High Performance Algorithms for Multiple Streaming Time Series

by

Xiaojian Zhao

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

January 2006

Dennis Shasha

“To my parents and my wife, for all they did for me”

Dedicated to all that helped me

Acknowledgements

This dissertation would never have materialized without the contribution of many individuals to whom I have the pleasure of expressing my appreciation and gratitude.

First of all, I gratefully acknowledge the persistent support and encouragement from my advisor, Professor Dennis Shasha. He provided constant academic guidance and inspired many of the ideas presented here. Dennis is a superb teacher and a great friend.

Secondly, I wish to express my deep gratitude to Professor Richard Cole. He has been offering his generous help since the beginning of my Ph.D. study, which is not limited to academic research. In particular, his help was indispensable for me to go through my first semester at NYU, four extremely tough months.

I thank Professor Clifford Hurvich for serving on both my proposal committee and thesis committee. His comments on my research have been very important. I also thank the other members of my dissertation committee, Ernest Davis and Farshid M. Asl, for their interest in this dissertation and for their feedback. Rich interactions with colleagues have improved my research and made it enjoyable. While I cannot list them all I would like to thank a few members of the database group: Alberto Lerner, David Tanzer, Aris Tsirigos, Xin Zhang and Zhihua Wang, who have lent both voices and helpful suggestions

in the course of this work. Additional thanks to Lee Rhodes for his comment on our system and Eamonn Keogh of University of California at Riverside for his data sets.

I am thankful for many friends with whom I share more than just an academic relationship: Rosemary Amico, Anina Karmen and Maria L. Petagna performed the administrative work required for this research. They were vital in making my stay at NYU enjoyable.

Finally and most importantly, I would like to thank my parents for their efforts to provide me with the best possible education.

Abstract

Data arriving in time order (a data stream) arises in fields ranging from physics to finance to medicine to music, to name a few. Often the data comes from sensors (in physics and medicine for example) whose data rates continue to improve dramatically as sensor technology improves. Furthermore, the number of sensors is increasing, so analyzing data between sensors becomes ever more critical in order to distill knowledge from the data. Fast response is desirable in many applications (e.g. to aim a telescope at an activity of interest or to perform a stock trade). In applications such as finance, recent information, e.g. correlation, is of far more interest than older information, so analysis over sliding windows is a desired operation.

These three factors – huge data size, fast response, and windowed computation – motivated this work. Our intent is to build a foundational library of primitives to perform online or near online statistical analysis, e.g. windowed correlation, incremental matching pursuit, burst detection, on thousands or even millions of time series. Beside the algorithms, we also propose the concept of "uncooperative" time series, whose power spectra are spread over all frequencies with any regularity.

Previous work [87, 98] showed how to do windowed correlation with Fast Fourier Transforms and Wavelet Transforms, but such techniques don't work

for uncooperative time series. This thesis will show how to use sketches (random projections) in a way that combines several simple techniques – sketches, convolution, structured random vectors, grid structures, combinatorial design, and bootstrapping – to achieve high performance, windowed correlation over a variety of data sets. Experiments confirm the asymptotic analysis.

To conduct matching pursuit (MP) over time series windows, an incremental scheme is designed to reduce the computational effort. Our empirical study demonstrates a substantial improvement in speed.

In previous work [87], Zhu and Shasha introduced an efficient algorithm to monitor bursts within windows of multiple sizes. We implemented it in a physical system by overcoming several practical challenges. Experimental results support the authors' linear running time analysis.

Contents

List of Figures

List of Tables

List of Appendices

Chapter 1

Introduction

Massive data are generated every second in various applications. For instance,

- In mission operations for NASA's MISR Satellite, spatial samples are acquired every 275 meters. Over a period of 7 minutes, a 360 km wide swath of Earth comes into view from cameras pointed in 9 different directions. Terabyte data are generated every day.
- In telecommunications, the AT&T long distance data stream consists of approximately 300 million records per day from 100 million customers.
- In astronomy, the MACHO Project to investigate the dark matter in the halo of the Milky Way monitors several million stars photometrically. The data rate is as high as several GBytes per night.
- There are roughly 50,000 securities trading in the United States, and up to 100,000 quotes and trades (ticks) are generated per second.

These applications share several special streaming characteristics, as pointed out below, and demonstrate that the query processing is different from that in

the conventional static or slow updating database system.

- Updates come in the form of insertions of new elements rather than modifications of existing data.
- Due to its continuous nature, a query should be answered in an incremental way.
- In most cases the large volume of data make it impossible to review past data, therefore a one pass processing algorithm is desired.

With the advance of new techniques, data rates continue to improve dramatically, so multi-stream analysis between data sources becomes ever more critical in order to distill knowledge from the data. Online response is desirable in many applications. Our intent is to build a foundational library of primitives to perform online or near online multi-stream information extraction on thousands or even millions of time series. Besides their immediate use, such primitives could provide a first level analysis of time series for online clustering and data mining systems.

In the second section, we will present a new sketch based data correlation strategy. It is an extension of DFT based statstream [98]. With comparable efficiency, the new algorithm can handle more data types. This new statstream is a randomized algorithm. The whole algorithm is based on the Johnson-Lindenstrauss (JL) Lemma which states that high dimensional data points can be mapped to a low dimensional space while preserving the norm within a factor. Our algorithm computes a synopsis vector for each time series; this is a random mapping from the original data space to a manageable low dimension. The JL

lemma proves that the approximation of the original distance may achieve sufficient accuracy with high probability as long as the synopsis size is larger than a bound. This synopsis is used to filter the non-correlated time series pairs in our algorithm. Although the lemma gives a bound, it appears to be a considerable overestimate in practice. We give a set of strategies to determine a good bound on the needed synopsis size as well as other system parameters. After the synopsis filtering, the correlation will be verified using the full time series data on the pairs surviving the filter. Only those pairs passing the verification will be reported as highly correlated.

We contrast this work with the considerable recent body of work on massive data streams [37, 77, 52] where the assumption is that data can be read once and is not stored. In our applications, we assume an initial filtering step must be completed in one pass, but a second pass may search for data in a well-organized and potentially growing data structure.

In the third section, we will introduce a novel incremental matching pursuit (MP) scheme. The new scheme allows the updating of approximating vectors every basic window. The consistency of projection space between consecutive sliding windows enables a significant saving of computational efforts compared to the naive approach which requires a full calculation for each window position. To the best of our knowledge nobody has discussed how to apply MP incrementally to a group of time series.

The fourth section is an implementation of Zhu and Shasha's Shifted Binary Tree (SBT) [87] in a physical system. We will give an overview of the practical challenges encountered in the implementation and our solutions.

An empirical study of each algorithm will be given in the corresponding section.

At the end of the thesis, we discuss how we intend to extend our algorithms in future work.

Chapter 2

Review

2.1 Streaming Database

In many practical cases, the data set we will process is subject to dynamic updating, that is, new data arrive continuously at a random rate. Due to its varied applications in finance, physics, telecommunication etc, there has been a lot of research on streaming databases.

Babcock et al. [12] discuss models and issues in designing a new type of Data Stream Management System (DBMS). Conventional DBMS's are inherently unable to support continuous queries, which are typical for streaming data. Due to its limited memory, a traditional DBMS cannot hold continuously arriving data. Moreover, in most if not all cases, an approximate answer is sufficient for queries to data streams. This cannot be handled appropriately by current DBMS's which are normally designed to provide an exact answer. In Babcock et al.'s paper, general schemes such as sampling, batch processing, and synopsis data structure are discussed, focusing on how to keep up with the data stream rate and to produce timely answers. They discuss a query language extend-

ing standard SQL. In their follow-up work, more technical issues are addressed such as achieving a lower memory overhead by a careful query arrangement [13], sharing resources among sliding windows for aggregate queries [10], and execution of joins in a memory-limited setting [88]. Several other database systems for streaming data have also been constructed: Aurora [2], MAIDS [3], Niagara [4], Telegraph [1], to name just a few of them.

Dater et al. [35] consider the problem of maintaining statistics such as count, sum, and L_p-norm over a sliding window of the last N elements. Dater and co-authors propose a scheme for the count as the fundamental technique. To avoid inaccuracy in the case of skewed data segments, with which other histogram algorithms fail to deal, the sliding window is divided into buckets of pseudo-exponentially increasing sizes. The merging or creation operations are conducted over the buckets whenever a new "1" arrives or the data element expires. The well structured bucket sizes guarantee that the estimate of count is within an error bound of $1+\epsilon$ and the total number of bits of memory needed is $\frac{1}{\epsilon}\log^2 N$ where prior knowledge of N is not required. The authors then extend this basic scheme to maintain other statistics such as sum and L_p-norm.

Gehrke et al. [48] consider the computation of correlated aggregates over multiple streams. They present a set of strategies with which such queries as $COUNT\{x : x > 0.5 * MAX(x)\}$ and $MAX\{y : x < AVG(x)\}$ can be computed approximately in a single pass over the data stream. The answer to these queries depends on low level independent information: extreme (MAX, MIN) or average (AVG). They maintain a histogram to represent the data distribution around the maximum, minimum or average of data seen. To avoid degradation from the given partitioning policy with the arrival of new tuples, two approaches, wholesale and piecemeal, were proposed. In the wholesale

approach, the buckets are revised from scratch, while in the piecemeal approach, the existing bucket allocation is preserved whenever possible. Their algorithm can handle both sliding window queries (e.g. in the past 30 mins) and landmark queries (e.g. since last month). Many other one-pass algorithms have been proposed by researchers to obtain median, quantiles and other statistics [52, 77], join query [31, 7, 46], and mining [47].

Many of the sampling and histogram based algorithms make some assumptions regarding the data distribution such as that data come from a uniform distribution. In some cases the central limit theorem is hypothesized in order to tighten the bound or to reduce the sampling size.

Zhu and Shasha [87] propose a strategy for computing sliding window correlations among multiple data sources. Here a sliding window is a moving window over the time series stream. Suppose there are N time series and a sliding window size of w. In each stream, sliding windows begin at time 0, b, $2b$, ... where $b < w$ and b is called the basic window size. Two sliding windows $x1$ from stream $s1$ and $x2$ from stream $s2$ that begin at times $t1$ and $t2$, respectively, are said to correlate highly if their correlation is above a threshold value. Zhu and Shasha's algorithm finds all highly correlated pairs of sliding windows assuming that each window of data can be well modeled by its first few Fourier coefficients. Their algorithm can deal with both synchronous and asynchronous correlation.

2.2 Time Series Similarity Measures

Given a query point, a typical query, called a similarity query, is to find the most similar data points in the data set or its nearest neighborhood in the

metric space. Jagadish et al. [59] give a general discussion about similarity queries. Das et al. [32] talked about some fundamental problems concerning similarity queries. Many papers discussed similarity measurements such as time warped measures [16, 95, 99], weighted measures [67, 91] and multi-dimensional indexing [90]. See [54] for a tutorial.

Correlation is also a similarity metric. The most widely used statistic measurement is Pearson correlation which will be defined later. Though Pearson correlation is optimal for detecting linear relationships, it is a weak test for highly non-linear relationships. In addition Pearson correlation can be strongly affected by a single outlier. Therefore a non-parametric form of correlation, called Spearman's rank correlation is also used in many applications [89, 6]. In addition to temporal computation, the correlation can also be performed spatially [96].

One unavoidable aspect of time series similarity measure is the high dimension. For instance, a vector longer than 50 is common in applications. However high dimensional data pose a serious problem to those algorithms designed for low dimensions. Therefore data reduction is the first step of many time series techniques.

Next we give a brief general description of GEMINI and then review several of the most popular data reduction methods, among which random projection is emphasized due to its special role in our algorithm.

2.2.1 GEMINI Framework

GEMINI (Generic Multimedia Indexing Method) framework was first proposed by Faloutsos in his celebrated paper [43]. In this paper, he gave a framework

which procudes no false negatives in the output.

In GEMINI the distance measure will be computed first in the feature space and then in the original data space. The former has the characteristic that all the original distances are reduced. Therefore the query for the data within a specific distance from the query point will be returned with a superset of true positives. In the second step, the data points passed are then verified in the original data space.

The first step acts as a filter, while the verification in the original data space eliminates the false positives. All and only true positives are finally returned.

The mapping functions between data space and feature space vary. The two most popular ones are Discrete Fourier Transforms (DFTs) and Discrete Wavelet Transforms (DWTs). Often, the first several coefficients of DFT or DWT capture the trends of the original time series.

Figure 2.1 shows the GMINI framework.

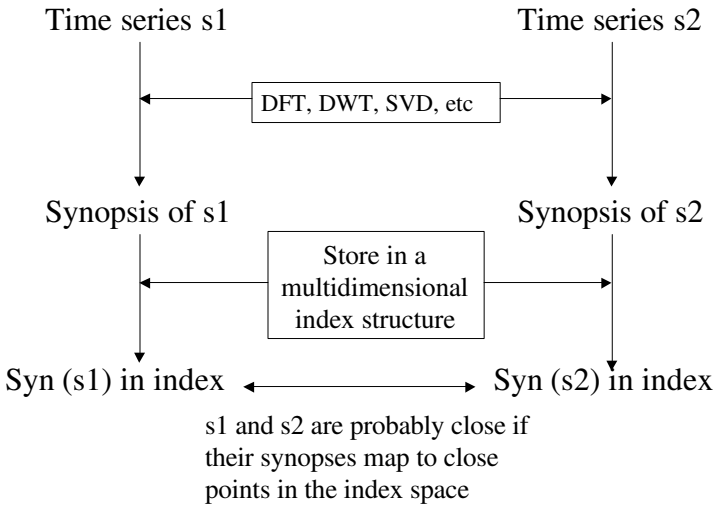


Figure 2.1: GEMINI Framework

2.2.2 Data Reduction

High dimensional data are used in various applications e.g. multimedia similarity matching, time series correlation searching, etc. However, many algorithms or techniques that work quite well in a low dimension deteriorate when scaled to a high dimensional space. This is the notorious "Curse of Dimensionality". For instance, in a $2D$ data space where data points are scattered evenly, the number of the data points contained in a $2r - radius$ ball is four times that in a $r - radius$ ball. However, this ratio will increase exponentially to 2^n in a n-dimensional space. A lot of the work was done to explore this huge discrepancy [14, 21, 62, 15, 19, 20]

Naturally, data reduction has been a topic of interest; it reduces high dimensional data into a manageable synoptic data structure while preserving the characteristics of the data to a large extent. Most data reduction techniques for time series result in reducing the dimensionality of the time series.

Linear Orthogonal Transform

In L_p distances where p can be any positive number, the Euclidean distance ($p = 2$) is most used. Therefore a transform is preferred that preserves the distance, which is met by any orthonormal transform. Among them are DFT, Wavelet and SVD. Such data reduction techniques follow the scheme below:

- Find a set of complete, normal and orthogonal vectors V of the same size as the time series;
- Transform the time series to the space spanned by V;
- Keep the most significant d coordinates ($d < n$).

These first d coordinates form a vector which is used to approximate the original time series. The choice of the user-defined threshold d depends on the characteristics of the data sets.

Discrete Fourier Transform (DFT): The Discrete Fourier Transform was first used in reducing the dimension of time series in Agrawal, Faloutsos and Swami's work [9]. It has been widely used since then in the database and data mining community [43, 84, 94, 79, 73, 100].

DFT has its pros and cons. On the positive side, DFT possesses a good ability to compress most natural signals, especially those with obvious trends. The computation of the DFT transform is fast ($O(n \log n)$). It is also able to support time warped queries. However, DFT cannot deal directly with sequences of different lengths and it does not support weighted distance measures.

Discrete Wavelet Transform (DWT): The Fourier Transform summarizes the frequency characteristics of time series from a global view. Thus it is difficult to maintain a high resolution in time. Although "Windowed Fourier Transform (WFT)" was proposed to address this shortcoming, the uniform resolution in time and frequency makes it unsuitable for a multi-scale analysis. The Wavelet Transform, a generalized version of WFT, can avoid this difficulty by projecting the signal to a multi-level space with a local wavelet.

There are many wavelets and comparisons are made between different wavelets [25, 83]. As a time series representation, the wavelet is good at compressing stationary signals. The approximation can be computed linearly, but wavelet processing requires that the signals must have a length $n = 2^{integer}$, otherwise the time series have to be padded, which introduces a cost.

In addition to their use in conventional similarity searching [25, 55, 83], wavelets are also popular in the query approximation [24, 49, 78, 86].

It is commonly believed that DWT works for any application in which DFT works. However, Wu [93] compared DFT and DWT and claimed that although the DWT based technique has several advantages, DWT does not reduce relative matching errors or increase query precision in similarity search. He supported his idea by exploring the conjugate property of DFT in the real domain; he showed that the DFT-based and DWT-based techniques yield comparable results on similarity search in time-series databases.

Singular Value Decomposition (SVD): SVD [69, 85] is an optimal linear dimensionality reduction technique as we will discuss later. However, SVD is computationally expensive. It needs $O(MN^2)$ time and $O(MN)$ space where M is the row number of a matrix while N is the column number. Any insertion into the database requires recomputation of the transformation. SVD can not support weighted distance measures or non Euclidean measures.

There is some promising research to reduce SVD's time and space complexity. Drineas [38, 37] proposed the randomized SVD approach. He claims that the sampling of the rows or columns can form a new matrix which with high probability shares similar singular vectors to those of the original matrix. In Papadimitriou's work [82], random projection is used in latent semantic indexing [36] to map the rows into low dimension in order to reduce the size of the matrix.

The orthogonal transforms differ in their properties. The DFT and Wavelet Transform are data-independent, which means that the transformation matrix is determined a-priori, while data-dependent transforms can be fine-tuned to the specific data set and therefore in principle can achieve better performance, concentrating the energy into a few features in the feature vector. On the other hand, data-dependent algorithms suffer from expensive computation time. Due

to the evolution of data sets over time, a recomputation of the transformation matrix is necessary to avoid performance degradation.

Therefore data independent transforms (DFT and DWT) are mostly used in algorithms where data change rapidly while SVD finds its application where data is updated slowly or when the expensive computation is affordable.

2.2.3 Random Projection

Let's first give some intuition for the random projection approach.

Imagine you are walking in an unfamiliar place and you unfortunately get lost. The only equipment you have is a cell phone. You want to know if you are close to your friend with whom you want to meet. The easiest way to find your bearings in this setting is to exchange the relative location information of each other, e.g. "Hi, I am about 100 meters from a silver building and 50 meters from CircuitCity...". If you two are close to each other, the similar location information will be shared. Of course, there is the possibility that two places are surrounded by the similar landmarks but located in NYC and LA respectively. However if the number of the landmarks is sufficiently large and they are chosen randomly, we may give a unique conclusion with high probability.

Next let's examine the formal definition of random projection.

Given a vector $\vec{t} = t[1, \cdots, m]$, its sketch vector $\vec{S}(t)$ is constructed as follows. We form a random vector $v_i[1, \cdots, m]$ by picking each component $v_i[j]$ to be an independent random variable with some specific distribution (e.g. normal distribution N(0,1)). The sketches of the data vector are constructed by the dot product with the random vector. That is,

$$\vec{S}(t)[i] = \vec{t} \cdot \vec{v}_i = \sum_j t[j] \cdot v_i[j]$$

Example. Given $\vec{t}$=(1 2 3 4) and suppose we want to construct a sketch vector of size two. The random variables are drawn from N(0,1). The random vectors are $\vec{v}_1$ and $\vec{v}_2$ where

$$\vec{v}_1 = (0.13, -0.24, 0.47, -0.19)$$

$$\vec{v}_2 = (-0.25, -0.64, 0.17, -0.89)$$

Then the sketch of $\vec{t}$ is (0.3, -4.58).

The sketch of a vector possesses many advantageous properties. Lots of researchers have studied it since the following lemma was first developed by Johnson and Lindenstrauss.

Original Johnson-Lindenstrauss Lemma. *Let $\vec{v}_1$,$\vec{v}_2$,$\cdots$,$\vec{v}_m$ be a sequence of points in the d-dimensional space over the reals and let ϵ, $F \in (0, 1]$. Then there exists a linear mapping f from the points of the d-dimensional space into the points of the k-dimensional space where $k = O(\log(1/F)/\epsilon^2)$ such that the number of vectors which approximately preserve their length is at least $(1-F)m$. We say that a vector $\vec{v}_i$ approximately preserves its length if:*

$$\|\vec{v}_i\|^2 \leq \|f(\vec{v}_i)\|^2 \leq (1+\epsilon)\|\vec{v}_i\|^2$$

or in a more direct form:

Johnson-Lindenstrauss (JL) Lemma. *For any $0 < \epsilon < 1$ and any integer n, let k be a positive integer such that,*

$$k > 4(\epsilon^2/2 - \epsilon^3/3)^{-1} \log n$$

Then for any set V of n points in R^d, there is a map $f : R^d \to R^k$ such that for all $u, v \in V$

$$(1-\epsilon)\| u - v \|^2 \leq \| f(u) - f(v) \|^2 \leq (1+\epsilon)\| u - v \|^2$$

Further this map can be found in randomized polynomial time

It is proven that a suitable random vector can be drawn from an order 1 stable distribution such as normal distribution.

Dimitris Achliptas extended it to more distributions by his lemma.

Dimitris Lemma. *Let P be an arbitrary set of n points in R^d, represented as an $n \times d$ matrix A. Given $\epsilon, \beta > 0$, let*

$$k_0 = \frac{4+2\beta}{\epsilon^2/2 - \epsilon^3/3} \log n$$

For integer $k \geq k_0$, let R be a $d \times k$ random matrix with $R(i;j) = r_{ij}$, where $\{r_{ij}\}$ are independent random variables from either one of the following two probability distributions:

$$r_{ij} = \begin{cases} +1 & \textit{with probability } 1/2; \\ -1 & \textit{with probability } 1/2. \end{cases}$$

or

$$r_{ij} = \begin{cases} +\sqrt{3} & \textit{with probability } 1/6; \\ 0 & \textit{with probability } 2/3; \\ -\sqrt{3} & \textit{with probability } 1/6. \end{cases}$$

Let

$$E = \frac{1}{\sqrt{k}} AR$$

Let $f : R^d \rightarrow R^k$ *map the* i^{th} *row of* A *to the* i^{th} *row of* E. *With a probability at least* $1 - n^{-\beta}$, *for all* $u, v \in P$

$$(1 - \epsilon)\| u - v \|^2 \leq \| f(u) - f(v) \|^2 \leq (1 + \epsilon)\| u - v \|^2$$

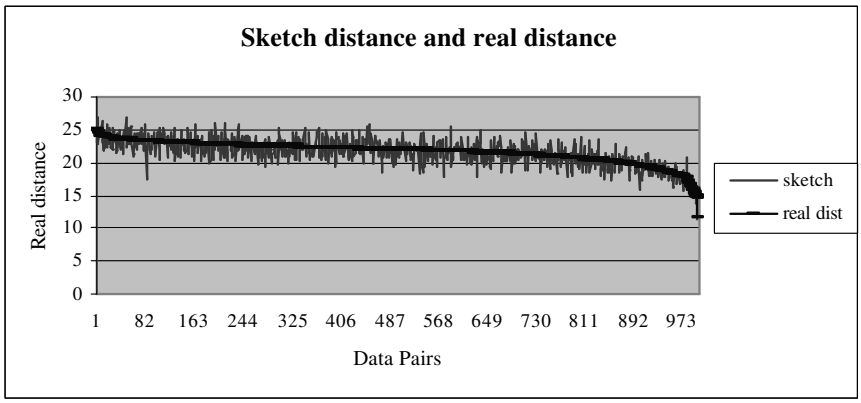
Now we present the definition of sketch distance.

Sketch distance: The Euclidean distance of the sketches of two time series, that is

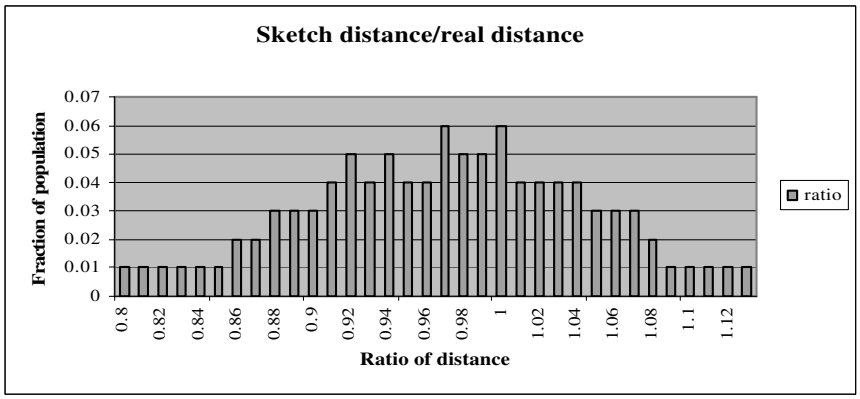
$$d_{sk} = \|x_{sk} - y_{sk}\| = \sqrt{(x_{sk1} - y_{sk1})^2 + (x_{sk2} - y_{sk2})^2 + \cdots + (x_{skk} - y_{skk})^2}$$

Figure 2.2 shows the comparison between the sketch distance and original distance in the form of bar and curve graph. In the figures, the sketch size is 64. In Figure 2.2(a), the real distance is sorted in descending order, and the sketch distance distances are ordered correspondingly; the sketch distance occurs in a tight band around the real distance. In Figure 2.2(b) the ratio between the sketch distance and real distance is given. The symmetric and bell-like shape looks like a normal distribution with center at the ideal value. Figure 2.3 also indicates the equivalence between sketch distance and real distance.

In the empirical study, more results are shown with different sketch size.



(a) The sketch distance vs. the real distance of stock return pairs



(b) The ratio of sketch distance /real distance for stock return pairs

Figure 2.2: The sketch distances and the real distances of stock returns are close

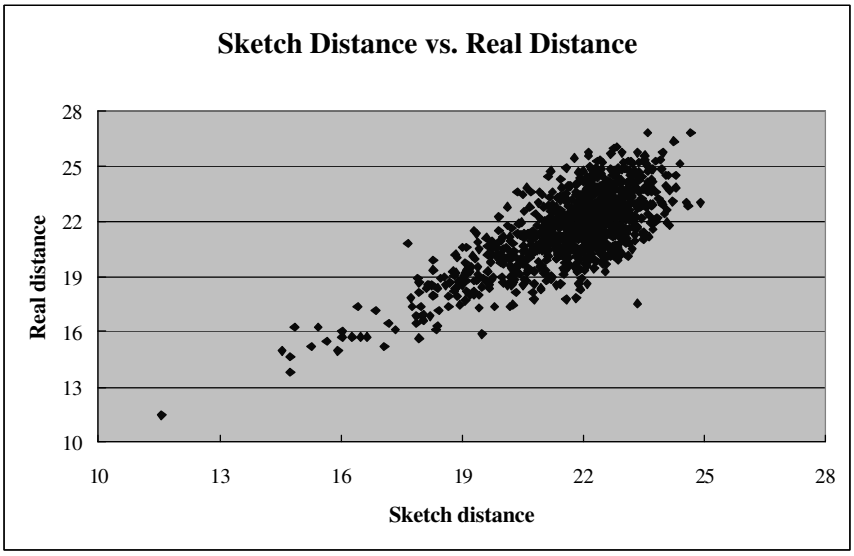


Figure 2.3: GEMINI Framework

Although the JL lemma first appeared in 1984, it was not brought to the theoretical computer science community until date [74]. Since then many algorithms have been developed based on the idea of random projection. Now it has become a standard way to improve the complexity of approximate algorithms by polynomial [82, 23, 50] or even exponential factors [58].

After the introduction of the JL lemma, several researchers gave simpler proofs because of its importance, e.g. [45, 58, 11, 34, 8]. The tightest bound, $k > k_0 = 4(\epsilon^2/2 - \epsilon^3/3)^{-1}$, was given in [34]. In the proof of [8], a new random distribution is introduced.

These proofs, however, are all nondeterministic. None of them give a strategy to find a mapping. In the work of Engebretsen and Indyk [41], a deterministic algorithm is proposed to find an embedding with the properties guaranteed by the JL lemma.

One thing to note is that, in addition to Euclidean distance, random projec-

tion can be used to approximate L_p-distance [56]. It is also possible to approximate the Hamming or L_0-distance using stable distributions [29].

Random projection is used frequently in the dimensionality reduction explicitly or implicitly, which has been found to be a computationally efficient, yet sufficiently accurate method. Indyk et al. [57] use it to identify the trends of the time series. In their work a set of sketches over the time series of length of power of two are first computed, which serve as the sketch base to produce the sketch of time series of arbitrary length by summation. Papadimitriou et al. [82] use random projection in the preprocessing of textual data, prior to applying Latent Semantic Indexing (LSI). In their work the columns of the random projection matrix are required to be orthogonal to each other, which is proven not necessary later [17]. However, it is noticed that the random vector may be organized carefully for a specific purpose. In the paper by Kaski [61] random projections were used on textual data in WEBSOM, a program that organizes document collections into Self-Organizing Map (SOM). Kurimo [70] also applies random projection to the indexing of audio documents, prior to using LSI and SOM. Kleinberg [68] and Indyk and Motwani [58] use random projections in nearest-neighbor search in a high dimensional Euclidean space. Dasgupta [33] has used random projections in learning high-dimensional Gaussian mixture models. Due to the excellent quality of its distance preservation, random projection naturally finds its application in clustering. Zhang, Fern, and Brodley illustrate in their paper [44] that random projection can help to uncover the natural structure in high dimensional data by multiple runs of clustering. They exploit how to choose the dimensionality for a random projection in order to preserve separation among clusters in general clustering applications. In addition work has been done to apply the random projection in privacy-preserving

data mining [75].

Ella Bingham and Heikki Mannila [17] show that projecting the data onto a random lower-dimensional subspace yields results comparable to conventional dimensionality reduction methods such as principal component analysis. In their work the similarity of data vectors, either in terms of their Euclidean distance or their inner product, is measured. They also note that the JL theorem and analogous results in [8] give much larger bounds on the size of the sketch than necessary for good results.

2.3 Matching Pursuit

Matching Pursuit (MP) is an algorithm, introduced by Stephan Mallat and Zhifeng Zhang [76], for approximating a target vector by greedily selecting a linear combination of vectors from a dictionary. Figure 2.4 holds pseudo-code for the MP algorithm. MP takes as input a target vector v_t and a set of vectors V from which it quickly extracts as output a smaller subset V_A along with weights c_i so that $v_t \approx \sum_{v_i \in V_A} c_i v_i$.

MP has many applications including in image processing [42], physics [92, 22], medicine [39, 18] and more. Several researchers have proposed fast variants of the algorithm [53, 80, 26]. In this thesis we will give a brief description of our incremental matching pursuit technique.

Given a vector pool containing n time series $V = (v_1, v_2, \cdots, v_n)$, a target vector v_t, tolerated error ϵ, and approximating vector set $V_A = \emptyset$. Define $\cos\theta_i = \vec{v}_t * \vec{v}_i$ as the cosine between v_t and a vector v_i in V. Here vector $\vec{v} = \frac{v}{\|v\|}$.

1. Set $i = 1$;
2. Search the pool V and find the vector v_i whose $|\cos\theta_i|$ with respect to v_t is maximal;
3. Compute the residue $r = v_t - c_i v_i$ where $c_i = \frac{\|v_t\|}{\|v_i\|}\cos\theta$. $V_A = V_A \cup \{v_i\}$
4. If $\|r\| < \epsilon$ terminate, return V_A
5. Else set $i = i + 1$ and $v_t = r$, go back to 2

Figure 2.4: Matching Pursuit (MP) algorithm

Chapter 3

Statstream Over Uncooperative Time Series[1]

3.1 StatStream Revisited

Zhu and Shasha [87] propose a strategy for computing sliding window correlations among multiple data sources. They use DFT as a means to reduce the dimension of data. The first a few coefficients of DFT transformation are used to compute the "distance" between time series. Those time series pairs whose distance in coefficient space is smaller than a certain threshold are picked out for the second-stage verification. The above operations are performed within each sliding window.

[1]This work is published in SIGKDD'05 [28]

3.2 Problem Statement

Correlation, over windows from the same or different streams, has many variants. This thesis focuses on synchronous and asynchronous (a.k.a. lagged) variations, defined as follows:

- (Synchronous correlation) Given N_s streams, a start time t_{start}, and a window size w, find, for each time window W of size w, all pairs of streams S_1 and S_2 such that S_1 during time window W is highly correlated (over 0.95 typically) with S_2 during the same time window. (Possible time windows are $[t_{start} \cdots t_{start+w-1}]$, $[t_{start+1} \cdots t_{start+w}]$, $\cdots$ where t_{start} is some start time.)

- (Asynchronous correlation) Allow shifts in time. That is, given N_s streams and a window size w, find all time windows W_1 and W_2 where $|W_1| = |W_2| = w$ and all pairs of streams S_1 and S_2 such that S_1 during W_1 is highly correlated with S_2 during W_2.

3.2.1 What Makes a Time Series Cooperative?

Given N_s streams and a window of size $winsize$, computing all pairwise correlations naively requires $O(winsize \times (N_s)^2)$ time. Fortunately, extremely effective optimizations are possible, though the optimizations depend on the kind of time series they are.

- **Category 1 ("cooperative"):** The time series often exhibits a fundamental degree of regularity, (or in other words, its energy is concentrated in a few frequency components) at least over the short term, allowing long

time series to be compressed to a few coefficients with little loss of information using data reduction techniques such as Fast Fourier Transforms and Wavelet Transforms. Using Fourier Transforms to compress time series data was originally proposed by Agrawal et al. [9]. This technique has been improved and generalized by [43, 72, 84]. Wavelet Transforms (DWT) [25, 49, 83, 93], Singular Value Decompositions (SVD) [69], and Piecewise Constant Approximations [66, 64, 81, 94] have also been proposed for similarity search. Keogh has pioneered many of the recent ideas in the indexing of dynamic time warping databases [63, 90]. The performance of these techniques varies depending on the characteristics of the datasets [87].

- **Category 2 ("uncooperative"):** In the general case, such regularities are absent. However, sketch-based approaches [8, 56] can still give a substantial data reduction. These are based on the idea of taking the inner product of each time series window, considered as a vector, with a set of random vectors (or equivalently, this can be regarded as a collection of projections of the time series windows onto the random vectors). Thus, the guarantees given by the Johnson-Lindenstrauss lemma [60] hold. In time series data mining, sketch-based approaches have been used to identify representative trends [30, 57] and to compute approximate wavelet coefficients [49], for example.

3.3 Our Contribution

Previous work [98, 87] shows how to solve the windowed correlation problem in the cooperative setting using high quality digests based on Fourier transforms. Unfortunately, many applications generate uncooperative time series. Stock market returns (change in price from one time period (e.g., day, hour, or second) to the next divided by initial price, symbolically $(p_{t+1} - p_t)/p_t$) for example are "white noise-like." That is, there is almost no correlation from one time point to the next.

For collections of time series that don't concentrate power in the first few Fourier/Wavelet coefficients, which we term *uncooperative*, we proceed as follows:

1. We adopt a sketch-based approach.

2. Unfortunately, computing sketches directly for each neighboring window is very expensive. For each new datum, for each random vector, it costs $O(sw)$ time where sw is the size of the sliding window. (We will be using 25 to 60 random vectors.) To reduce this expense, we combine two ideas: convolutions and "structured random vectors" to reduce the time complexity to $O(sw/bw)$ integer additions and $O(\log bw)$ floating point operations per datum and random vetor. The length bw is the time delay before a correlation is reported (e.g., if sw were an hour then bw might be a minute).

3. Even with this, we obtain sketch vectors of too high a dimensionality for effective use of multi-dimensional data structures. We combat this well-known "curse of dimensionality" by using groups of sketches and combin-

ing the results as in the scheme due to [71].

4. There are four parameters to be set (two of which we introduce later). Optimizing these parameters to achieve good recall and precision requires a search through a large parameter space. For this we use combinatorial design. We validate both the use of combinatorial design and the stability of the parameter choices experimentally through bootstrapping.

The end result is a system architecture that, given the initial portions of a collection of time series streams, will determine (i) whether the time series are cooperative or not; (ii) if so, it will use Fourier or Wavelet methods; and (iii) if not, it will discover the proper parameter settings and apply them to compute sketches of the evolving data streams.

3.4 Algorithmic Ideas

Following [87, 98], our approach begins by distinguishing among three time periods from smallest to largest.

- timepoint – the smallest unit of time over which the system collects data, e.g., a second.
- basic window – a consecutive subsequence of timepoints over which the system maintains a *digest* (i.e., a compressed representation) e.g., two minutes.
- sliding window – a user-defined consecutive subsequence of basic windows over which the user wants statistics, e.g., an hour. The user might ask,

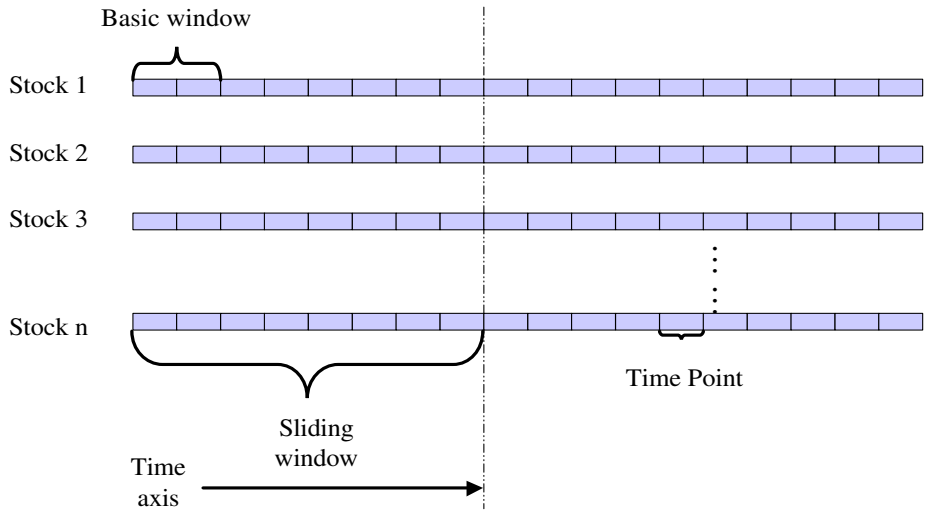


Figure 3.1: Sliding windows and basic windows.

"which pairs of streams were correlated with a value of over 0.9 for the last hour?"

Figure 3.1 shows the relationship between sliding windows and basic windows.

The use of the intermediate time interval called the basic window yields two advantages [87, 98],

1. (Near online response rates) Results of user queries need not be delayed more than the basic window time. In this example, the user will be told about correlations for the 2PM to 3PM window by 3:02 PM and correlations for the 2:02 PM - 3:02 PM window by 3:04 PM.[2]

2. (Free choice of window size) Maintaining stream digests based on the basic

[2]One may wonder whether the basic window and therefore the delay can be reduced. The tradeoff is with computation time. Reducing the size of the basic window reduces the compression achieved and increases the frequency and hence expense of correlation calculations.

window allows the computation of correlations over windows of arbitrary size (chosen up front) with high accuracy.

For cooperative time series, the strategy is just this: given a digest consisting of the first few, say k, Fourier coefficients for a sliding window up to basic window number n, recompute the digest after basic window $n+1$ ends as follows. Compute a digest of basic window $n+1$ and then update the sliding window digest's Fourier coefficients. This update takes constant time per coefficient.

Fourier Transforms do an excellent job of summarizing many time series, as pointed out by [9] and subsequently used in [43, 51, 72, 84]. Empirical studies [98] using both random walk and stock market price data show that the cost savings compared to a naive approach are significant and the accuracy is excellent (no false negatives and few false positives). Unfortunately, as pointed out above, the Fourier Transform behaves very poorly for white noise style data. For such data, we use sketches.

3.4.1 The Sketch Approach

The sketch approach, as developed by Kushikvitz et al. [71], Indyk et al. [56], and Achlioptas [8], provides a very nice guarantee: with high probability a random mapping taking points in R^m to points in $(R^d)^{2b+1}$ (the (2b+1)-fold cross-product of R^d with itself) approximately preserves distances (with higher fidelity the larger b is).

Quantitatively, given a point $\mathbf{x} \in R^m$, we compute its dot product with d random vectors $\mathbf{r}_i \in \{1, -1\}^m$. The first random projection of $\mathbf{x}$ is given by $\mathbf{y}_1 = (\mathbf{x} * \mathbf{r_1}, \mathbf{x} * \mathbf{r_2}, ..., \mathbf{x} * \mathbf{r_d})$. We compute $2b$ more such random projections $\mathbf{y_1}, ..., \mathbf{y_{2b+1}}$. If $\mathbf{w}$ is another point in R^m and $\mathbf{z_1}, ..., \mathbf{z_{2b+1}}$ are its projections

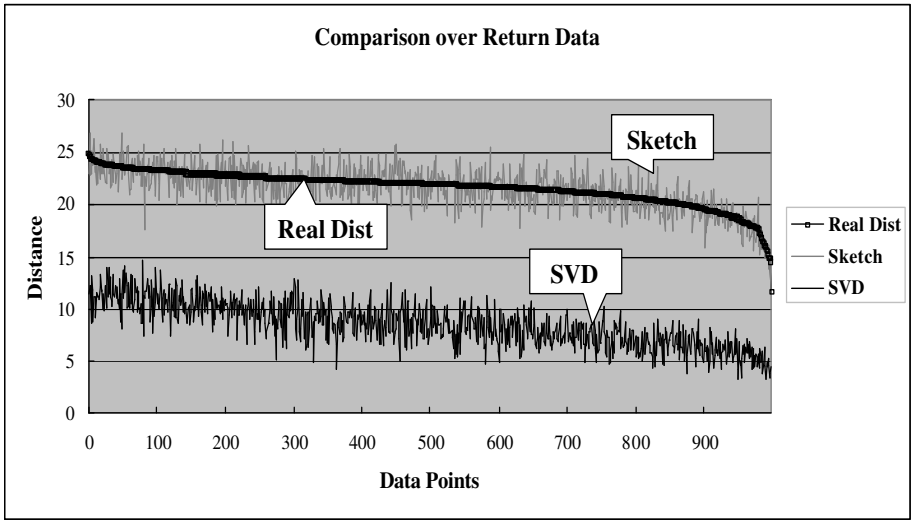


Figure 3.2: The sketch approach is superior to the Singular Value Decomposition, Wavelet, and Discrete Fourier Transform approaches for uncooperative time series. Of those three, Singular Value Decomposition is the best so it is the one to which sketches are compared.

using dot products with the same random vectors then the median of $\|\mathbf{y_1} - \mathbf{z_1}\|, \|\mathbf{y_2} - \mathbf{z_2}\|, ...\|\mathbf{y_{2b+1}} - \mathbf{z_{2b+1}}\|$ is a good estimate of $\|\mathbf{x} - \mathbf{w}\|$. It lies within a $\theta(1/d)$ factor of $\|\mathbf{x} - \mathbf{w}\|$ with probability $1 - (1/2)^b$.

Sketches work much better than SVD method (thus better than Fourier methods) for uncooperative data. Figure 3.2, compares the distances of the Fourier and sketch approximations for 1,000 pairs of 256 timepoint windows having a basic window size of length 32. Here sketch size=30 and SVD coefficient number=30. As you can see, the sketch distances are closer to the real distance. On the other hand, the SVD approximation is essentially never correct.

For each random vector $\mathbf{r}$ of length equal to the sliding window length $sw = nb \times bw$, we compute the dot product with each successive length sw chunk

of the stream (successive chunks being one timepoint apart and bw being the length of a basic window). As noted by Indyk [57], convolutions (computed via Fast Fourier Transforms) can perform this efficiently off-line. The difficulty is how to do this efficiently online.

Our approach is to use a "structured" random vector. The apparently oxymoronic idea is to form each structured random vector $\mathbf{r}$ from the concatenation of nb random vectors: $\mathbf{r} = \mathbf{s_1}, ..., \mathbf{s_{nb}}$ where each $\mathbf{s_i}$ has length bw. Further each $\mathbf{s_i}$ is either $\mathbf{u}$ or $-\mathbf{u}$, and $\mathbf{u}$ is a random vector in $\{1, -1\}^{bw}$. This choice is determined by a random binary k-vector $\mathbf{b}$: if b_i=1, $\mathbf{s_i}$=$\mathbf{u}$ and if b_i=0, $\mathbf{s_i}$=$-\mathbf{u}$. The structured approach leads to an asymptotic performance of $O(nb)$ integer additions and $O(\log bw)$ floating point operations per datum and per random vector. In our applications, we see 30 to 40 factor improvements in runtime over the naive method.

In order to compute the dot products with structured random vectors, we first compute dot products with the random vector $\mathbf{u}$. We perform this computation by convolution once every bw timesteps. Then each dot product with $\mathbf{r}$ is simply a sum of nb already computed dot products. (We explain this in more detail in the appendix.)

The use of structured random vectors reduces the randomness, but experiments show that this does not appreciably diminish the accuracy of the sketch approximation, as we can see from Figure 3.3.

Though structured random vectors enjoy good performance, as we will see, please note that a clever use of unstructured (that is, standard) random vectors together with convolutions can lead to an asymptotic cost of $O(\log sw \log(sw/bw))$ floating point multiplications per datum. Structured random vector approaches use $O(\log bw)$ multiplications and $O(sw/bw)$ additions

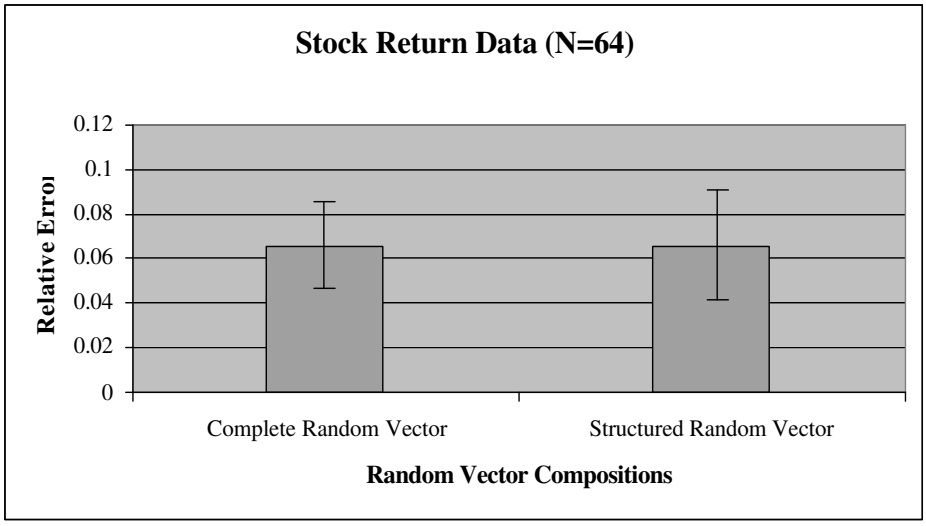


Figure 3.3: Real pairwise distance, estimated sketch distances for 64 random vectors, and estimated sketch distances for 64 structured random vectors.

per datum. For the problem sizes we consider in this thesis, the structured random vector approach is faster, though in principle it must be weighed against the small loss in accuracy.

3.4.2 Partitioning Sketch Vectors

In many applications, sketch vectors are of length up to 60. (In such a case, there are 60 random vectors to which each window is compared and the sketch vector is the vector of the dot products to those random vectors). Multi-dimensional search structures don't work well for more than 4 dimensions in practice [87]. Comparing each sketch vector with every other one destroys scalability though because the runtime is then proportional to the square of the number of windows under consideration.

For this reason, we adopt an algorithmic framework that partitions each

sketch vector into subvectors and builds data structures for the subvectors. For example, if each sketch vector is of length 40, we might partition each one into ten groups of size four. This would yield ten data structures. We then combine the "closeness" results of pairs from each data structure to determine an overall set of candidate correlated windows.

Note that we use correlation and distance more or less interchangeably because one can be computed from the other once the data is normalized. Specifically, Pearson correlation is related to Euclidean distance as follows:

$$D^2(\hat{x}, \hat{y}) = 2(1 - corr(x, y))$$

Here $\hat{x}$ and $\hat{y}$ are obtained from the raw time series by computing $\hat{x} = \frac{x - avg(x)}{var(x)}$.

3.4.3 Algorithmic Framework

Given the idea of partitioning sketch vectors, we have to discuss how to combine the results of the different partitions. This introduces four parameters, as we will see. Suppose we are seeking points within some distance d in the original time series space.

- Partition each sketch vector s of size N into groups of some size g.
- The ith group of each sketch vector s is placed in the ith grid structure of dimension g (In Figure 3.4 $g = 2$).
- If two sketch vectors $s1$ and $s2$ are within distance $c \times d$ in more than a fraction f of the groups, then the corresponding windows are candidate highly correlated windows and will be checked exactly.

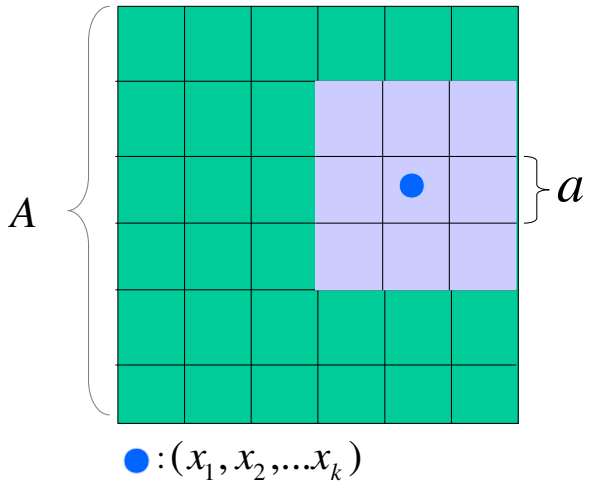


Figure 3.4: Grid Structure: Assume a set of data points in a 2D space, where a 2-dimensional orthogonal regular grid is super-imposed on this space. In practice the indexed space is bounded. Without loss of generality, we assume each dimension is contained in [0,1] (See Appendix C for a general proof of grid size bound). Let the spacing of the grid be a. The indexing space, a 2-dimensional square with diameter 1 is partitioned into $\lceil \frac{1}{a} \rceil^2$ small cells. Each cell is a 2-dimensional square with side length a. All the cells are stored in a 2-dimensional array in main memory. In such a main memory grid structure, we can compute the cell to which a point belongs. Let us use (c_1, c_2) to denote a cell that is the c_1^{th} in the first dimension and the c_2^{th} in the second dimension. A point p with coordinates x_1, x_2 is within the cell $(\lfloor \frac{x_1}{a} \rfloor, \lfloor \frac{x_2}{a} \rfloor)$. We say that point p is mapped to that cell. This can be easily extended to k-dimensions.

3.4.4 Combinatorial Design

This framework eliminates the curse of dimensionality by making the groups small enough that multi-dimensional search structures (even grid structures)

can be used. The framework also introduces the challenge of optimizing the settings of four parameters: the length N of the sketch vector, the size g of each group, the distance multiplier c, and the fraction f.

Our optimization goal is to achieve extremely high recall (above 0.95) and reasonable precision (above 0.02). We are satisfied with a fairly low precision because examining 50 times the necessary pairs on the raw data is much better than examining all pairs, as we show later in our experiments. Increasing the size of the sketch vector improves the accuracy of the distance estimate and therefore the precision but also increases the search time. The tradeoff between extra filtering cost with larger N and extra verificatn cost with smaller precision should be made. In our experiments, accuracy improved noticeably as the size increased to about 60; beyond that, accuracy did not improve much. Larger group sizes also improve accuracy, but increase the search time. A typical set of possible parameter values therefore would be:

Size of Sketch (N): 30, 36, 48, 60
Group Size (g): 1, 2, 3, 4
Distance Multiplier (c): 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3
Fraction (f): 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1

As we will see, every possible selection of parameter values requires a test on many pairs of windows (typically a few million) in order to get a robust set of parameters. For this reason, we would like to avoid testing all possible settings (1,560 in this example). Instead, we use combinatorial design.

Combinatorial design is effectively a disciplined sampling approach with some guarantees [27]. The key idea of n-factor combinatorial design is that the

a1	a2	a3	a4
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1
0	0	1	1
1	1	0	0

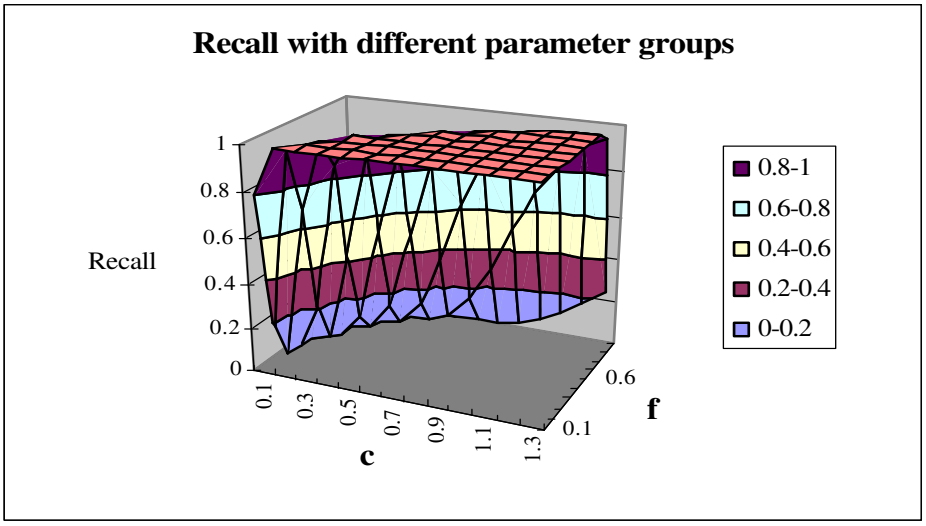
Table 3.1: An example of two-factor combinatorial design.

tests will cover all n-way combinations of parameters. For concreteness, two-factor combinatorial design requires that for every pair of parameters (a.k.a. factors) $p1$ and $p2$ and for every value $v1$ from $p1$ and $v2$ from $p2$, some experiment will test $p1.v1$ and $p2.v2$ together. This property is not the same as exhaustive search, of course. For example, if there were 4 binary variables, one possible two factor combinatorial design would be the one found in table 3.1.

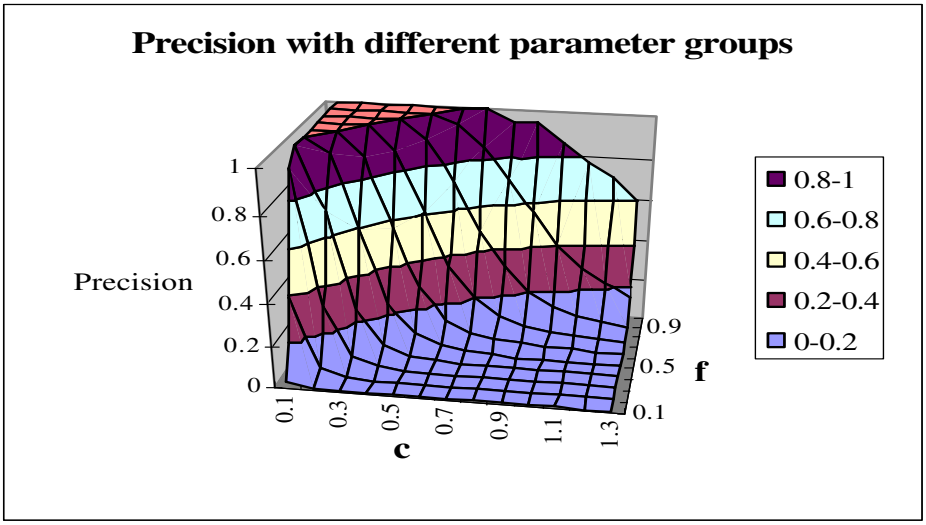
In our example, a two-factor combinatorial design would reduce the number of experiments from 1,560 to only 130.

When faced with a sampling proposal like combinatorial design, one must ask whether some global optimum is missed through sampling. This could be a particularly significant issue if small changes in parameter values could yield large changes in time or quality of result. We call such a situation *parameter discontinuity* and the hoped-for opposite *parameter continuity*. Fortunately, across a wide variety of data sets, our framework appears to enjoy parameter continuity. Figure 3.5 illustratess this property.

Table 3.2 demonstrates that the best value found by combinatorial design



(a) Recall



(b) Precision

Figure 3.5: Parameter continuity for recall and precision

Data title	$prec_{cd}^{mean}$	$prec_{cd}^{std}$	$prec_{ex}^{mean}$	$prec_{ex}^{std}$
spot_exrates	0.18	0.02	0.2	0.03
cstr	0.16	0.02	0.18	0.03
foetal_ecg	0.22	0.01	0.25	0.008
evaporator	0.007	0.0001	0.007	0.0001
steamgen	0.32	0.02	0.34	0.01
wind	0.001	0.001	0.001	0.0001
winding	0.05	0.02	0.06	0.02
buoy_sensor	0.02	0.003	0.03	0.005
eeg	0.12	0.03	0.14	0.07
price	0.11	0.04	0.14	0.03
return	0.008	0.002	0.009	0.001

Table 3.2: Combinatorial design vs. exhaustive search over a parameter search space of size 1,560.

is close to that returned by exhaustive search. In the table, we have listed the precision of the best parameters for each data set after doing the bootstapping tests. Here "best" is defined as those having average recall ≥ 0.99 and standard deviation for recall≤ 0.001 as well as reasonably high precision and low standard deviation for precision.

In fact, we use parameter continuity in a second way: the c and f values may take any real value. For the purposes of sampling them with combinatorial design, however, we make them discrete. Once we find a good set of discrete values, we may want to find better values by exploring a local neighborhood around that good set. For example, if the optimal parameter set has $c =$

0.7, then we will search $0.63, 0.64, 0.65, 0.66, 0.67, \cdots, 0.74, 0.75, 0.76, 0.77$. We call this local neighborhood search *refinement*. To see whether separating the refinement step from the initial parameter search works well, we tested whether an exhaustive search on a dense parameter space (c values having two digits of precision in our case) would have yielded a substantially different result from a combinatorial design followed by refinement approach.

Table 3.3 shows that the two approaches (combinatorial design then refinement vs. exhaustive search over the dense parameter space) achieve very similar results in terms of the precision (i.e. its mean and std). The table also shows the relationship between precision and the number of pairs whose distances are slightly greater than the target distance. Population Ratio 1 is the number of pairs with distance value 1.1 $\times$ desired distance divided by the number of pairs within the desired distance. Population Ratio 2 is the ratio based on 1.2 $\times$ the desired distance. When these ratios are high, the precisions for both the combinatorial design and exhaustive approaches are low.

3.4.5 Bootstrapping To Determine Parameter Robustness

Optimizing parameter settings for one data sample may not yield good parameter settings for others. For example, suppose that we find the optimal parameter settings for stock return data over the first month. Will those settings still work well for a later month? Without further assumptions we cannot answer this, but we can get an idea by using bootstrapping [40].

The goal of bootstrapping is to test the robustness of a conclusion on a sample data set by creating new samples from the initial sample with replacement.

Data title	$prec_{cd}^{mean}$	$prec_{cd}^{std}$	$prec_{ex}^{mean}$	$prec_{ex}^{std}$	Popu_ratio 1	Popu_ratio 2
spot_exrates	0.20	0.02	0.22	0.04	1.19	1.64
cstr	0.18	0.01	0.19	0.03	1.3	2.05
foetal_ecg	0.23	0.04	0.26	0.06	1.12	1.43
evaporator	0.007	0.0001	0.007	0.0001	15.81	102.26
steamgen	0.35	0.01	0.36	0.02	1.13	1.44
wind	0.002	0.001	0.002	0.001	14.8	564.87
winding	0.07	0.02	0.07	0.02	1.21	2.19
buoy_sensor	0.03	0.005	0.03	0.003	1.52	3.24
eeg	0.15	0.04	0.15	0.03	1.18	1.74
price	0.13	0.02	0.15	0.02	1.32	2.18
return	0.008	0.001	0.009	0.001	18.1	117.75

Table 3.3: Combinatorial design then refinement vs. exhaustive search over the dense parameter space. Population ratios indicate the number of pairs at 1.1 and 1.2 times the target distance divided by the number at the desired distance.

In our case, the conclusion is that a given parameter setting with respect to recall and precision shows good behavior. To be concrete, suppose we take a sample S of one million pairs of windows. A bootstrapped sample would consist of one million pairs drawn from S with replacement. Thus the newness of a bootstrapped sample comes from the duplicates.

We use bootstrapping to test the stability of a choice of parameters. After constructing each bootstrapped sample, we check the recall and precision of that sample given our chosen parameter settings. Provided the mean recall over all bootstrapped samples less the standard deviation of the recall is greater than our threshold (say 0.95) and the standard deviation for precision is low, then the parameter setting is considered to be good. This admittedly heuristic criterion for goodness reflects the idea that the parameter setting is "usually good" (under certain normality assumptions, roughly 3/4 of the time).

Otherwise, we take a bigger sample, perform combinatorial design, optimize, bootstrap, and do the standard deviation test again.

3.5 The Issues in Implementation

One aspect of the sketch computation we have yet to explain is how to update the sketch within a basic window in an online manner. The challenge comes from the normalization of the data vector. Since the data are processed every basic window (normalization and computing sketching), the sketch within a sliding window should be adjusted every time a new basic window emerges.

For instance, when a basic window X_{bw} arrives, it's normalized as follows.

$$\hat{X}_{bw} = \frac{X_{bw} - avg(X_{bw})}{var(X_{bw})}$$

Its sketch is

$$X_{sk}^{bw} = \hat{X}_{bw} \cdot R_{bw}$$

while the data normalization and its sketch within a sliding window are as follows.

$$\hat{X}_{sw} = \frac{X_{sw} - avg(X_{sw})}{var(X_{sw})}$$

$$X_{sk}^{sw} = \hat{X}_{sw} \cdot R_{sw}$$

The difficulty lies in that $avg(X_{sw})$ and $var(X_{sw})$ change over each basic window. So within a sliding window, we have to update the normalization of data vectors and thus their sketches whenever a new basic window occurs.

Now we will show that the updating is trivial and the sketch needs to be computed only once.

Update Sketch. *Given random vector $R = (R_0, R_1, \cdots, R_{bw-1})$ within a basic window, a sliding window X_{sw} and the data vectors within each basic window $X_{bw}^0, X_{bw}^1, \cdots, X_{bw}^{nb-1}$, except the last one, X_{bw}^{nb}, which has not yet been processed. Let avg_{sw} and var_{sw} denote the average and variance of the whole sliding window. To normalize the whole sliding window based on the same average and variance, the sketch of the basic window $X_{bw}^i, i = 0, 1, \cdots, nb$ will be updated as follows.*

$$X_{sk}^i = \frac{(X_{bw}^i \cdot R) - avg_{sw} \sum_{i=0}^{bw-1} R_i}{var_{sw}}$$

where

avg_{sw} will be updated by removing the oldest basic window and adding the new arrival X_{nb}, that is

$$avg_{sw} = \frac{1}{sw}(\sum_{i=0}^{nb-1} sum(X_{bw}^i) - sum(X_{bw}^0) + sum(X_{bw}^{nb}))$$

And var_{sw} *is analogous*

$$var_{sw} = EX_{sw}^2 - (EX_{sw})^2$$

Here

$$EX_{sw}^2 = \frac{1}{sw}(\sum_{i=0}^{nb-1} sum(X_{bw}^i{}^2) - sum(X_{bw}^0{}^2) + sum(X_{bw}^{nb2})$$

We need to maintain only $\sum_{i=0}^{nb-1} sum(X_{bw}^i)$, $\sum_{i=0}^{nb-1} sum(X_{bw}^i{}^2)$ for a sliding window and $X_{bw}^i \cdot R$, $sum(X_{bw}^i)$ and $sum(X_{bw}^i{}^2)$ for each basic window, which costs $O(1)$ for each datum and each sketch.

3.6 Experiments

Our approach has many moving parts. We use sketches, partition them into groups, and then combine the results from the groups. We use an optimization approach based on sampling (two-factor combinatorial design) of the parameter space and of the data space. None of this can be well-justified theoretically without some rather onerous assumptions.

Fortunately, we have several data sets from stock market data and from the UC Riverside repository [65] that afford us an empirical test of the method.[3]

The Hardware is a 1.6G, 512M RAM PC running RedHat 8.0. The language is K (www.kx.com).

[3]The stock data in the experiments are end-of-day prices from 7,861 stocks from the Center for Research in Security Prices (CRSP) at Wharton Research Data Services (WRDS) of the University of Pennsylvania [5]. All the other empirical data sets came from the UCR Time Series Data Mining Archive [65] maintained by Eamonn Keogh.

3.6.1 Experiment: how common is the uncooperative case?

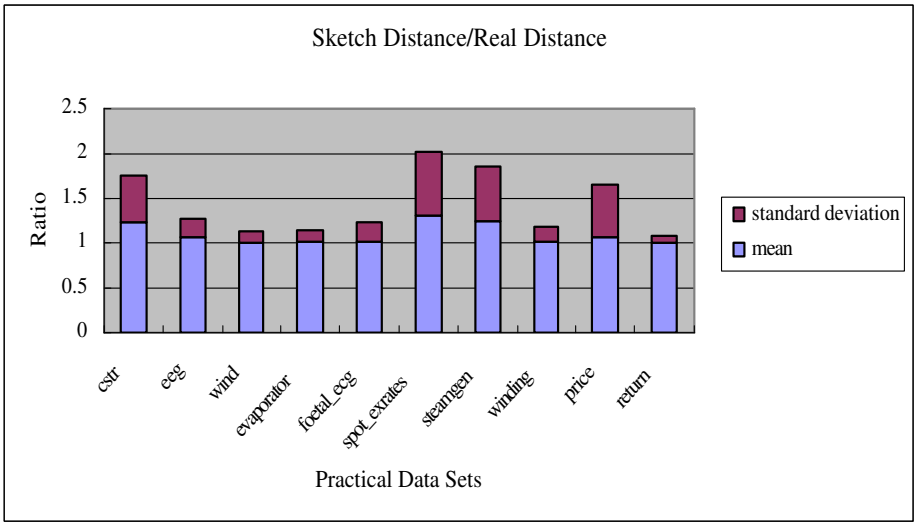
In this experiment, we took a window size of 256 across 10 data sets and tested the accuracy of the Fourier coefficients as an approximation of distance, compared with structured random vector-based sketches. Figure 3.6 shows that that the Discrete Fourier Transform-based distance performs badly on some data types while our sketch based distance works stably across all the data sets. This implies that many data sets spread their energy irregularly.

3.6.2 Experiment: How good is bootstrapping?

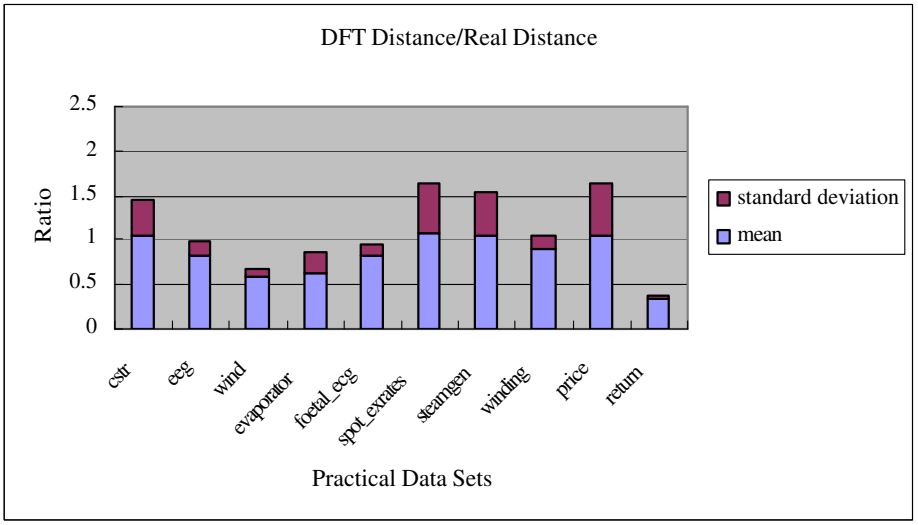
The operational claim of bootstrapping is to simulate samples across a whole data set by repeated samples from a single initial sample with replacement. In our case, we want the optimal parameters found in one sample (with bootstrapping) to meet the recall and precision thresholds in completely disjointed samples. Table 3.4 shows that for disjointed samples, the recall still meets the 99% threshold and the precision is as good or often higher than in the bootstrapping experiments on the test sample (table 3.2). So, using the parameters derived from a training sample of a data set (and confirmed by using bootstrapping) works well across that entire data set.

3.6.3 Performance Tests

The previous subsection shows that the sketch framework gives a sufficiently high recall and precision. The next question is what is the performance gain of using (i) our sketch framework as a filter followed by verification of the raw



(a) Sketch



(b) DFT

Figure 3.6: DFT distance versus sketch distance over empirical data

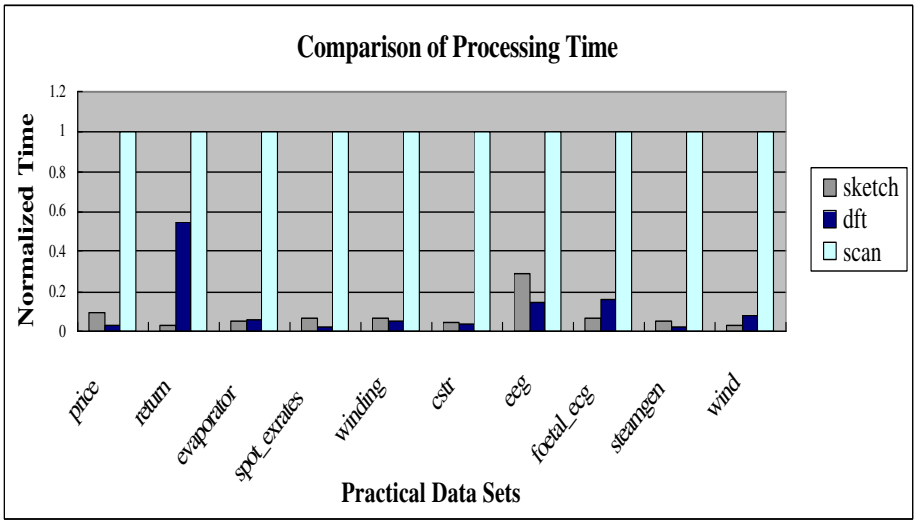


Figure 3.7: System performance over a variety of datasets. Sliding window=3616, basic window=32 and sketch size=60

data from individual windows compared with (ii) simply comparing all window pairs. Because the different applications have different numbers of windows, we take a sample from each application, yielding the same number of windows.

To make the comparison concrete, we should specify our software architecture a bit more. The multi-dimensional search structure we use is in fact a grid structure. The reason we have rejected more sophisticated structures is that we are asking a radius query: which windows (represented as points) are within a certain distance of a given point? A multi-scale structure such as a quadtree or R-tree would not help in this case. Moreover, the grid structure can be stored densely in a hash table so empty cells take up no space.

Figure 3.7 compares the results from our system, a Fourier-based approach, and a linear scan over several data sets. To perform the comparison we normalize the results of the linear scan to 1. The figure shows that both the sketch-based

Data title	rec^{mean}	rec^{std}	$prec^{mean}$	$prec^{std}$
spot_exrates	0.99	0.01	0.25	0.02
cstr	0.99	0.003	0.2	0.01
foetal_ecg	0.99	0.001	0.26	0.02
evaporator	1	0	0.007	0.0003
steamgen	1	0	0.33	0.01
wind	1	0	0.002	0.001
winding	0.99	0.007	0.1	0.01
buoy_sensor	1	0	0.03	0.008
eeg	1	0	0.13	0.02
price	0.99	0.001	0.18	0.02
return	1	0	0.008	0.002

Table 3.4: The recall and precision of disjoint sample sets

approach described here and the Fourier-based approach are much faster than the linear scan. Neither is consistently faster than the other. However as already noted, the sketch-based approach produces consistently accurate results unlike the Fourier-based one.

3.6.4 Stability of Parameter Settings Across Applications

Our general approach is to find the optimal parameter settings for each data set by sampling and bootstrapping. An interesting question is how similar these parameter settings are. If very similar, there might be a single good default

Data title	rec^{mean}	rec^{std}	$prec^{mean}$	$prec^{std}$
spot_exrates	0.99	0.007	0.47	0.06
cstr	0.99	0.01	0.26	0.02
foetal_ecg	1	0	0.26	0.06
evaporator	1	0	0.007	0.0001
steamgen	0.98	0.006	0.61	0.04
wind	0.98	0.007	0.05	0.005
winding	0.98	0.06	0.13	0.04
buoy_sensor	0.95	0.05	0.06	0.01
eeg	0.99	0.006	0.14	0.001
price	0.99	0.01	0.13	0.03
return	0.98	0.01	0.008	0.001

Table 3.5: The recall and precision of empirical data sets with the optimal parameters for price data. Recall decreases as compared with choosing the optimal parameters for each data set on its own.

to choose. This would eliminate the need for the sample-bootstrap step of our framework. In the following experiment, we take the best parameters for stock market prices and apply them to other data sets. The following table 3.5 shows the recall and precision obtained. In many cases, recall is better for the per-data set optimal parameters compared with the one-size-fits-all approach. So seeking a set of optimized parameters for each data set is probably a better idea.

Chapter 4

High Performance Incremental Matching Pursuit[1]

Imagine a scenario where a group of representative stocks must be chosen to form an index e.g. Standard and Poor's (S&P) 500. This amounts to finding a group of stocks which dominate the whole market statistically. Mathematically speaking, the "total" market vector is calculated by summing up all the stock prices of the corresponding time point weighted by their market capitalization [2]. The collection of selected stocks can capture the majority of the "energy" in the market and therefore are able to replicate the whole market. The solution lies in the decomposition strategy. One might first think of an orthogonal decomposition scheme, such as Fourier Transform, Wavelet, etc, by which the components selected at each step must be orthogonal to each other so that the whole spaces are spanned without any redundancy. In practice, however, it is difficult, if not impossible to find a group of orthogonal vectors. A greedy decomposition come

[1] This work is published in [97]

[2] Here the question is simplified for demonstration

into play then. People can choose the stock whose weighted price vector has the smallest included angle with the total market vector. Then the second stock is searched to approximate the residue between the market vector and the first stock vector, and so forth. This can be considered as an application of Matching Pursuit (MP) where the candidate pool consists of all the stocks in the market and the target vector is a weighted total market vector. Besides the financial industry, Matching Pursuit finds application in various fields. Since it was first published by Stephan Mallat and Zhifeng Zhang [76] the MP algorithm has been investigated intensively. In this section, we propose an incremental matching pursuit technique.

4.1 Problem Statement

The classic Matching Pursuit algorithm is shown in Figure 2.4. The selected group of vectors are reported at the end of the procedure. In some applications, in particular many real-time systems, the incoming time series streams are updated dynamically, which results in the necessity of running matching pursuit periodically (e.g. every 30 seconds). This idea motivated our incremental matching pursuit.

The incremental problem is to adjust the selected representative vector set V_A and the corresponding weights at each update. Formally, given a target vector and a vector pool of size n, whenever an update takes place over both the target vector and the vectors in the pool, MP is performed. Here an operation "update" on a vector is defined such that a new basic window (bw) of data is inserted to the head and older data of length bw is dropped off from the tail.

A naive method for the problem is straightforward. Whenever an update

happens, MP is run. However, recomputing MP from scratch for each new sliding window is inefficient. A better idea is to reuse the previously computed linear combination of vectors. We may expect a slight change of the approximating vector set V_A and perhaps a larger change in the weights, when the basic window is small (reminder: A basic window is a sequence of time points as defined in Figure 3.1). Whether this holds for an application is an empirical question. In our experiments on stock prices, this holds for very small basic windows only. With a relatively large basic window size (e.g. 30 time points), only the most significantly weighted approximating vectors from V_A remain important; most of the vectors vary sharply. Moreover, any perturbation may direct the approximation to a different path and result in a different set V_A.

Our solution lies in the angle space — the information given by the angle vector $(\cos\theta_1, \cos\theta_2, \ldots)$ where $\cos\theta_i = \vec{v_t} * \vec{v_i}$ for each $v_i \in V_A$.

4.2 Opportunities in Angle Space

The empirical study shows that the angle vector $(\cos\theta_1, \cos\theta_2, \ldots)$ changes only slightly over incremental vector updates. This gives us a clue to a promising, though heuristic algorithm. The basic idea is that although the approximating vectors v_i may vary a great deal between two consecutive sliding windows, every *angle* $\cos\theta_i$ of the corresponding rounds remains relatively consistent.

Therefore, instead of searching through all of V for the vector best approximating the residue or new target vector at each iteration, if a vector in the pool is found having $|\cos\theta|$ with respect to the current target vector that is larger than a certain threshold, then it is chosen. If such a vector doesn't exist, the vector with largest $|\cos\theta|$ is chosen as usual. This vector is selected as the

Given a vector pool containing n time series $V = (v_1, v_2, \cdots, v_n)$, a target vector v_t, tolerated error ϵ, and approximating vector $V_A = \emptyset$; cache $C = \emptyset$; threshold vector $T = \emptyset$. Define $\cos\theta = \vec{v_t} * \vec{v_i}$ as the cosine between v_t and a certain vector v_i in V. Vector $\vec{v} = \frac{v}{\|v\|}$.

1. Initialization. Perform MP over the initial sliding window to arrive at $v_t \approx \sum c_i v_i$. Let $C = V_A = \{v_i |$ representative vectors selected from $V\}$ and $T_j = |\cos\theta_j|$ which is calculated at j^{th} iteration.
2. while(Update) {
3. Set i=1 and $V_A = \emptyset$
4. Search the cache C. The first vector with $|\cos\theta| \geq T_i$ will be selected as representative vector v_i. If there is no such vector in C, turn to V and do the same search. If no such vector exists in V either, the vector in V whose $|\cos\theta|$ is largest is chosen to be representative vector v_i
5. Compute the residue $r = v_t - c_i v_i$ where $c_i = \frac{\|v_t\|}{\|v_i\|}\cos\theta$. $V_A = V_A \cup \{v_i\}$;
6. If $\|r\| < \epsilon$
7. Terminate, set $C = C \cup V_A$, output V_A, at next *update* go back to 3
8. Else
9. Set $i = i + 1$ and $v_t = r$, go back to 4
10. }

Figure 4.1: Incremental Matching Pursuit (MP) algorithm

representative vector and its residue with the target vector will be the new target vector for the next round of search. Here the difference from the standard algorithm resides in the search strategy: whenever a "good" vector is found, the current iteration is stopped, as opposed to the exhaustive search for an optimal approximating vector in Figure 2.4. The gain from this new algorithm is obvious: there is no need to compute the $\cos\theta$ between the target vector and *all* the vectors included in the pool V in each iteration.

One major concern with this method is the approximation power. Since the resultant vector of each search step is not optimal — in other words, not the one with largest $|\cos\theta|$ — the overall approximation power may be compromised. The empirical study shows that this is not a problem. We may carefully choose a vector of thresholds to yield results comparable to those calculated by regular MP.

Here is an example:

Given a threshold vector, say, $T = \{0.9, 0.8, 0.7, 0.6, 0.5, \ldots\}$, a target vector v_t and a candidate vector pool V, the first iteration is conducted by computing $\cos\theta$ between v_t and v in V one by one. The first vector found with $|\cos\theta| \geq 0.9$ will be selected as the representative vector in this iteration, naming it v_1. Otherwise, if there is no such vector, the vector in V with largest $|\cos\theta|$ is chosen to be the representative vector v_1. Then update the target vector by $v_t = v_t - P(v_t, v_1)$, where $P(v_t, v_1)$ is the projection of v_t onto v_1. Test the termination criterion; if it is not met, start the next iteration. The second iteration is similar to the first one — the only difference being that the threshold for comparison is 0.8. Continue the algorithm with 0.7 in the third iteration, 0.6 in the fourth iteration, etc. until the termination condition is satisfied.

Figure 4.1 gives the full pseudo-code.

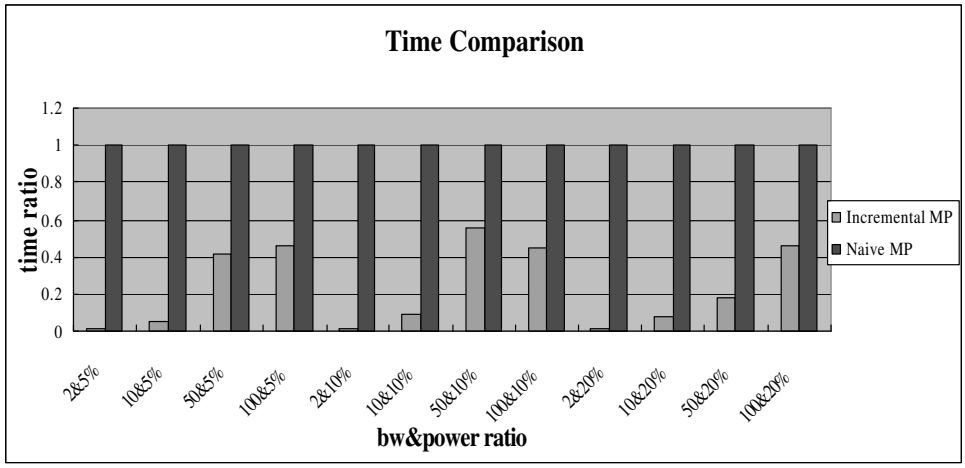
In practical applications, we apply the regular non-incremental MP to the initial sliding window. Its $|\cos\theta|$ at each iteration will be used to initialize the threshold vector T. When the approximation power of using this angle vector T gets unacceptably bad due to new data characteristics, the threshold vector is reinitialized to reflect the changes.

One bonus of this algorithm comes from the cache technique. Just as described above, the approximating vectors V_A in the present sliding window may appear with high probability in the search launched for the following sliding window, and we take advantage of this property by keeping track of a cache C for the pool V. The representative vector search is therefore performed first from the cache. If no "good" vector can be found in the cache, the rest of the vector pool is searched.

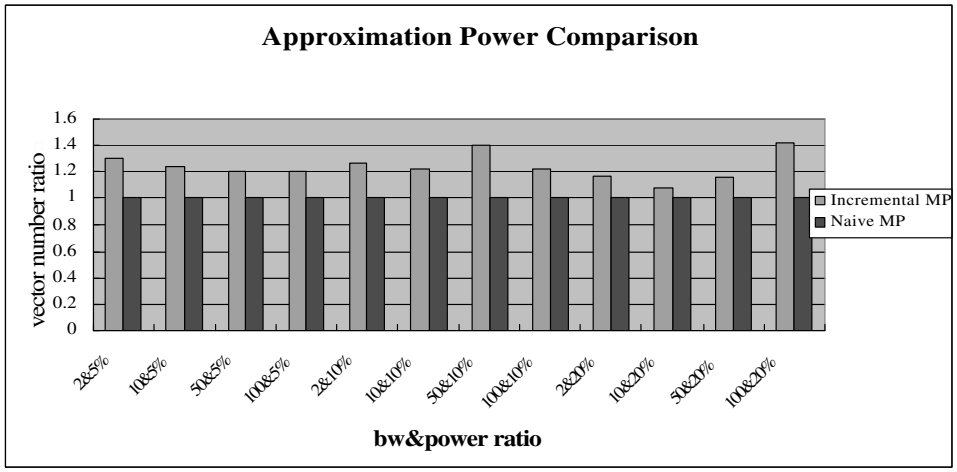
4.3 Empirical Study

The experimental data comes from the same sources as in the last section. Additional synthesized random walk time series are also used to illustrate gains in other applications. Similar results also hold for data distributions such as white noise.

Figure 4.2 compares the results from incremental MP and naive MP. The sliding window size is fixed at $sw = 200$ time points. Whenever an update event happens, both incremental and regular MP are triggered. The power ratio in the figure is defined as $\frac{\|residue\|}{\|Original\ target\ vector\|}$ (e.g. 5%). So a small power ratio entails more iterations. Performance is measured in terms of average time costs and returned approximating vector number (i.e. the average size of V_A in each sliding window). To better demonstrate the comparison, we normalize the



(a) If the basic window (bw) size is small, then an incremental approach helps significantly



(b) Incremental MP requires about 20% more vectors than the naive Matching Pursuit

Figure 4.2: Time and approximation power comparison

results of regular MP to 1.

One apparent observation in Figure 4.2(a) is the significant speed improvement when bw is small compared to the sliding window size. This substantial speedup derives largely from the vector cache. Figure 4.2(b) shows that the number of vectors required by incremental MP is no more than 1.4 times the number required by naive MP to achieve the same representation fidelity.

The experimental results suggest the potential application of incremental MP in a real-time setting where rapid response is as important as discovering a small approximating set V_A.

Chapter 5

An Implementation of the Shifted Binary Tree

Zhu and Shasha proposed a simple yet efficient algorithm to detect bursts within multiple window sizes over a time series interval. In this chapter, we first review their algorithm and then introduce an implementation of this algorithm for a physical system.

5.1 Problem Statement

Consider the following application that motivates this research. An astronomical telescope, Milagro[1] was built in New Mexico by a group of astrophysicists from the Los Alamos National Laboratory and several universities. This telescope is actually an array of light-sensitive detectors covering a pool of water about the size of a football field. It is used to constantly observe high-energy photons from the universe. When a certain number of photons are observed, the scientists can assert the existence of a Gamma Ray burst. The scientists hope

to find primordial black holes or completely new phenomena by the detection of Gamma Ray bursts. The occurrences of Gamma Ray bursts are highly variable, flaring on timescales from minutes to days. Once such a burst happens, it needs to be reported immediately. Then other telescopes can focus on that portion of sky to confirm the new astrophysical event. The data rate of the observation is extremely high. Hundreds of photons can be recorded in a second from a tiny spot in the sky.

There are also many applications in data stream mining and monitoring when people are interested in discovering time intervals with unusually high numbers of events. For example:

- In telecommunications, a network anomaly might be indicated if the number of packets lost within a certain time period exceeds a certain threshold.
- In finance, stocks with unusually high trading volumes would attract the notice of the traders (or regulators). Also stocks with unusually high price fluctuations within a short time period provide more opportunity for speculation, calling for them to be watched more closely.

Formally, given an aggregate function F (such as sum or count), the problem of interest is to discover subsequences, s of a time series stream such that $F(s) >> F(s')$ for most subsequences, s of size $|x|$. In the case of burst detection, the aggregate is the sum. If we know the duration of the time interval, we can maintain the sum over sliding windows of a known window size and sound an alarm when the moving sum is above a threshold. Unfortunately, in many cases, we cannot predict the length of the burst period. In fact, discovering that length is part of the problem to be solved.

In the above example of Gamma Ray Burst detection, a burst of photons associated with a special event might last for a few milliseconds, a few hours, or even a few days. There are different thresholds associated with different durations. A burst of 10 events within 1 second could be very interesting. At the same time, a burst that lasts longer but with a lesser density of events, say 50 events within 10 seconds, could be of interest too.

Suppose that we want to detect bursts for a time series of size n and we are interested in all n sliding window sizes. A brute-force search has to examine all the sliding window sizes and starting positions. Because there are $\Theta(n^2)$ windows, the lower bound of the time complexity is $\Theta(n^2)$. This is very slow for many applications. Fortunately, because we are interested only in those few windows that experience bursts, it is possible to design a nearly linear time algorithm. Zhu et al. [87] give a burst detection algorithm with time complexity approximately proportional to the size of the input plus the size of the output, i.e. the number of windows with bursts.

Now, let's formalize the problem as follows.

Problem: For a time series x_1, x_2,$\cdots$, x_n, given a set of window sizes w_1, w_2,$\cdots$, w_m, an aggregate function F and threshold $f(w_j)$, $j = 1, 2, \cdots, m$ associated with each window size, the problem of monitoring elastic window aggregates of the time series is to find all the subsequences of each window size such that the aggregate applied to the subsequences exceeds their window size threshold, i.e.

$\forall i \in 1 \cdots n, \forall j \in 1 \cdots m, s.t.\ F(x[i \cdots i + w_j - 1]) \geq f(w_j)$

The threshold above can be estimated from the historical data or a model of the time series. Elastic burst detection is a special case of monitoring data streams within elastic windows.

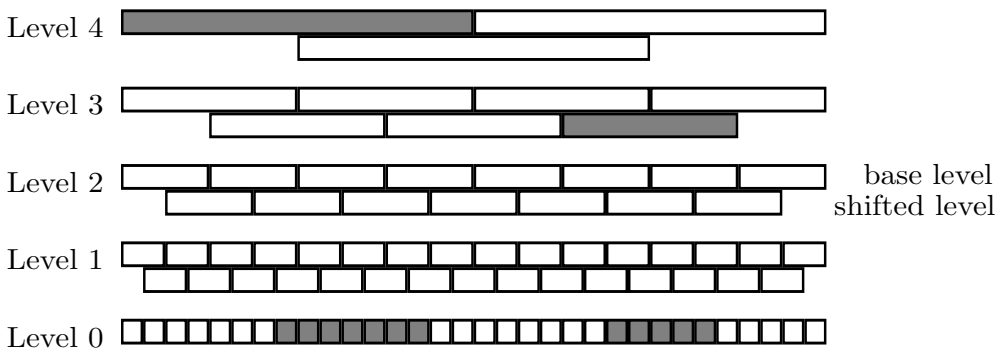


Figure 5.1: The algorithmic structure of Shifted Binary Tree(SBT)

5.2 A Brief Review of Shifted Binary Tree

In a Shifted Binary Tree (SBT) (Figure 5.1), the adjacent windows of the same level are half overlapping. We can see that the size of a SBT is approximately double that of a normal wavelet tree, because at each level, there is an additional line of windows. These additional windows provide valuable overlapping information for the time series. They can be maintained either explicitly or implicitly. If we keep only the aggregates for a traditional wavelet data structure, the aggregates of the overlapping windows at level i can be computed from the aggregates at level $i-1$ of the wavelet data structure. To build a SBT, we start from the original time series and compute the pairwise aggregates (sum) for each two consecutive data items. This will produce the aggregates at level 1. A downsampling on this level will produce the input for the higher level in the SBT. Downsampling is simply sampling every second item in the series of aggregates. In figure 5.1, downsampling will retain the upper consecutive non-overlapping windows in each level. This process is repeated until we reach the level where a single window includes every data point. Figure 5.2 gives the pseudo-code to

build a SBT. Like wavelet trees, an SBT can also be constructed in $O(n)$ time. For a subsequence starting and ending at arbitrary positions, there is always a window in the SBT that tightly includes the subsequence

Given: $x[1..n]$, $n = 2^a$

Return: Shifted Binary Tree $SBT[1..a][1..]$

$b = x$;

FOR $i = 1$ TO a //remember $a = \log_2 n$

    //merge consecutive windows and form level i of the Shifted

    //Binary Tree

    FOR $j = 1$ TO size$(b) - 1$ STEP 2

        $SBT[i][j] = b[j] + b[j + 1]$;

    ENDFOR

    //downsampling, retain a non-overlapping cover

    FOR $j = 1$ TO size$(SBT[i])/2$

        $b[j] = SBT[i][2 * j - 1]$;

    ENDFOR

ENDFOR

Figure 5.2: Algorithm to construct Shifted Binary Tree

Because for time series of non-negative numbers the aggregate sum is monotonically increasing, the sum of the time series within a sliding window of any size is bounded by the sum of its surrounding window in the Shifted Binary Tree. This fact can be used as a filter to eliminate those subsequences whose sums are far below their thresholds.

Given: time series $x[1..n]$, $n = 2^a$ Shifted Binary Tree

$SBT[1..a][1..]$, window size w, threshold $f(w)$

Return: Subsequence of x with burst

$i = [\log_2 w]$;

FOR $j = 1$ TO size($SBT[i+1]$)

    IF ($SBT[i+1][j] > f[w]$)

    //possible burst in subsequence $x[(j-1)2^i + 1..(j+1)2^i]$

    //first we compute the moving sums with window size 2^i

    //within this subsequence.

      FOR $c = (j-1)2^i$ TO $j2^i$

        y=sum($x[c..c-1+2^i]$);

        IF $y > f[w]$

          detailed search in $x[c..c-1+2^i]$

        ENDIF

      ENDFOR

    ENDIF

ENDFOR

Figure 5.3: Algorithm to search for burst

Figure 5.3 gives the pseudo-code for spotting potential sub-sequences of size w, $w \leq 2$ with sums above its threshold $f(w)$. The algorithm will search for bursts in two stages. First, the potential burst is detected at the level $i+1$ in the SBT, which corresponds to the subsequence $x[(j-1)2^i+1..(j+1)2^i]$ In the second stage, those subsequences of size 2^i within $x[(j-1)2^i+1..(j+1)2^i]$ with sum less than $f(w)$ are further eliminated. The moving sums of sliding window size 2^i can be reused for burst detection of other window sizes $w' \neq w, w' \leq 2^i$. A detail search for bursts on the surviving subsequences is then performed. A detail search in a subsequence computes the moving sums with window size w in the subsequence and verifies if there are bursts there.

5.3 MILAGRO

Various energetic processes in the universe, such as swallowing of matter by black holes, eject fast moving particles. MILAGRO (Multiple Institution Los Alamos Gamma Ray Observatory) was designed to detect high energy particles of light (photons), called "VHE gamma rays", and measure their arrival direction and energy. The arrival direction tells the observer where in the sky the particle came from, and the energy tells something about the physical mechanisms in the source which accelerated the particle.

MILAGRO (Figure 5.4) consists of a large man-made pond filled with detectors. When a VHE gamma ray enters the earth's atmosphere, it interacts producing new particles which in turn interact themselves producing even more particles. When the particles in this "shower" hit the pond, they emit light which is measured by the pond detectors. The time difference between different detectors being hit allows determination of the original particle's direction. The

Figure 5.4: MILAGRO:Multiple Institution Los Alamos Gamma Ray Observatory

number of detectors hit and how much light they measure gives an indication of the original particle's energy.

More technically, physicists partition the sky into a 1800×900 grid structure And within each cell time windows of different sizes are maintained. A burst of hits within each grid may signal an unusual physical event of interest. Whenever the number of hits within a time window exceeds the threshold, this area of sky will be examined in detail. The time windows slide forward as time goes on.

5.4 The Challenges and Our Solutions

The burst detection system used in MILAGRO possesses the following several characteristics:

- There are in total 1800×900 cells. Within each of them, burst detection is conducted;

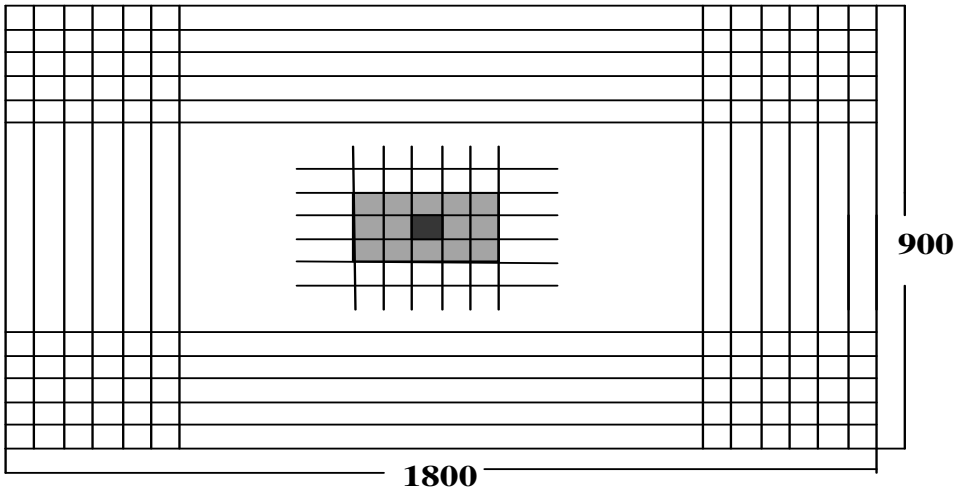


Figure 5.5: Partition the sky into 2-D grid structure. A hit at any cell will affect the surrounding cells

- Seven time window sizes are of interest: they range from $O(10^{-1})$ second to $O(10^{1})$ seconds;
- A hit on any cell will affect its surrounding cells as well. In the current resolution, whenever a cell is hit by a particle, this will amount to hitting each cell within the 5×5 square around it;
- The particle arrival is modeled as a Poisson distribution. However, the mean arrival ratio is updated dynamically. This is explained by the fact that physicists are only interested in a sudden increase of particle arrival. Therefore a background particle hit rate is maintained dynamically across the sky. Whenever a hit is observed, the probability that this is a special event is calculated by looking at the average arrival ratio from the past 5 seconds to the next 5 seconds.

The system in use detects the burst within each cell with the naive method, i.e. shift a time window of the smallest size from the beginning to the end, then examine the second window size across the time length. The whole procedure is performed for each time window size at each cell of sky. This is inefficient because the computational effort of the previous step is wasted which otherwise could be saved for the later steps.

The Shifted Binary Tree (SBT) is designed precisely to detect the event bursts within multiple time windows. In the previous sections we showed that the construction time of a half-overlapping multi-layer structure (SBT) is linear over which multi-scale windowed burst detection can be done efficiently. However, to apply SBT scheme to this physical system, we are faced with several practical challenges.

- Vast amounts of data (1800×900 time series) mean that any trivial overhead may accumulate to become a nontrivial expense. Even frequent dynamic memory allocation can aggregate to a large expense of time, not to mention frequent secondary storage access.

- Unavoidable overheads of data management. Data pre-processing (i.e. fetching and storage) requires much work. Due to the huge data size, even one pass traversal over the whole data set is expensive. And we can not control the arrival of data. What we can do is to run a program which will pull the data from somewhere, then store the data in the memory and return a pointer.

- Thresholds are updated for each sky cell over time due to the different background noises as stated above. So even for the same layer of the same tree, the thresholds at different time periods are different. To guarantee no

false negatives, the thresholds over each time window needs to be chosen carefully.

We propose the following solutions.

1. Algorithmic level

   - Build the SBT tree only including those levels covering the sliding windows. For example in one cell, a hit lasts 600 seconds. Considering that the time resolution is 0.01 second, its SBT tree has to cover $60,000$ time points which amounts to 16 ($= \log_2 60000$) levels. Constructing the entire tree will be a waste if we only care about 7 time scales. Due to its intrinsic characteristic, a SBT tree can give some fringe benefits: adding one more level to a SBT tree is trivial compared to the cost devoted to adding one more level in the naive brute-force method (see the empirical study). For physicists, more window sizes mean more information.
   - Combine neighboring cells into a single super cell to save processing time. A SBT tree is constructed for this aggregated cell and its threshold is set to be the minimum of its sub-cells' threshold. If there is an alarm reported for this large bucket, go down to each small component (i.e. a two-stage search). One natural question is how large this super cell should be. The answer lies in the overhead of the extra search. If it is too large, burst detection will frequently return false positives. Since a SBT tree has to be built for this big cell, this tree will be a net waste. In our final system, we combine nine cells.

2. Software design level

- Use dense storage. The raw data is stored in a file which is an array of the coordinates and the hit time. To perform burst detection within each cell, data have to be distributed to each cell. One intuitive data structure is a three dimensional array. The first two dimensions store the X and Y coordinates and each element of the third dimension corresponds to a small time interval. A boolean variable is stored indicating if a hit is observed at that moment. For example, if the time length observed covers 50 seconds and our resolution is 0.01 seconds, this $3d$ array will be $data[1800][900][5000]$ (most of which is empty though). For example, all the hit time for cell $(1000, 400)$ is stored at $data[1000][400]$. Suppose at the time 0.00s (i.e. the first time point) a hit is received, $data[1000][400][0]$ will be 1, otherwise it will be 0. However this storage scheme is unrealistic due to its huge size. If we use a byte (unsigned char in C[1]) to store a boolean variable, the total space needed ($1800 \times 900 \times 5000 \times 1$ byte) would be more than $1KG$ Thus data cannot be stored in main memory. To avoid page faults, instead, we store the data explicitly in a one dimension array $ebuffer$, each element of which contains a hit time. The data for each sky cell is stored contiguously. In this way each sky cell only needs to keep the index of the first and last data point in this array. For example, for cell $(1000, 400)$ its time data is stored from, say 120000 to 126000 in ascending order.

- Use low-cost algorithms. Since we conduct burst detection along the

[1]A bit map will not help considering its masking cost when an individual bit is accessed.

time axis continuously, many operations can be done incrementally. For example a background noise map is updated every 0.01 seconds, and the sum of all the hits occurring in the past 5 seconds and next 5 seconds is needed to construct the new map. We don't re-sum all the hits every time a threshold is updated. Instead, adding the head (new data) and removing the tail can reduce the updating cost to $O(1)$. Some other techniques also help. For example, avoid accessing a data point far from the one being processed, since this may lead to a page faults with our sequential storage strategy. Move the computational intensive codes out of the loop, use low strength operations such as bit shift instead of *power*, late evaluation, and so on.

3. Compiler/Hardware level

   - Switch on the optimization flag of the compiler $-o$. But don't be too greedy. $-o2$ is a good tradeoff between performance and robustness.
   - Run a GNU Profiler to obtain the statistical data of the running cost at each step. This turns out to be very helpful. We found that a large amount of time was spent on memory allocation for each sky cell, which may not be a problem on other platforms. We solved this problem by reusing the memory, A 10% improvement was thus made.

Generally, the most significant improvement comes from the algorithmic optimization; searching over a SBT tree is nearly linear in the size of the time series if the event rate is not high, which is the case in this physical system. A profiler squeezes out the last bit of the performance.

5.5 Empirical Study

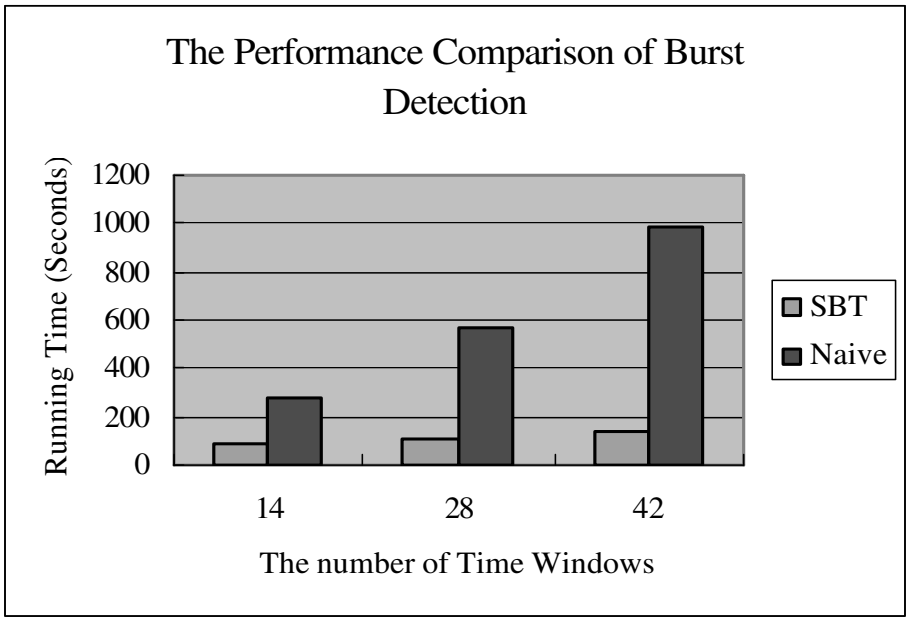
The performance of the system was improved significantly with the Shifted Binary Tree. We compare the performance for two scenarios.

In the first experiment, we compared the running time of the naive method and the SBT. Both of them run in each sky cell. Instead of monitoring seven window sizes, 14, 28 and 42 time scales are looked at. As you may see in Figure 5.6(a), the extra cost incurred by increasing the number of monitoring windows is trivial for SBT tree while the running time is roughly doubled and tripled for the naive method.

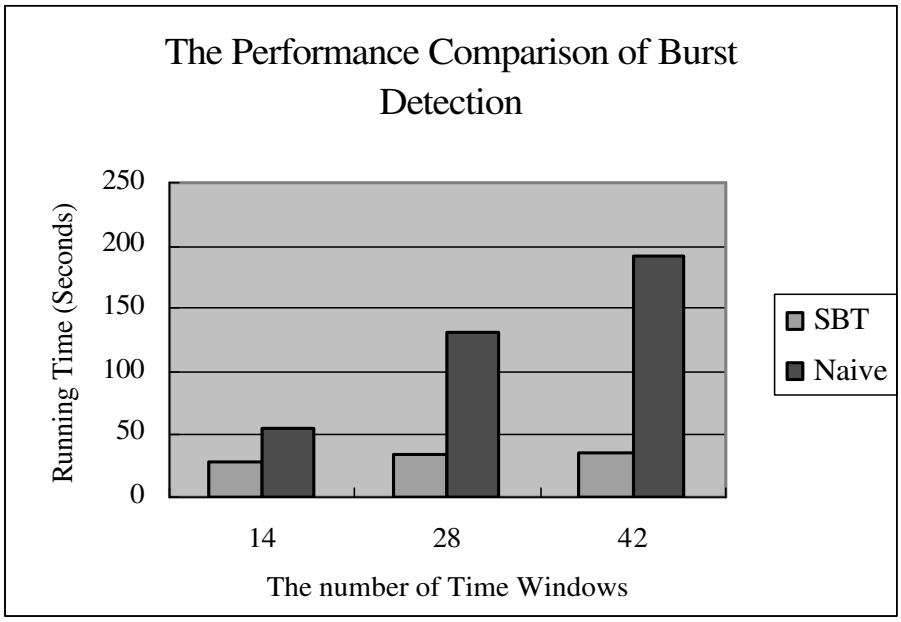
In the second experiment, we combined nine cells in a square into one to conduct a two step search. The first step is done in this 9-in-1 big cell. If no burst is observed in this combined cell, we can claim for sure there is no burst in each sub-cell. Otherwise, the second finer search is conducted in each sub-cell. Due to the sparseness of hit bursts, this coarse-to-fine strategy saves substantial computational effort. Again we tested the system further by adding more time windows. Similar results were obtained (Figure 5.6(b)).

In both of the experiments, the same alarms were reported.

We can conclude from the experiments that the Shifted Binary Tree outperforms the naive method by a factor of 2 to 10, the factors increasing with more windows.



(a) Perform the burst detection within each cell separately



(b) Combine near nine cells into one

Figure 5.6: The performance comparison between Shifted Binary Tree (SBT) and the original quadratic algorithm

Chapter 6

Conclusion

In this thesis, two high performance algorithms and one implementation for a physical system are introduced. We propose the concept of "uncooperative" data first. This is important in time series processing since spectral-based methods may fail when people deal with this type of data. This is the motivation for the sketch based statstream algorithm, which is distribution-independent and then can be used as an extension of Zhu and Shasha's DFT statstream [87]. The detailed algorithmic procedures are given and the implementation issue are also addressed. The codes are available upon request.

Incremental matching pursuit can report the approximation incrementally in every basic window. The efficiency can be improved significantly by taking advantage of the redundancy between consecutive sliding windows. We tested this algorithm both on real data sets and synthesized data sets and obtained satisfactory results.

Some algorithms published may not work in practice as well as claimed because of various practical challenges. In this thesis we also describe an implementation of Zhu and Shasha's Shifted Binary Tree [87] in the astrophysical

system MILAGRO. The challenge and our solutions are presented. The empirical study shows a substantial improvement of speed while the same accuracy (no false negatives) is attained.

6.1 Future extensions

6.1.1 Lagged Correlation

The sketch-based statstream can be easily extended to incorporate lagged correlation which is defined as the correlation between pairs of the same length but they need not be synchronized. For instance, a sliding window X_{t1} contains the data points at time points from $t1$ to $t1 + sw$ from time series X while the other sliding window Y_{t2} from time series Y covers the data points from $t2$ to $t2 + sw$. Here $dt = |t1 - t2| \geq 0$. The length of the time lag dt may depend on the user's preference.

The algorithm will be slightly more complicated than the one previously described.

A parameter ts is used as a timestamp to record the time points at which the sliding window is hashed into a grid structure. In the case of a typical nearest neighborhood query of X, the neighborhood searching step will include only those points Y with $|ts_X - ts_y| \leq T_{max}$. T_{max} is a user-defined time-delay threshold.

To free the space, the grid structure will periodically remove old data points.

6.1.2 Anomaly Detection

Anomaly detection is helpful in financial fraud monitoring, system maintenance and physical experiments towards discovering the novel events. There is not a specific definition of the anomaly. A general definition says that any unusual events are anomalous. Here we define anomaly as follows: any data points failing to have at least three supporters. The supporter, for instance, may be those points nearest to the objective point whose distance is smaller than a threshold. Our Statstream system can find the nearest neighbors for any points and is able to report anomalies in an online fashion incrementally.

Appendix A

Theoretical Probabilistic Guarantees for Recall

Whereas any approximation technique can give poor precision if most distances are near the target distance, recall can be bounded using Chernoff and Chebyshev bounds. The analysis gives a bound on the probability of a false dismissal, i.e., the sketch estimate of the distance between a pair is larger than the real distance between the pair. Intuitively, Chebyshev's inequality gives a probability of false dismissal for a group. Chernoff bounds show us how to combine the sometimes erroneous returns from the groups to get a less erroneous final result.

For concreteness, we assume that we want a recall of 99% and no more than 200 sketches. (In our experiments, we never exceeded 70.) Here is the analysis.

Let v be a length 1 vector whose length is being approximated (So $\|v\|$ is the L2 distance between two time series windows in our case.). We obtain a bound on the probability that $\|v\|$ will be overestimated.

Consider a group of $\{r_1, r_2, \cdots, r_g\}$ of g random vectors.

Lemma 1 $E[\|(v * r_1, v * r_2, \cdots, v * r_g)\|^2] = g$.

Lemma 2 Let $f(v) = \|(v*r_1, v*r_2, \cdots, v*r_g)\|^2$. Then the variance $E[(f(v) - E[f(v)])^2] \leq 2g$.

Chebyshev's inequality states that $Pr[\|X - E[X]\| \geq t\sigma_X] \leq 1/(t^2)$, where σ_X is the square root of the variance.

We apply this with $X = f(v)$ and $t = 2$ say (we set several constants in this analysis to reasonable but ultimately arbitrary values; this is the first).

Then $Pr[X \geq E[X] + 2\sigma_X] \leq 1/4$. Substituting g for $E[X]$ and $2g$ for the square of σ_X, we obtain $Pr[X \geq g(1 + 2\sqrt{2}/\sqrt{g})] \leq 1/4$.

Setting the group size g to 4 (arbitrary setting again), we get a c of $1 + \sqrt{2}$ or about 2.4.

Let the third arbitrary setting be that f, the fraction of groups that must report that the pair is within the threshold 2.4 g, is 1/2. Now we look at the influence of having multiple groups.

Lemma (Chernoff) Let $Y_1, Y_2, \cdots, Y_k$ be independent random trials with $Pr[Y_i = 1] = p_i$ and $Pr[Y_i = 0] = 1 - p_i$. Let $Y = \sum_{i=1}^{k} Y_i$ and let $\mu = E[Y] = \sum_{i=1}^{k} p_i$.
Then, for any $\delta > 0$, $Pr[Y > (1 + \delta)\mu] < ((e^{\delta})/((1 + \delta)^{1+\delta}))^{\mu}$.

For our problem, we let X_i be the estimated length for v in the ith group and Y_i is defined by

$$Y_i = \begin{cases} +1 & \text{if } X_i \geq c \times g; \\ 0 & \text{otherwise} \end{cases}$$

When $Y_i = 1$, a group has overestimated the distance. The Chernoff bounds allow us to bound how likely it is that this will happen for $(1 - f)$ of the k groups, where k will be chosen to achieve a recall of 99%. Because we had set t to 2 above, each group will be wrong with probability $p_i \leq 1/4$. The expected

value of the sum is therefore $k/4$. Because we have set f to be $1/2$, a bad case would arise when $Y > k/2$ (that is more than half the groups declare the distance to be too high). Therefore $\delta = 1$. So
$Pr[Y > k/2] < ((e^{\delta})/((1+\delta)^{1+\delta}))^{\mu} = (e/4)^{k/4}$.

This false dismissal error probability is less than 1% (equivalently, recall $\geq$ 99%) only when $(k/4) \geq 12$, so the number of groups has to be 48. Because each group contains four sketches, this gives a total of $48 \times 4 = 192$ random vectors, more than three times what we need in practice.

Notice that there are other ways to get a recall $\geq$ 99%. For example if f is $1/4$, $\delta = 2$ (more than $3/4$ of the groups must be wrong). Therefore $Pr[Y > 3k/4] < (e^2/27)^{k/4}$. This requires only fifteen groups or 60 random vectors.

Of course there are many other possibilities given $c = 2.4$ and $g = 4$. One interesting compromise is to set $f = 0.35$. In this case, the theoretical analysis indicates that the needed number of groups is 21 (84 sketches).

The empirical study indicates that this theoretical analysis is conservative for our data sets. For our empirical data sets, with these settings, a recall of 99% could be achieved with 16 or fewer sketches (4 groups).

Appendix B

Structured Random Projection for Sliding Window

Given a sliding window length sw and a basic window length bw, we show how to compute the sketches for the sliding windows starting at each timepoint. Naively, this can be done by computing the inner product of the length sw window ending at each data point with each of the random vectors. This involves much redundant computation, however, so is extremely inefficient. Instead, we will use a far more efficient approach based on the convolution between a "structured random vector" and a data vector of length sw.

Recall that sliding windows consist of an integral number of basic windows, so $sw = nb * bw$, where $nb = \frac{bw}{sw}$ is the number of basic windows within a sliding window. A random vector $\mathbf{r}_{bw}$ is constructed as follows.

$$\mathbf{r}_{bw} = (r_0, r_1, \cdots, r_{bw})$$

where

$$r_i = \begin{cases} +1 & \text{with probability } \frac{1}{2}; \\ -1 & \text{with probability } \frac{1}{2}. \end{cases}$$

To form a random vector of length sw, another random vector $\mathbf{b}$ is constructed of length nb ($= sw/bw$). We call the second vector the *control vector*.

That is,

$$\mathbf{b} = (b_0, b_1, \cdots, b_{nb})$$

where

$$b_i = \begin{cases} +1 & \text{with probability } \frac{1}{2}; \\ -1 & \text{with probability } \frac{1}{2}. \end{cases}$$

The random vector $\mathbf{r}$ for a sliding window is then built as follows.

$$\mathbf{r} = (\mathbf{r}_{bw} \cdot b_0, \mathbf{r}_{bw} \cdot b_1, \cdots, \mathbf{r}_{bw} \cdot b_{nb})$$

Please note that the fully random unit here is of size bw and the structured random vector is made up of a concatenation of nb instances of the random unit or its negation. This reduction of randomness does not noticeably diminish the accuracy of the sketch estimation as we showed in the body of the thesis.

The choice of sw and bw depends on the application under consideration, because bw is the delay before results are reported. For instance, a trader may ask which pairs of stock returns have been correlated with a value of over 0.9 for the last three hours and want the correlation information reported every 30 seconds. Here the sliding window size is 3 hours and the basic window size 30 seconds.

Example:

We are given a time series $X = (x_1, x_2, \cdots)$, sliding window size $sw = 12$, and a basic window size $bw = 4$.

If the random vector within a basic window is $\mathbf{r}_{bw} = (1, 1, -1, 1)$, the control vector $\mathbf{b} = (1, -1, 1)$, the random vector for a sliding window will be $\mathbf{r}_{sw} = (1, 1, -1, 1, -1, -1, 1, -1, 1, 1, -1, 1)$.

End of example

To obtain the sketch, we will not compute the inner product between the random vector $\mathbf{r}_{sw}$ and a given data sliding window. Instead we use a "structured convolution". To see why this might help, let's continue the example from above.

Example

$X^1_{sw} = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12})$

$X^5_{sw} = (x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16})$

The sketches are the following dot products.

$X^1_{sk} = \mathbf{r}_{sw} \cdot X^1_{sw} = b_1 \cdot \mathbf{r}_{bw} \cdot (x_1, x_2, x_3, x_4) + b_2 \cdot \mathbf{r}_{bw} \cdot (x_5, x_6, x_7, x_8) + b_3 \cdot \mathbf{r}_{bw} \cdot (x_9, x_{10}, x_{11}, x_{12})$

$= \mathbf{b} \cdot (\mathbf{r}_{bw} \cdot (x_1, x_2, x_3, x_4), \mathbf{r}_{bw} \cdot (x_5, x_6, x_7, x_8),$

$\mathbf{r}_{bw} \cdot (x_9, x_{10}, x_{11}, x_{12}))$

$X^5_{sk} = \mathbf{r}_{sw} \cdot X^5_{sw} = b_1 \cdot \mathbf{r}_{bw} \cdot (x_5, x_6, x_7, x_8) + b_2 \cdot \mathbf{r}_{bw} \cdot (x_9, x_{10}, x_{11}, x_{12}) + b_3 \cdot \mathbf{r}_{bw} \cdot (x_{13}, x_{14}, x_{15}, x_{16})$

$= \mathbf{b} \cdot (\mathbf{r}_{bw} \cdot (x_5, x_6, x_7, x_8), \mathbf{r}_{bw} \cdot (x_9, x_{10}, x_{11}, x_{12}),$

$\mathbf{r}_{bw} \cdot (x_{13}, x_{14}, x_{15}, x_{16}))$

The computation of $\mathbf{r}_{bw} \cdot (x_5, x_6, x_7, x_8)$ occurs many times. That repeated computation is eliminated in the structured convolution.

End of Example

Thus we see the computation of x^i_{sk} entails calculating nb dot products with $\mathbf{r}_{bw}$, followed by a dot product of these results with $\mathbf{b}$.

As we have seen, we want to take the dot product of $\mathbf{r}_{bw}$ with each length bw

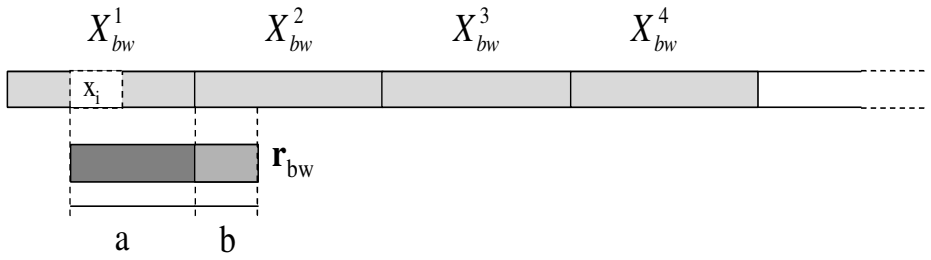


Figure B.1: Dot products with two basic windows

window of X. For efficiency, we will be computing convolutions with successive disjoint windows of X of length bw. This yield dot products of initial and final portions of $\mathbf{r}_{bw}$ with corresponding pieces of these disjoint windows, which we call partial dot products. Appropriate pairs of partial dot products then need to be added together to construct a sketch.

Figure B.1 illustrates the partial dot products computed over basic window X^1_{bw} and X^2_{bw}. nb such computations are needed to form a dot product of length sw.

Figure B.2 shows the convolution of a basic window of data with a random vector.

Figure B.3 shows the addition of corresponding pairs of partial dot products. Each line represents a sum e.g. $x_4 + (x_5 - x_6 + x_7)$.

The final step is to compute the dot product with **b** of the results of the dot products with $\mathbf{r}_{bw}$.

The full procedure is shown in Figure B.4.

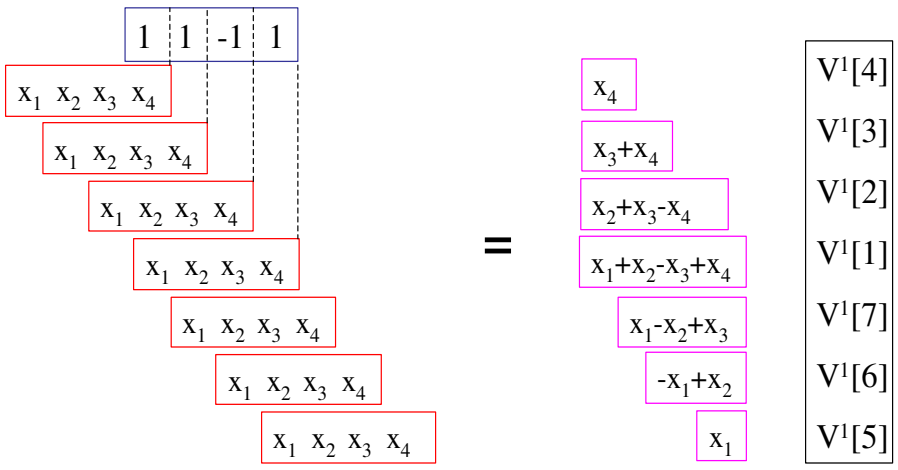


Figure B.2: Structured convolution

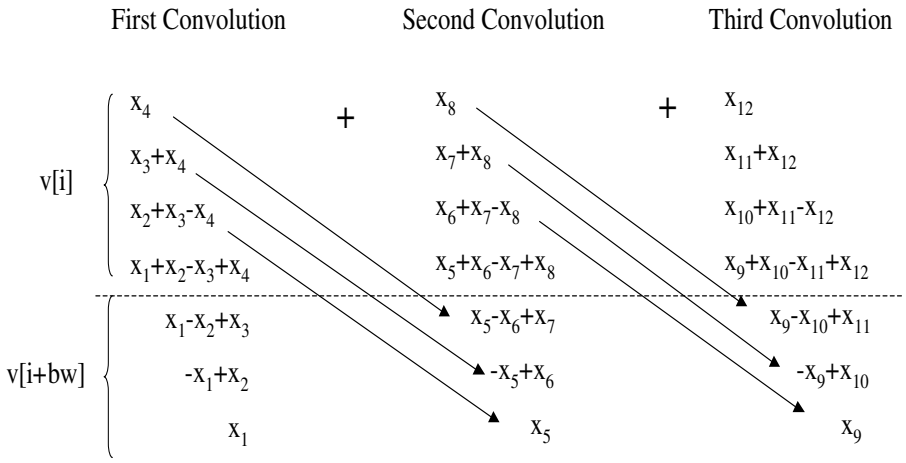


Figure B.3: Sum up the corresponding pairs

Given time series $X = (x_1, x_2, \cdots, x_n)$, random vector $\mathbf{r}_{bw} = (r_1, r_2, \cdots, r_{bw})$, and control vector $\mathbf{b} = (b_1, b_2, \cdots, b_{nb})$,

1. Convolve $\mathbf{r}_{bw}$ with $(x_1, x_2, ..., x_{bw})$ without wraparound. This yields a vector $\mathbf{v^1}$ of length $2 * bw - 1$:

$$\begin{cases} v^1[i] = \sum_{j=i}^{bw} x_j \cdot r_{j-i+1} & 1 \leq i \leq bw; \\ v^1[i+bw] = \sum_{j=1}^{i-1} x_j \cdot r_{bw+j-i+1} & 2 \leq i \leq bw. \end{cases}$$

2. Repeat *Step* 1 nb times over data chunks of length bw starting from locations $bw + 1$, $\cdots$, $nb * bw + 1$ to obtain vectors v^{bw+1}, v^{2bw+1}, $\cdots$, $v^{nb*bw+1}$.

3. Compute

$$u^{h*bw+1}[i] = v^{h*bw+1}[i] + v^{(h+1)bw+1}[i+bw]$$

   for $1 \leq i \leq bw$ and $h \geq 0$.

   Then

$$u^{h*bw+1}[i] = (x_{[h*bw+i]} \cdots x_{[(h+1)bw+i]}) \cdot (r_1, \cdots, r_{bw})$$

4. Compute the inner product between control vector $\mathbf{b}$ and each of

$$(u^{h*bw+1}[i], u^{(h+1)*bw+1}[i], \cdots, u^{(h+nb-1)*bw+1}[i])$$

   for $1 \leq i \leq bw$ and $h \geq 0$.

Figure B.4: Structured convolution procedure

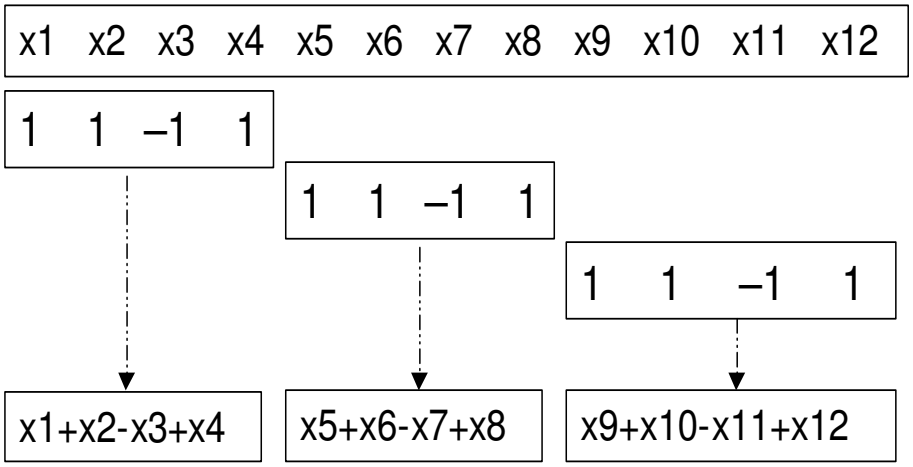


Figure B.5: Dot product of every basic window

In most cases, the point-wise computation is unnecessary since a measure such as correlation is highly correlated for two consecutive data vector. Hence the sketch may as well be computed every basic window by dot product, rather than by convolution, as shown in Figure B.5. The whole updating procedure is explained in Figure B.6.

B.0.3 Algorithmic Analysis

The structured random vector procedure eliminates the redundant computations inherent in doing a full dot product for the window of length sw starting at every timepoint. Each basic window of data is convolved with a random vector basic window only once, which will eliminate the redundancy existing in the direct convolution.

Here is the performance analysis.

1. Naive algorithm:

Given time series $X = (x_1, x_2, \cdots, x_n)$, random vector $\mathbf{r}_{bw} = (r_1, r_2, \cdots, r_{bw})$, and control vector $\mathbf{b} = (b_1, b_2, \cdots, b_{nb})$:

Stage 1: Compute the sketch of the first sliding window:

1. Compute the dot product between $\mathbf{r}_{bw}$ and $(x_1, x_2, ..., x_{bw})$, yielding v^1.
2. Repeat *Step* 1 $nb-1$ times over data chunks of length bw starting from locations $bw+1$, $\cdots$, $(nb-1)*bw+1$ to obtain v^2, v^3, $\cdots$, v^{nb}.
3. Compute the inner product between control vector $\mathbf{b}$ and

$$(v^1, v^2, \cdots, v^{nb})$$

thereby forming the sketch of the first sliding window.

Stage 2: Slide the computation forward:

1. Obtain the next data vector of length bw, $(x_{bw*h+1}, x_{bw*h+2}, ..., x_{bw*h+bw})$, and compute its dot product with $\mathbf{r}_{bw}$, yielding v^h with $h \geq nb$.
2. Compute the inner product between $\mathbf{b}$ and

$$(v^{h-nb+1}, v^{h-nb+2}, \cdots, v^h).$$

Figure B.6: Structured convolution procedure every basic window

For each random vector and each timepoint, the dot product requires $O(sw)$ integer additions.

2. New algorithm:

   If the point-wise sketch is computed for each datum and random vector, the costs consists of two parts.

   (a) $O(sw/bw)$ integer additions;
   (b) $O(\log bw)$ floating point operations (using the Fast Fourier Transform in computing the convolution).

   In the case that the sketch is computed only every basic window, the time for each datum and random vector will be:

   (a) $O(sw/bw^2)$ integer additions

Appendix C

An Upper Bound of the Grid Size

Now let's examine how to embed this sketch filter in a grid structure.

At first, we assume our parameter group is (N, g, c, f) and therefore there are totally $ng = N/g$ groups. We will assign one grid structure to each sketch group.

For each grid structure the critical parameter is the largest value A and diameter a. Now let's bound the size of A.

Upper Bound on Grid Size. *Let $X_{sk0}, X_{sk1}, \cdots, X_{skm}$ be the sketches of a data vector $X = (x_0, x_1, \cdots, x_{sw})$; then:*

$$|X_{ski}| \leq \sqrt{sw}, i = 0, 1, \cdots, m$$

Proof.

$$X_{ski} = R_i \cdot X = (r_0, r_1, \cdots, r_{sw}) \cdot (x_0, x_1, \cdots, x_{sw})$$

where $r_i = 1$ and -1 with probability $1/2$ each. Let $\hat{x}_i = |x_i|$

$$=> X_{ski}^2 \leq (\hat{x}_0 + \hat{x}_1 + \cdots + \hat{x}_{sw})^2$$

$$= \hat{x}_0^2 + \hat{x}_1^2 +, \cdots, + \hat{x}_{sw}^2 + 2\hat{x}_0\hat{x}_1 + \cdots + 2\hat{x}_i\hat{x}_j + \cdots + 2\hat{x}_{sw-1}\hat{x}_{sw}$$

$$\leq \hat{x}_0^2 + \hat{x}_1^2 + \cdots, + \hat{x}_{sw}^2 + \hat{x}_0^2 + \hat{x}_1^2 + \cdots + \hat{x}_i^2 + \hat{x}_j^2 + \cdots + \hat{x}_{sw-1}^2 + \hat{x}_{sw}^2$$

$$= sw(\hat{x}_0^2 + \hat{x}_1^2 + \cdots, + \hat{x}_{sw}^2)$$

$$= sw \cdot 1$$

$$|X_{ski}| \leq \sqrt{sw}$$

□

So $A = \sqrt{sw}$ in our case.

Bibliography

[1] Adaptive dataflow for querying streams, deep web, and beyond. http://telegraph.cs.berkeley.edu/.

[2] The aurora project. http://www.cs.brown.edu/research/aurora/.

[3] Maids overview. http://maids.ncsa.uiuc.edu/about/index.html.

[4] Niagara query engine. http://www.cs.wisc.edu/niagara.

[5] Wharton research data services (wrds). http://wrds.wharton.upenn.edu/.

[6] Understanding alraworks panorama's metric correlation engine. Technical report, Altaworks Corporation, 2004.

[7] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. SIGMOD, 1999.

[8] D. Achlioptas. Database-friendly random projections. Santa Barbara, CA, May 2001. SIGMOD.

[9] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity searching in sequence databases. In *Proceedings of the 4th International Conference of Foundations of Data organization and Algorithms (FODO)*, pages 69–84, Chicago, Illinois, MN, 1993. Springer Verlag.

[10] A. Arasu and J. Widom. Resource sharing in continuous sliding-window aggregates. VLDB, 2004.

[11] R. I. Arriaga and S. Vempala. Algorithmic theories of learning. *Foundations of Computer Science*, 1999.

[12] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. Madison, Wisconsin, 2002. ACM SIGMOD-PODS.

[13] B. Babcock, S. Babu, and M. D. R. Motwani. Chain: Operator scheduling for memory minimization in data stream systems. SIGMOD, 2003.

[14] S. Berchtold, C. Bohm, D. A. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. PODS, 1997.

[15] S. Berchtold, K. D., and K. H. P. The x-tree: An index structure for high-dimensional data. VLDB, 1996.

[16] D. J. Berndt and J. Clifford. Finding patterns in time series: a dynamic programming approach. *Advances in Knowledge Discovery and Data Mining*, 1996.

[17] E. Bingham and H. Mannila. Random projection in dimensionality reduction: Applications to image and text data. KDD, 2001.

[18] K. Blinowska, P. Durka, and W. Szelenberger. Time-frequency analysis of nonstationary eeg by matching pursuit. World Congress of Medical Physics and Biomedical Engineering, August 1994.

[19] C. Bohm. Efficient indexing high-dimensional databases. Ph.D. thesis, University of Munich, Germany, 1998.

[20] C. Bohm. A cost model for query processing in high-dimensional data spaces. *ACM Trans. Database System*, 2000.

[21] C. Bohm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces-index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3), September 1995.

[22] L. Borcea, J. G. Berryman, and G. C. Papanicolaou. Matching pursuit for imaging high-contrast conductivity. *Inverse Problems*, 15:811–849, 1999.

[23] A. Borodin, R. Ostrovsky, and Y. Rabani. Subquadratic approximation algorithms for clustering problems in high dimensional spaces. *Symposium on Theory of Computing*, 1999.

[24] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. VLDB, 2000.

[25] K.-P. Chan and A. W.-C. Fu. Efficient time series matching by wavelets. ICDE, 1999.

[26] K. Cheung and Y. Chan. A fast two-stage algorithm for realizing matching pursuit. In *IEEE ICIP*, pages 431–434, Thessaloniki Greece, October 2001.

[27] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton. The combinatorial design approach to automatic test generation. *IEEE Software*, 1996.

[28] R. Cole, D. Shasha, and X. Zhao. Fast window correlations over uncooperative time series. Chicago, August, 2005. SIGKDD.

[29] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms(how to zero in). VLDB, 2002.

[30] G. Cormode, P. Indyk, N. Koudas, and S. Muthukrishnan. Fast mining of massive tabular data via approximate distance computations. ICDE, 2002.

[31] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. SIGMOD, 2003.

[32] G. Das, D. Gunopulos, and H. Mannila. Finding similar time series. Lecture Notes In Computer Science in the First European Symposium on Principles of Data Mining and Knowledge Discovery, 1997.

[33] S. Dasgupta. Learning mixtures of gaussians. In *In 40th Annual IEEE Symp. on Foundations of Computer Science*, pages 634–644, 1999.

[34] S. Dasgupta and A. Gupta. An elementary proof of the Johnson-Lindenstrauss Lemma. Technical Report TR-99-006, Berkeley, CA, 1999.

[35] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM*, 31(6), September 2002.

[36] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for information Science*, 41(6), 1990.

[37] E. Drinea, P. Drineas, and P. Huggins. A randomized singular value decomposition algorithm for image processing. Panhellenic Conference on Informatics (PCI), 2001.

[38] P. Drineas, E. Drinea, and P. S. Huggins. *An Experimental Evaluation of a Monte-Carlo Algorithm for Singular Value Decomposition in Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[39] P. Durka and K. Blinowska. Analysis of eeg transients by means of matching pursuit. *Ann.Biomed.Engin.*, 23:608–611, 1995.

[40] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, 1994.

[41] L. Engebretsen, P. Indyk, and R.O'Donnell. Derandomized dimensionality reduction with applications. SODA, 2002.

[42] O. D. Escoda and P. Vandergheynst. Video coding using a deformation compensation algorithm based on adaptive matching pursuit image decompositions. In *IEEE ICIP*, pages 77–80, Barcelona, Spain, September 2003.

[43] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. Minneapolis, MN, May 1994. SIGMOD.

[44] X. Z. Fern and C. E. Brodly. Random projection for high dimensional data clustering: A cluster ensemble approach. Int. Conf. on Machine Learning, 2003.

[45] P. Frankl and H. Maehara. The johnson-lindenstrauss lemma and the sphericity of some graphs. *Journal of Combinatorial Theory B*, 1988.

[46] S. Ganguly, M. Garofalakis, and R. Rastogi. Processing data-stream join aggregates using skimmed sketches. International Conference on Extending Database Technology(EDBT), 2004.

[47] V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining very large databases. *IEEE Computer*, (8), 1999.

[48] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. SIGMOD, 2001.

[49] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. VLDB, 2001.

[50] A. Goel, P. Indyk, and K. Varadarajan. Reductions among high dimensional proximity problems. *Symposium on Discrete Algorithms*, 2001.

[51] D. Q. Goldin and P. C. Kanellakis. On similarity queries for time-series data: constraint specification and implementation. The 1st Int'l Conference on the Principles and Practice of Constraint Programming, 1995.

[52] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. SIGMOD, 2001.

[53] R. Gribonval. Fast matching pursuit with a multiscale dictionary of gaussian chirps. *IEEE Transaction on Signal Processing*, 49(5):994–1001, MAY 2001.

[54] D. Gunopulos and G. Das. Time series similarity measures. SIGKDD Tutorial, 2000.

[55] Y. Huhtala, J. Krkkinen, and H. T. Toivonen. Mining for similarities in aligned time series using wavelets. *Data Mining and Knowledge Discovery: Theory, Tools, and Technology,SPIE Proceedings Series*, 1999.

[56] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 189. IEEE Computer Society, 2000.

[57] P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. VLDB, 2000.

[58] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. 30th Annu. ACM Symposium. Theory Computing, 1998.

[59] H. Jagadish, A. O. Mendelzon, and T. Milo. Similarity-based queries. PODS, 1995.

[60] W. Johnson and J. Lindenstrauss. Extensions of lipschitz mapping into hilbert space. *Contemporary Mathematics*, 26, 1984.

[61] S. Kaski. Dimensionality reduction by random mapping. Int. Joint Conf. on Neural Networks, 1998.

[62] N. Katayama and S. Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. ACM SIGMOD, 1997.

[63] E. Keogh. Exact indexing of dynamic time warping. VLDB, 2002.

[64] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. SIGMOD, 2001.

[65] E. Keogh and T. Folias. The ucr time series data mining archive. Riverside CA. University of California - Computer Science & Engineering Department, 2002. http://www.cs.ucr.edu/∼eamonn/TSDMA/index.html.

[66] E. Keogh, M. P. K. Chakrabarti, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 2000.

[67] E. J. Keogh and M. J. Pazzani. Relevance feedback retrieval of time series data. SIGIR, 1999.

[68] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *In Proc. 29th ACM Symp. on Theory of Computing*, pages 599–608, 1997.

[69] F. Korn, H. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. SIGMOD, 1997.

[70] M. Kurimo. *Indexing audio documents by using latent semantic analysis and SOM.* 1999.

[71] E. Kushikvitz, R. Ostrovsky, and Y. Ranbani. Efficient search for approximate nearest neighbors in high dimensional spaces. STOC, 1998.

[72] C. S. Li, P. S. Yu, and V. Castelli. Hierarchyscan: A hierarchical similarity search algorithm for databases of long sequences. ICDE, 1996.

[73] R. A. K. Lin, H. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling and translation in time-series databases. VLDB, 1995.

[74] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Foundations of Computer Science*, 1994.

[75] K. Liu, H. Kargupta, and J. Ryan. Multiplicative noise, random projection, and privacy preserving data mining from distributed multi-party data. Communication, 2003.

[76] S. Mallat and Z. Zhang. Matching pursuit with time-frequency dictionary. *IEEE Transactions on Signal Processing*, 12(41):3397–3415, 1993.

[77] G. Manku, S. Rajagopalan, and B. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. SIGMOD, 1999.

[78] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. SIGMOD, 1998.

[79] Y. Moon, K. Whang, and W. Loh. Duality based subsequence matching in time-series databases. SIGMOD, 1997.

[80] F. Moschetti, L. Granai, P. Vandergheynst, and P. Frossard. New dictionary and fast atom searching method for matching pursuit representation

of displaced frame difference. In *IEEE ICIP*, pages 685–688, Rochester, NY, September 2002.

[81] T. Palpanas, M. Vlachos, E. Keogh, D. Gunopulos, and W. Truppel. Online amnesic approximation of streaming time series. ICDE, 2004.

[82] C. H. Papadimitriou, P. Raghavan, and H. Tamaki. Latent sematic indexing: A probabilitic analysis. PODS, 1998.

[83] I. Popivanov and R. Miller. Similarity search over time series data using wavelets. ICDE, 2002.

[84] D. Rafier and A. Mendelzon. Similarity-based queries for time series data. ACM SIGMOD, 1997.

[85] L. S. Singular value decomposition-a primer. Technical report, Department of Computer Science, Brown University, 1999.

[86] C. Shahabi, X. Tian, and W. Zhao. Tsa-tree: A wavelet-based approach to improve the efficiency of multi-level surprise and trend queries on time-series data. 12th International Conference on Scientific and Statistical Database Management(SSDBM), 2000.

[87] D. Shasha and Y. Zhu. *High Performance Discovery in Time Series: Techniques and Case Studies*. Springer, 2003.

[88] U. Srivastava and J. Widom. Memory-limited execution of windowed stream joins. VLDB, 2004.

[89] A. Tucker, S. Swift, and X. Liu. Variable grouping in multivariate time series via correlation. *IEEE Transactions on Systems, Man & Cybernetics*, 2001.

[90] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. SIGKDD, 2003.

[91] L. Wu, C. Faloutsos, K. P. Sycara, and T. R. Payne. Falcon: feedback adaptive loop for content-based retrieval. VLDB, 2000.

[92] Y. Wu and V. S. Batista. Quantum tunneling dynamics in multidimensional systems: A matching-pursuit description. *The Journal of Chemical Physics*, 121:1676–1680, 2004.

[93] Y. L. Wu, D. Agrawal, and A. E. Abbadi. A comparison of dft and dwt based similarity search in time-series databases. The 9th ACM CIKM Int'l Conference on Information and Knowledge Management, 2000.

[94] B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp forms. VLDB, 2000.

[95] B.-K. Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. ICDE, 1998.

[96] P. Zhang, Y. Huang, S. Shekhar, and V. Kumar. Correlation analysis of spatial time series datasets: A filter-and-refine approach. SIGKDD, 2003.

[97] X. Zhao, X. Zhang, T. Neylon, and D. Shasha. Incremental methods for simple problems in time series: algorithms and experiments. Montreal,

Canada, July, 2005. The 9th International Databases Engineering & Applications Symposium.

[98] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. Hong Kong, China, August 2002. VLDB.

[99] Y. Zhu and D. Shasha. Warping indexes with envelop transforms for query by humming. SIGMOD, 2003.

[100] Y. Zhu, D. Shasha, and X. Zhao. Query by humming - in action with its technology revealed. UCSD, June, 2003. SIGMOD.