j = j + 2;
12:     s = the line segment ((i, x_i), (i + 1, x_{i+1}));
      }
    }
```

Figure 6.1: M-SegmentBound, an online algorithm for the multidimensional PLA-SegmentBound problem.

segerr is minimized at each dimension. Hence the $Msegerr$ obtained with the *segerr* values is also minimized.

We should explain the operation in Line (7) of the algorithm. A line segment in an M-dimensional space is determined by the two endpoints in the M-dimensional space. For the first (start) point, the value of its k-th dimension is the value of the start point of (one-dimensional) line segment s'_k; and similarly the value of the second (end) point's k-th dimension is the value of the end point of (one-dimensional) line segment s'_k, where $k = 1, \ldots, M$. It is easy to see that

Theorem 6 (Complexity – M-SegmentBound) *The M-SegmentBound algorithm for the M-dimensional PLA-SegmentBound problem has the time complexity $O(Mn)$ and space complexity $O(1)$, where n is the number of points in a time series to be*

compressed. ■

Theorem 7 (Optimality – M-SegmentBound) *The M-SegmentBound algorithm for the M-dimensional PLA-SegmentBound problem produces a minimum number of segments to compress a time series in the M-dimensional space .*

The optimality comes from the fact that the M-SegmentBound algorithm works in the same way of the SegmentBound algorithm. More specifically, we can calculate the exact value of the smallest $MSegerr$ with Equation 6.2 and check if it is within the given bound (Line (6) of the algorithm).

6.3 Multidimensional PLA-PointBound Problem

Unfortunately, the above idea cannot be used to extend the PointBound algorithm (3.2) to solve the multidimensional PLA-PointBound problem. The main difference is that the PointBound algorithm checks the existence of approximating line segment in one-dimensional space, and as long as such line segments exist, the PointBound algorithm will be able to find one but does not guarantee the approximating error is minimum. Due to this reason, we cannot use Equation 6.1 to obtain the minimum $Mmaxerr$ and thus we cannot simply check whether we should output current line segments or move ahead to another new point in the multidimensional case. Designing optimal algorithm for the multidimensional PLA-PointBound problem is left open for our future work.

In this section, we extend algorithm PLAZA to obtain approximate solution for the multidimensional PLA-PointBound problem. The main idea is to adjust the error bound of each individual dimension as long as the error $Mmaxerr$ calculated with Equation 6.1 is within the given bound. Initially, we equally assign the allowed error bound to each dimension.

With the initial error bound of each dimension, we perform algorithm PLAZA on each dimension. Assume that at some point, algorithm PLAZA needs to stop

and output the current segment on the k-th dimension ($1 \leq k \leq M$). We adjust the error bounds of each dimension within the constraint of $Mmaxerr \leq \epsilon$ to see if further search ahead is possible. Specifically, we increase the error bound of k-th dimension and reduce the error bounds on other dimension so that algorithm PLAZA can move forward on all dimensions. If the adjustment of error bounds among different dimensions is impossible, we output current segment and start a new segment with initial error bound on each dimension equally assigned. It is worth noting that the error bound adjustment at a point only impacts on the current point, because we only consider the overlapping area of zoning angles. That is, all previous points still fall within $Mmaxerror$ bounds if approximated using a line segment chosen based on the overlapping angle in each dimension.

There are certainly many ways to adjust error bounds among different dimensions. We have explored different strategies for the error bound adjustment and find the adjustment method presented in Figure 6.2 is very easy to implement. That method can effectively prevent early output of a line segment whenever error bound adjustment is still feasible.

Input: Error bound on each dimension $\epsilon_1, \ldots, \epsilon_M$;
Current point;
Output: TRUE(adjustment successful) or FALSE (otherwise)
Method:
1: Sort the bounds $\epsilon_1, \ldots, \epsilon_M$ in increasing order;
2: FOR $(i = 1; i++; i \leq M - 1)$ DO {
3: IF (the i-th dimension of current point cannot be approximated with error bound ϵ_i) {
4: Increase ϵ_i by ϵ_δ, where ϵ_δ is the minimum value so that on i-th dimension, the current point can be compressed together with previously uncompressed points with the increased error bound.
5: Decrease ϵ_{i+1} by ϵ_δ so that $Mmaxerr$ remains unchanged;
}
}
6: IF $(\epsilon_M > 0)$ return TRUE; else return FALSE;

Figure 6.2: ErrorBoundAdjustment, the algorithm to adjust error bound among different dimensions for the PLA-PointBound problem.

Chapter 7

Experimental Evaluation

In this chapter, we evaluate the performance of our online algorithms by simulation in Matlab and by real implementation with MICA2 motes [16].

7.1 Existing Work

The work in [19, 23] is related to our online algorithms. After much deliberation, however, we find it is impossible to get fair comparison between our online algorithms and those methods due to the following reasons.

First, the SWAB algorithm in [19] uses a moving window to constrain the time period under consideration. Its performance largely depends on the size of the moving window. As such, SWAB is largely different from our methods that do not maintain any window. Furthermore, it is hard to allocate enough space for the moving window in space-limited tiny sensors.

Second, the algorithms in [23] use an amnesic function to give weights to different points in the time series. The focus and the application context are different from ours.

Due to the above consideration, in the rest of this section, we focus on the implementation and the evaluation of our online algorithms.

7.2 Experimental Setting

We generated two audio files for test. The first file includes human voice with the sampling rate of 8 khz in mono channel. The second file includes piano music with the sampling rate of 44 khz in mono channel. Each file includes $1,000,000$ samples, and the size of each sample is 16 bits. Figures 7.1 and 7.2 show the waveform of the human voice data and the waveform of the piano music, respectively.

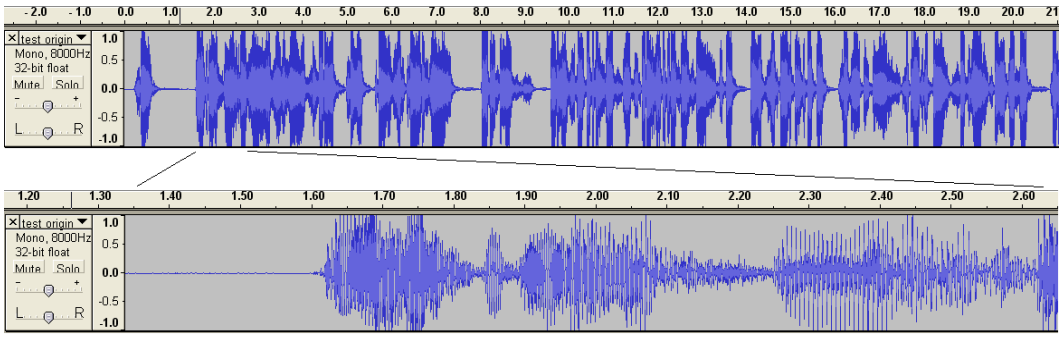

Figure 7.1: The waveform of the human voice data (the lower part is in a smaller time scale).

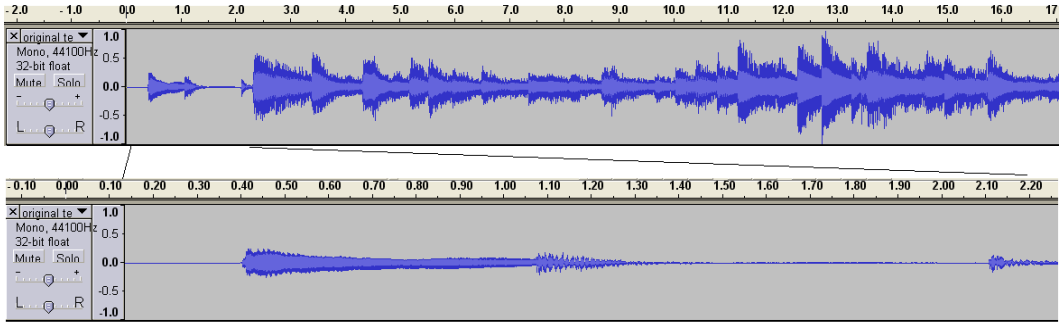

Figure 7.2: The waveform of the piano music data (the lower part is in a smaller time scale).

We use the files to test the performance of our online algorithms in bandwidth saving. We measure two metrics:

1. *Sample reduction ratio (inverted compression ratio).* It is defined as the total

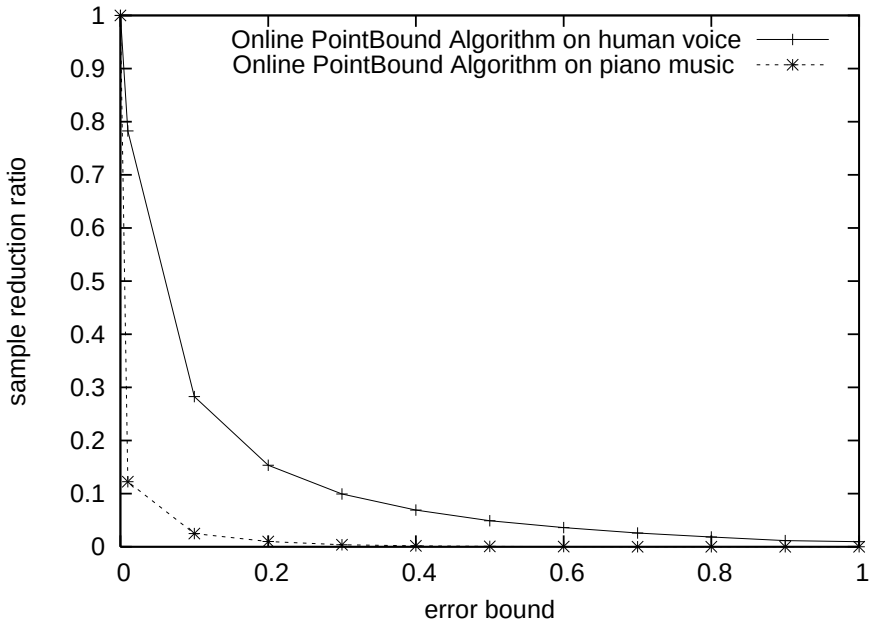

Figure 7.3: The sample reduction ratio of PointBound with respect to error bounds.

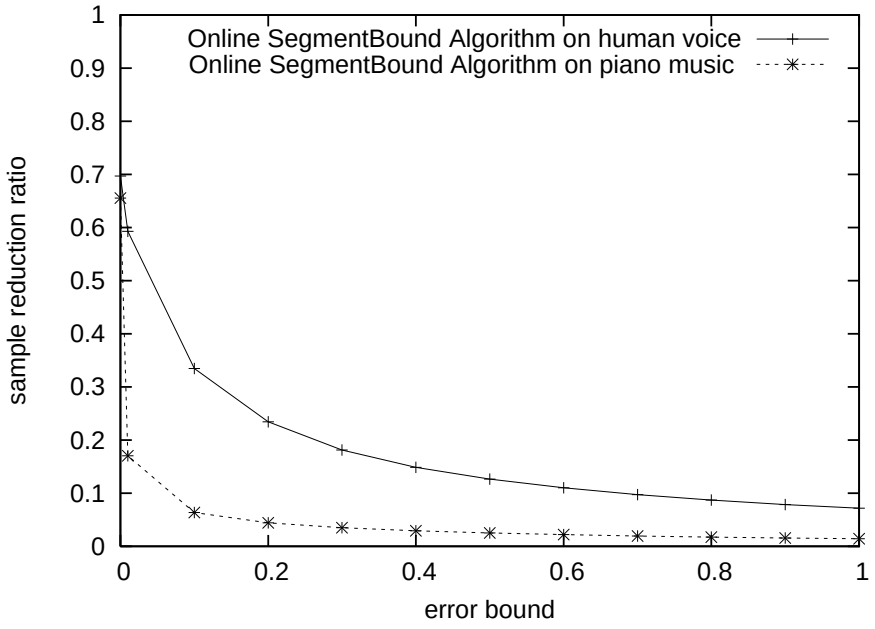

Figure 7.4: The sample reduction ratio of Segment-Bound with respect to error bounds.

number of points to represent the ϵ-PLA divided by the total number of points in the original time series.

2. *Distortion.* It is defined as $\frac{\sum_{i=1}^{n}(x_i-\tilde{x_i})^2}{n}$, where n is the total number of points in time series, x_i is the original value, and $\tilde{x_i}$ is the approximated value of x_i.

In simulation, we apply the online algorithms on the audio files and measure the sample reduction ratio. Simulation results are reported in Section 7.3 and Section 7.4. In the test using MICA2 motes, the original audio files are played on a desktop computer and are monitored and transmitted with a MICA2 mote over a wireless channel to a laptop computer. More details are provided in Section 7.5.

7.3 Results on Quality

In this section, we test the sample reduction ratio and distortion using the two audio data sets.

7.3.1 Results on Sample Reduction Ratio

Figures 7.3 and 7.4 show the results of algorithms PointBound and SegmentBound, respectively, with respect to various error bound values. As shown in the figures, we can obtain a higher bandwidth saving on piano music than on human voice. By replaying the audio files recovered from the samples by our algorithms, we perceive that the human voice recovered from the samples by our algorithms is fully recognizable with the segment error bound up to 0.4, or with the point error bound up to 0.2. The quality of recovered piano music is acceptable to us with the segment error bound up to 0.2, or with the point error bound up to 0.1.

Figures 7.3 and 7.4 clearly demonstrate significant bandwidth saving. With the online algorithms, we only need to transmit around 5% of the original sample size for piano music and around 20% of the original sample size for human voice. As such, both sound files can be transmitted with the current sensor nodes.

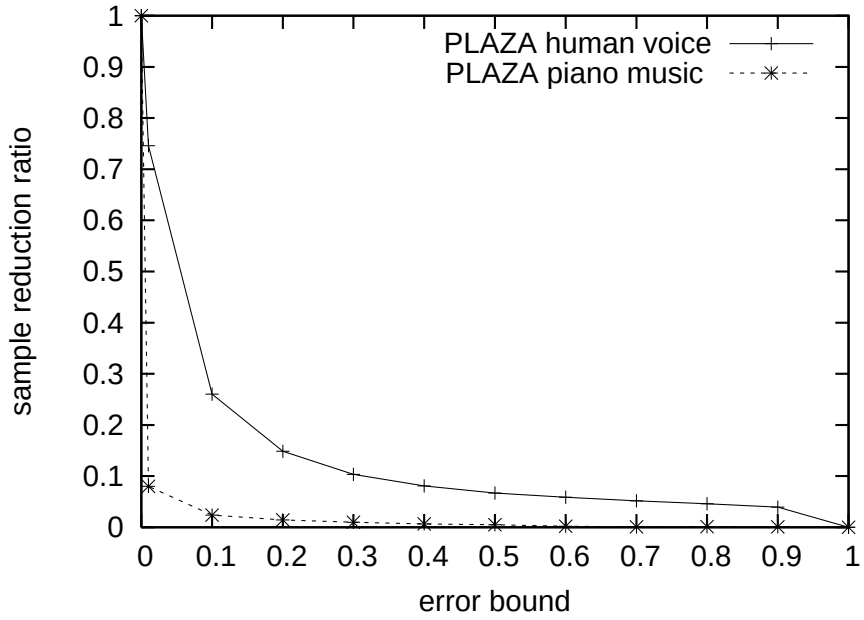

Figure 7.5: The sample reduction ratio of PLAZA with respect to error bounds.

Figure 7.5 shows the sample reduction ratio of algorithm PLAZA with respect to various point error bounds. We observe similar phenomenon as in Figures 7.3 and 7.4. With PLAZA, we perceive that the recovered human voice is fully recognizable with an (point) error bound up to 0.2, and the quality of recovered piano music is acceptable to us with the (point) error bound up to 0.1. From Figure 7.5, the above qualities correspond to a bandwidth reduction of nearly 3% of the original data size for piano music and about 15% of the original data size for human voice.

One interesting phenomenon is that the SegmentBound algorithm can reduce sample transmission volume even if the error bound is set to zero, as shown in Figure 7.4. This is because in the audio files, there are some silent periods where the sample values are all zeros. The SegmentBound algorithm finds a line segment to approximate those situations. This nice feature, however, does not exist in the algorithms for the PLA-PointBound problem. If the error bound is zero, the initial polygon is empty in the PointBound algorithm, and the degree of the initial feasible angle is zero in

PLAZA, resulting in no sample reduction.

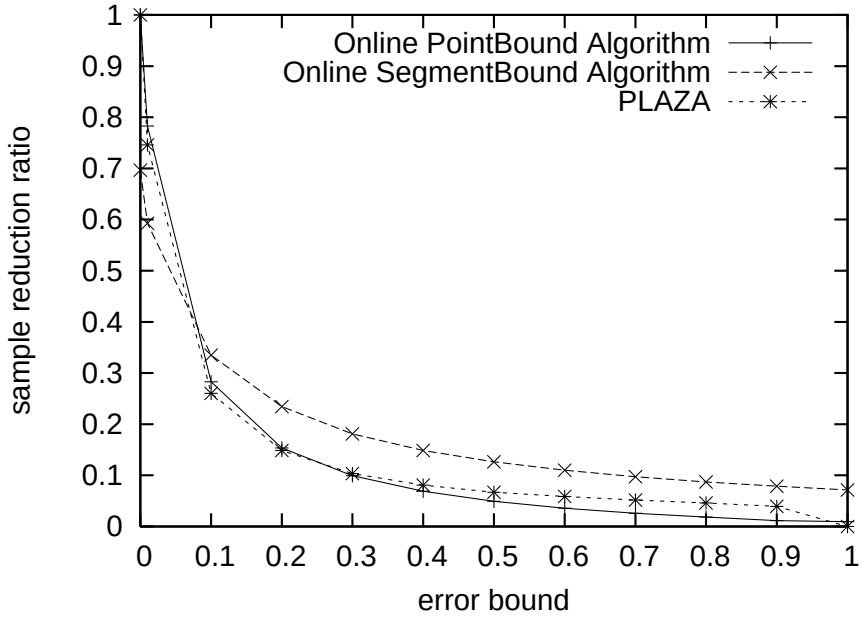

Figure 7.6: Comparison of the three algorithms on the human voice data set.

Figure 7.6 compares algorithms PLAZA, PointBound, and SegmentBound on the human voice data set. The gap between algorithms PLAZA and PointBound is very small when the error bound is less than 0.5. Algorithm PointBound leads to more samples than algorithm PLAZA when the error bound is less than 0.3. The gap between algorithm SegmentBound and the two algorithms for the PLA-PoinBound problem comes from the fact that, using the same error bound value, the PLA-SegmentBound problem puts a tighter error constraint than the PLA-PointBound problem. We observe the similar performance comparison of the three algorithms on the piano data set.

7.3.2 Results on Distortion

In Figures 7.7 and 7.8, we quantitatively show the distortion of our algorithms on the human voice data set and the piano music data set, respectively. The overall distortion on human voice is larger than that on piano music. With the same error

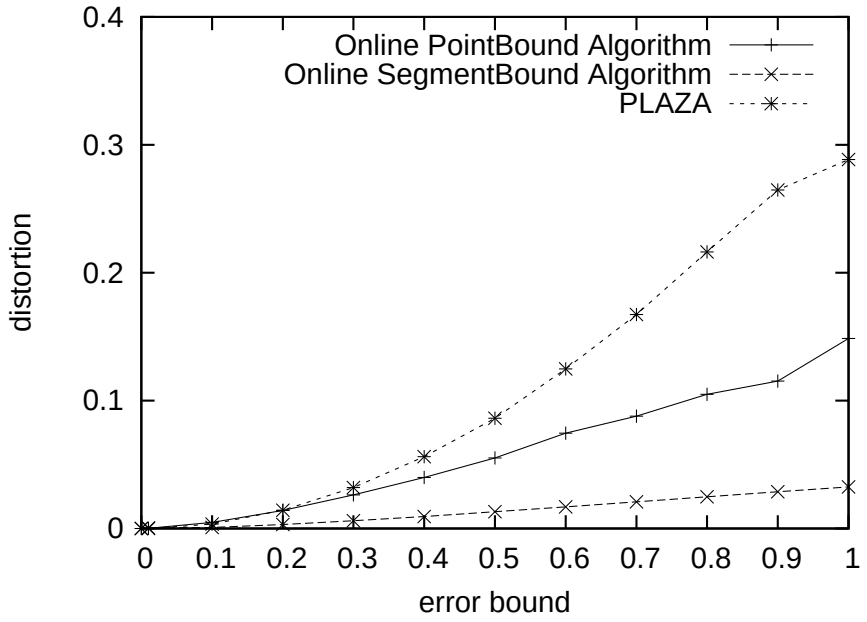

Figure 7.7: The distortion on the human voice dataset.

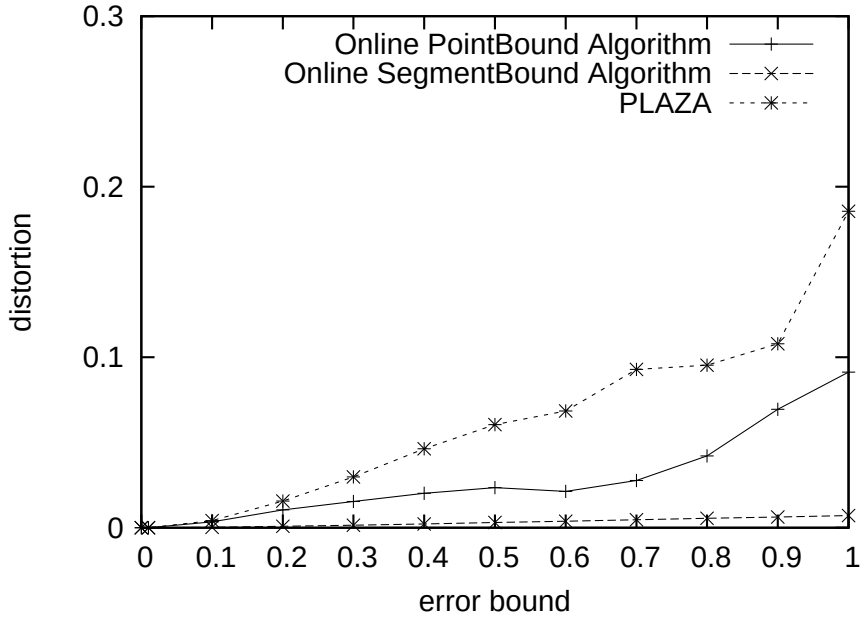

Figure 7.8: The distortion on the piano music voice dataset.

bound, algorithm PLAZA has the largest distortion. Algorithm PointBound is the next. Algorithm SegmentBound has the smallest distortion because the same error bound on the PLA-SegmentBound problem and the PLA-PointBound problem poses a tighter error constraint on the PLA-SegmentBound problem. The smaller distortion, however, comes with the cost of lower bandwidth saving as analyzed before. This is the tradeoff between bandwidth saving and quality of recovery.

7.4 Benchmarking PLAZA

We test the performance of PLAZA comparing to the optimal solution of its kind (i.e., using connecting line segments to tackle the PLA-PointBound problem). Due to the high complexity of the PLAZA Benchmark method, the audio files are too big to obtain the optimal results within reasonable time. We have to use a small portion of the audio files for this test.

Interestingly, the PLAZA method and the optimal PLAZA benchmark algorithm generate very similar PLA line segments. The audio files have many silent gaps where sample values are close to 0. Thus, algorithm PLAZA can obtain line segments very similar to those computed by the benchmark algorithm.

To further test algorithm PLAZA with more difficult scenarios, we artificially generated some data sets containing Gaussian noise with the mean equal to 0 and the variance equal to 1. We generate 5 Gaussian noise data sets. Each data set has $1,000$ samples. We run the simulation on the 5 data sets and calculate the average sample reduction ratio as the final result.

Figure 7.9 compares algorithms PLAZA, PointBound, and the PLAZA benchmark. Since algorithm PointBound may generate disconnected line segments, it has to send two endpoints for each line segments. In contrast, algorithm PLAZA and the PLAZA benchmark method always generate connecting segments, where every segment, except for the first one, requires the transmission of only one endpoint. The

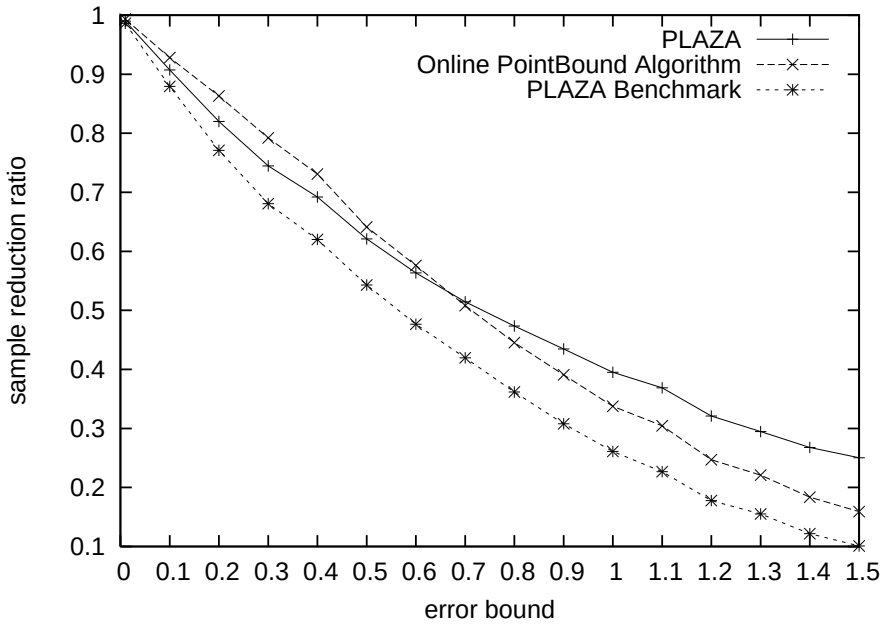

Figure 7.9: The comparison between algorithm PLAZA and the PLAZA benchmark method.

figure shows that algorithm PointBound actually needs more endpoints than algorithm PLAZA when the error bound is small, even though algorithm PointBound is optimal in terms of the number of segments. Compared to the PLAZA benchmark algorithm, algorithm PLAZA always generates more samples. The gap between algorithm PLAZA and the PLAZA benchmark method, however, is not significant when the error bound is small.

7.5 Results on Real Sensors

We implemented our online algorithms using MICA2 motes [16] from Crossbow Technology Inc. The test bed is illustrated in Figure 7.10.

A MICA2 mote includes a radio/processor board and a sensor board. The radio/processor board uses 900 Mhz radio. The sensor board includes a microphone that can be used for sampling sound. The interface of the base station is based on RS232. It is used to program the mote and in the mean time acts as a gateway to connect the laptop and the radio wireless sensor network. Our laptop does not have

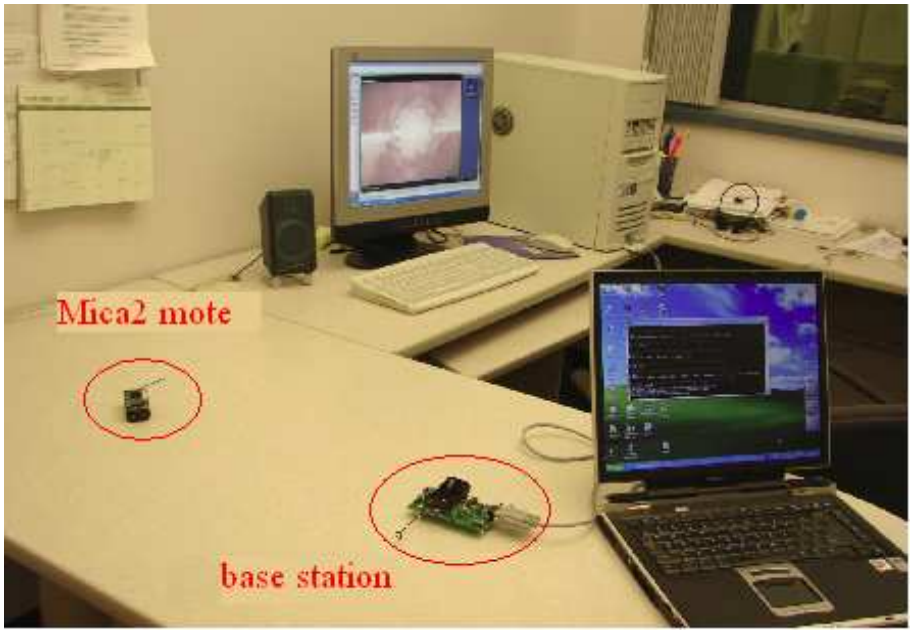

Figure 7.10: The test bed using real sensors.

an RS232 port and thus we use a USB/RS232 adaptor to connect to the base station interface. The original audio files are played on a desktop computer, monitored by a MICA2 mote, and transmitted over wireless channel from the MICA2 mote to the base station.

The sample reduction ratio on the real sensor test bed is close to the simulation results using Matlab. But the audio quality obtained using the real test bed is worse than that obtained in the Matlab simulation. The deterioration in audio quality is caused by the major restriction of TinyOS [6], the current operating system in MICA2 motes. The OS does not support multiple threads and thus it cannot perform radio transmission and sound sampling concurrently. Due to this limit, when we transmit data to the base station, the sensor board stops sampling and the sound during this period is missed, resulting in small silent gaps in the recovered audio.

The same task can be carried out with the most recent, more advanced sensor device, MICAz from the same company. With a higher price, MICAz sensors support up to 250 Kbps wireless transmission. This task, however, has never been fulfilled

with low-end devices like MICA2. To this end, we break the limit of scarce radio bandwidth and carry out a task that is hard to achieve without the mentioned fast online compression methods.

7.6 Evaluation in Other Applications

Although we only implemented the online algorithms in an acoustic sensor monitoring system, our algorithms are actually applicable to many other application domains such as electrocardiogram (ECG) monitoring for patients. We test our algorithm on an ECG data set obtained from http://www.cs.ucr.edu/~Eeamonn/TSDMA/datasets.html.

The maximum value on the data set is $2,490$ and the minimum value is $-8,190$. We test our online algorithms with error bound varying from 1 to 100.

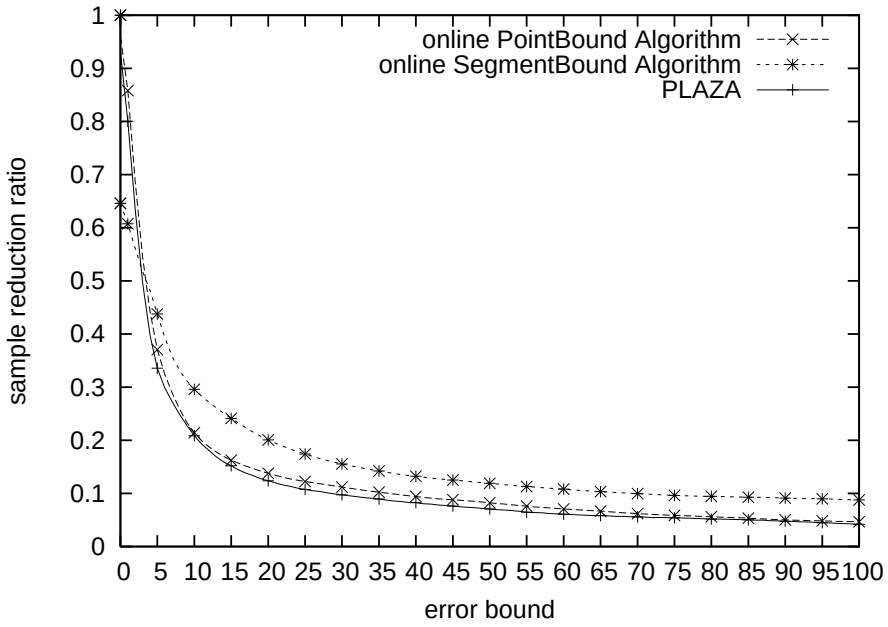

Figure 7.11: Results on an ECG data set.

Figure 7.11 compares the sample reduction ratio of algorithms PLAZA, PointBound, and SegmentBound on the ECG data set. The performance of algorithms PLAZA and PointBound is very similar. When the error bound is set to over 35, both algorithms can compress the data up to 10% of the original size. The gap between

	Max	Min	Average	Variance
Dim1	0.983	-0.916	0.280	0.020
Dim2	0.643	-0.661	0.016	0.0177
Dim3	-0.168	-2	-0.942	0.0165

Table 7.1: 3-dimensional accelerometer data from a sony ERS-210 aibo robot

algorithms SegmentBound and PointBound comes from the fact that the same error bound on the PLA-SegmentBound problem and the PLA-PointBound problem puts a tighter error constraint on the former Problem.

7.7 Experimental Results of Multidimensional PLA Problems

We test the performance of the multidimensional PLA problems on a three dimensional accelerometer data from a Sony ERS-210 Aibo Robot [11]. The Sony Aibo is a small, quadruped robot that comes equipped with a tri-axial accelerometer. This accelerometer returns data at a rate of 125 hertz. We test our algorithm on Aibo dataset obtained from the UCR time series collection. This dataset consists of the track of a robot playing soccer, including chasing after a moving ball, capturing the moving ball, kicking the moving ball, spinning in place searching for the ball, and standing stationary for a brief time period. The attributes of the multidimensional dataset are shown in Table 7.1. The table shows that the second dimension (Dim2) is harder to approximate due to the high variation in data.

The experimental results are shown in Figure 7.12. Note that the error bound on the X-axis is the whole error bound (i.e., $Mmaxerr$ or $Msegerr$) that has to be satisfied. The error bound allocated to each dimension is smaller than this value. For instance, if the specified bound in multidimensional PLA-PointBound problem, $Mmaxerr$, is ϵ, then the initial error bound on each dimension should be $\frac{\epsilon}{\sqrt{M}}$, where

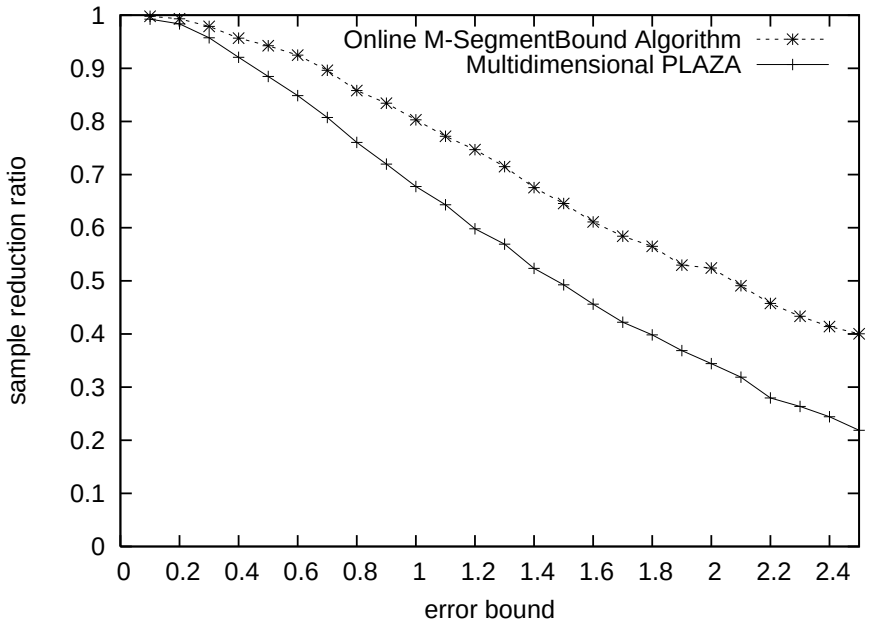

Figure 7.12: Multi-Dimensional PLA Problem

M is the size of dimension. As discussed in Chapter 6, the M-SegmentBound algorithm for the M-dimensional PLA-SegmentBound problem produces a minimum number of segments to compress a time series in the M-dimensional space. So the result in Figure 7.12 shows the minimum segmentation that is possible for the multi-dimensional PLA-SegmentBound problem. The multidimentional PLA-PointBound problem is, however, much harder to solve. When $Mmaxerr$ is equal to 1, we only obtain 60% bandwidth reduction ratio in this experiment. It is still unclear of the optimum result for the multidimentional PLA-PointBound problem, and we leave it as an open challenge.

Chapter 8

Conclusion

In this thesis, we tackle the problem of online compression of data streams in the resource-constrained network environment, where the traditional data compression techniques cannot apply. Particularly, we aim at *simple, fast* piecewise linear approximation (PLA) methods with quality guarantee for streaming data. Different from conventional PLA methods where the whole time series to be compressed is given in advance, our methods are particularly designed to compress data stream that is potentially unlimited, and the distribution of which is often unpredictable. We study two versions of the error-bounded PLA problems, namely the PLA-PointBound problem and the PLA-SegmentBound problem, which explore quality guarantees in different forms. In the PLA-PointBound problem, quality is guaranteed by specifying the maximal error that is permitted between real points and the approximated points at any time index. In the PLA-SegmentBound problem, the quality is specified by the maximal error that is permitted in each line segment.

For the above error-bounded PLA problems, we design fast online algorithms running in linear time complexity and requiring a constant space cost. To recap the key idea of our methods, we do not store the points explicitly, but instead monitor the range of all possible line segments that can be used to compress the points that have been seen but have not been compressed in a concise way. When a new point

arrives, we can check whether the point can be compressed using some line segments in the range. If so, it means that the new point can be compressed together with the points accumulated. We only need to adjust the range of the possible line segments to make sure the new point is also compressed. If not, it means that the new point cannot be compressed together with the points accumulated. A segment should then be output.

The online algorithms presented in this thesis for the error bounded PLA problems are optimal in terms of the number of generated segments. Moreover, our algorithms have an approximation factor of less than 2 to the optimum compression factor. To meet the needs from tiny, resource-constrained sensors, we develop another online algorithm, called PLAZA, which requires only very simple computation and generates connecting line segments. Furthermore, we design an optimal algorithm to tackle the multidimensional PLA-SegmentBound problem, and an approximate algorithm for the multidimensional PLA-PointBound problem. Our simulation results with Matlab and the test results using a real sensor test bed demonstrate that our fast online linear approximation methods are very effective for data stream compression and transmission over low bandwidth networks with nodes heavily constrained in computational power.

Equipped with the insights gained in this study, we see a lot of application opportunities for our methods. An appealing scenario is to build a "smart conference hall." By analyzing the data collected from an acoustic monitoring system deployed in a large conference place, we can identify and locate speakers as well as some of their activities. With this help, "smart conference hall" can automatically adjust audio devices to provide high-quality sound even if the speaker moves. This feature is especially important when the talk is recorded and broadcast on line. Further applications include acoustic monitoring systems in a surveillance area, for instance, bird surveillance in wildness and underwater monitoring system.

Meanwhile, there are some interesting open challenges for future work. For example, an interesting question is to design an online algorithm that can compute an ϵ-PLA consisting of connected line segments that has an approximation factor to the optimum.

Bibliography

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient Similarity Search In Sequence Databases. In D. Lomet, editor, *Proceedings of the 4th International Conference of Foundations of Data Organization and Algorithms (FODO)*, pages 69–84, Chicago, Illinois, 1993. Springer Verlag.

[2] B. S. Atal and L. S. Hanauer. Speech analysis and synthesis by linear prediction of the speech wave. *Journal of the Acoustical Society of America*, 50:637–655, 1971.

[3] Georgia Institute of Technology Broadband Wireless Networking Laboratory. Underwater acoustic sensor networks (uw-asn). http://www.ece.gatech.edu/research/labs/bwn/UWASN/.

[4] C. Buragohain, N. Shrivastava, and S. Suri. Space efficient streaming algorithms for the maximum error histogram. In *IEEE 23rd International Conference on Data Engineering (ICDE)*, pages 1026–1035, Istanbul, Turkey, 2007.

[5] K.P. Chan and A. W. Fu. Efficient time series matching by wavelets. In *ICDE '99: Proceedings of the 15th International Conference on Data Engineering*, pages 126–133, Washington, DC, USA, 1999. IEEE Computer Society.

[6] D.E. Cull, J. Hill, P. Bounadonna, R. Szewczyk, and A. Woo. A network-centric approach to embedded software for tiny devices. In *Proceedings of First*

International Workshop on Embedded Software (EMSOFT 2001), Tahoe City, CA, October 2001.

[7] W.K. Dobson, J.J. Yang, K.J. Smart, and F.K. Guo. High quality low complexity scalable wavelet audio coding. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference*, volume 1, pages 327–330. IEEE Computer Society, 1997.

[8] D. Douglas and T. K. Peucker. Algorithms for reduction of the number of points required to represent a digitized line or its caricature. 10(2):112–122, December 1973.

[9] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10(2):112–122, December 1973.

[10] J. G. Dunham. Optimum uniform piecewise linear approximation of planar curves. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(1):67–75, 1986.

[11] Sony Entertainment Robot Europe. Sony ers-210 aibo robot user guide. http://support.sony-europe.com/AIBO/downloads/en/ERS210CED_UG.pdf.

[12] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 419–429, New York, NY, USA, 1994. ACM Press.

[13] C.D. Giurcaneanu, I. Tabus, and J. Astola. Integer wavelet transform based lossless audio compression. In *Proceedings of NSIP-99, IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing*, volume 1, pages 378–382, Antalya, Turkey, 1999. IEEE Computer Society.

[14] M. T. Goodrich. Efficient piecewise-linear function approximation using the uniform metric: (preliminary version). In *SCG '94: Proceedings of the tenth annual symposium on Computational geometry*, pages 322–331, New York, NY, USA, 1994. ACM Press.

[15] P.S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms, multiresolution surface modeling course. In *Proceedings of the 24th International Conference on Computer Graphics and Interactive Techniques.*, 1997.

[16] Crossbow Technology Inc. Mica2 mote datasheet, wireless measurment system. "http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf".

[17] M. Ishijima, S. Shin, G.H. Hostetter, and J. Sklansky. Scan-along polygonal approximation for data compression of electrocardiograms. In *IEEE Transactions on Biomedical Engineering*, volume BME-30(11), pages 723–729, 1983.

[18] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3):263–286, 2001.

[19] E. Keogh, S. Chu, D. Hart, and M.J. Pazzani. An online algorithm for segmenting time series. In *ICDM:Proceedings of IEEE International Conference on Data Mining*, pages 289–296, 2001.

[20] E. Keogh and M. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, editors, *Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 239–241, New York City, NY, 1998. ACM Press.

[21] A. Koski, M. Juhola, and M. Meriste. Syntactic recognition of ECG signals by attributed finite automata. *Pattern Recognition*, 28(12):1927–1940, December 1995.

[22] G. Manis, G. Papakonstantinou, and P. Tsanakas. Optimal piecewise linear approximation of digitized curves. In *Digital Signal Processing Proceedings, 1997. DSP 97., 1997 13th International Conference*, pages 1079–1081. IEEE Computer Society, 1997.

[23] T. Palpanas, M. Vlachos; E. Keogh, D. Gunopulos, and W. Truppel. Online amnesic approximation of streaming time series. In *Data Engineering, 2004. Proceedings. 20th International Conference*, pages 339–349, 2004.

[24] S. Park, S.W Kim, and W. Chu. Segment-based approach for subsequence searches in sequence databases. In *SAC '01: Proceedings of the 2001 ACM symposium on Applied computing*, pages 248–252, New York, NY, USA, 2001. ACM.

[25] I. Popivanov and R.J Miller. Similarity search over time-series data using wavelets. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, pages 212–221, Washington, DC, USA, 2002. IEEE Computer Society.

[26] Y. Qu, C. Wang, and X.S. Wang. Supporting fast search in time series for movement patterns in multiples scales. In *Proceedings of the 7th ACM CIKM Int'l Conference on Information and Knowledge Management*, pages 251–258, November 1998.

[27] H. Shatkay and S. Zdonik. Approximate queries and representations for large data sequences. In *Proceedings of the 12th IEEE International Conference on Data Engineering*, February 1996.

[28] Hagit Shatkay and Stanley B. Zdonik. Approximate queries and representations for large data sequences. In *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*, pages 536–545, Washington, DC, USA, 1996. IEEE Computer Society.

[29] D. Sinha and J.D Johnston. Audio compression at low bit rates using a signal adaptive switched filterbank. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference*, volume 2, pages 1053–1056. IEEE Computer Society, 1996.

[30] P. Srinivasan and L.H. Jamieson. High quality audio compression using an adaptive wavelet packet decomposition and psychoacoustic modelling. In *IEEE Transactions on Signal Processing*, volume 46, pages 1085–1093, April 1998.

[31] H.J.L.M. Vullings, M.H.G. Verhaegen, and H.B. Verbruggen. ECG segmentation using time-warping. In *IDA '97: Proceedings of the Second International Symposium on Advances in Intelligent Data Analysis, Reasoning about Data*, pages 275–285, London, UK, 1997. Springer-Verlag.

[32] C. Wang and S. Wang. Supporting content-based searches on time series via approximation. In *Proceedings of the 12th International Conference on Scientific and Statistical Database Management*, July 2000.

[33] B. Warneke, M. Last, B. Leibowitz, and K.S.J. Pister. Smart dust: Communicating with a cubic-millimeter computer. *Ad Hoc Networks Journal*, 34(1):44–51, January 2001.

[34] Y. Wu, D. Agrawal, and A. Abbadi. A comparison of DFT and DWT based similarity search in time-series databases. In *CIKM '00: Proceedings of the ninth international conference on Information and knowledge management*, pages 488–495, New York, NY, USA, 2000. ACM Press.