ABSTRACT

HAN, TAE SIK. Efficient Subsequence Matching with *LCS* . (Under the direction of Assistant Professor Jaewoo Kang.)

Advances in sensors and wireless network technologies have produced many sensor network applications. In a typical setting, a large number of different types of sensors are deployed over a wide area. The sensor streams generated from individual sensors are then combined in a server node, naturally forming a multivariable time series, and then saved in a storage system. Searching and mining interesting patterns from this multivariable time series dataset is a key challenge in time series analysis.

In this paper, we will propose an efficient subsequence matching method in a single channel time series and extend the method to multivariable time series data. We propose a novel subsequence matching framework using a non-Euclidean measure, in particular, *LCS*, and a new index query scheme. The purpose of the subsequence matching is to find a query sequence in a long range of data sequences. Due to the abundance of applications, many solutions have been proposed. Virtually all previous solutions have used the Euclidean distance as the basis for measuring distance between sequences. Recent studies, however, suggest that the Euclidean distance often fails to produce proper results due to the irregularity in the data, which is not so uncommon in our problem domain. Addressing this problem, some non-Euclidean measures, such as *Dynamic Time Warping (DTW)* and *Longest Common Subsequence (LCS)* have been proposed. However, most of the previous work in this direction focused on the whole sequence matching problem where query and data sequences are of the same length.

The proposed framework is based on the Dual Match framework where data sequences are divided into a series of disjoint equi-length subsequences which are then indexed in an R-tree. We introduced a new way to compute the similarity bound in the index matching framework using *LCS*. The proposed query matching scheme that is named as multiple window sliding scheme, reduces many false alarms encountered in the previous approaches. We also developed an algorithm to skip expensive *LCS* computations by observing the warping paths.

We applied our subsequence matching framework to multivariable time series data. Multivariable stream data is ubiquitous today. Advances in sensors and wireless network technologies enable many sensor network applications, such as object tracking, surveillance, object guarding, structural integrity monitoring of large constructions, to name a few. We extended the proposed subsequence matching technique for a single channel time series to multivariable time series data. Our experimental results on 48 datasets in a single channel and 14 in a multivariable time series suggest that our approach greatly enhances the subsequence matching performance in various metrics.

Efficient Subsequence Matching with *LCS*

by

Tae Sik Han

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2007

Approved By:

Dr. Xiaosong Ma	Dr. Ting Yu
Dr. Jaewoo Kang Co-Chair of Advisory Committee	Dr. Rada Y. Chirkova Co-Chair of Advisory Committee

DEDICATION

To the past of my parents, my parents-in-law, and my wife, Hee Jin, for their sacrifice and support.

To the future of my daughter, Kaitlyn Hyoree Han, who is the source of incredible energy in my life.

BIOGRAPHY

Tae Sik Han was born in South Korea in 1970. He graduated from Yonsei University in February 1994 with a Bachelor of Science degree in both of Mathematics and Computer Science. He also earned his master's degree in Computer Science from Yonsei University. His research topic was ”Word Sense Disambiguation with two neural networks: BP and SOM.” After graduation, he worked for a system integration company as a database administrator. He developed many ETL models and processes that helped to downsize mainframe to unix-based systems. After a year, he moved to Korea Future Exchange to help initiating first future exchange house in Korea. After a year of service there, he moved to two consulting companies, KPMG and Andersen Consulting. He served many financial institutes helping them to develop new data models and data warehousing architecture. In 2001, he was selected as a recipient of the national scholarship, awarded by the Korean government, which enabled him to pursue a graduate degree in the United States. He joined the Department of Computer Science at North Carolina State University (NCSU) in Raleigh in Fall 2001. His research work was focused on discovering and solving problems in cutting-edge time series mining and data integration including subsequence matching, motif mining, and schema value matching work. Tae Sik carried out research on such databases as a graduate research assistant for Dr. Kang and as the first student member of his research group at NCSU. He also worked as a teaching assistant for the graduate and undergraduate courses of data structure, database, AI, data mining, and advanced database classes. In summer of 2006, he worked for the North Carolina state office of state auditor as a database specialist. Tae Sik devised new warehousing architecture and generated performance reports for the chief officer. Before finishing his doctoral degree, he spent a year at the SAS institute as a technical student A. He joined a project to integrate a Google Search Appliance with SAS Business Intelligence systems. He is now working for SAS as a senior software developer.

ACKNOWLEDGEMENTS

Foremost, I would like to thank my advisor, Dr. Jaewoo Kang, for his thoughtful direction and affectionate encouragement. This dissertation would not have materialized without his careful guidance and continuous support. I owe Dr. Kang every bit of credit for my research accomplishments and exciting experiences I encountered during my doctoral work at NCSU. I would like to acknowledge my advisory committee members, Drs. Rada Y. Chirkova, Xiaosong Ma, and Ting Yu for their enlightened comments and constructive suggestions related to my work.

I also would like to thank Dr. Seung-Kyu Ko for his valuable comments and advice on my dissertation work. The IT national scholarship program of the Korean Government that partially supported my graduate study is gratefully acknowledged. The last two years of study at NCSU were supported by the North Carolina Office of State Auditor and the SAS Institute. I sincerely appreciate their support which enabled the accomplishments presented in this dissertation. I would like to thank Dr. Eamonn Keogh for his papers and datasets that were very useful in experiments to validate idea. I would like to thank Dr. Lennox Superville for giving me a chance to join his project and for his invaluable advice to keep concentrating on my Ph.D. work. I would like to thank my direct manager, Craig Rubendall, at SAS for being my mentor and for placing his trust in me.I would like to thank Ms. Vilma Berg for her proofreading and advice in the final stage.

I would like to thank Pankaj Chopra and Laura for their encouragement through constructive discussions, fruitful collaborations, and endless kindness whenever I lost my way. I would like to thank Dr. Sungbo Seo for his support and advice on my research work while doing his visiting scholar research work at NCSU. I would like to thank Dr. Sangmin Kim for his thoughtful advice on every bit of my life at the university. I also would like to thank Seung-Yong Lee for his kind and sincere comments supporting my research work at NCSU. Finally, I want to thank all the previous members of the database laboratory under Dr. Kang's direction at NCSU for their heartwarming cooperation and family-like care. I miss those days when I had weekly meetings with my advisor and friends to present my progress, discuss problems, and share ideas.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Advances in sensors and wireless network technologies have given rise to many sensor network applications. In a typical setting, a large number of different types of sensors are deployed over a wide area. The sensor streams generated from individual sensors are then combined in a server node, naturally forming a multivariable time series, and then saved in a storage system. The collected multivariable time series data in the sink node is interpreted by the analysis module, and an event is matched. Searching and mining interesting patterns from this multivariable time series dataset is a key challenge in a time series analysis. The collected multivariable time series data in the sink node is interpreted by the analysis module, and an event is matched. The use of multiple channels of signals would increase the accuracy and usability of the analysis better than the use of single channel data.

Figure 1.1 shows an example of time series analysis for sleep apnea. Sleep apnea is a sleep disorder characterized by pauses in breathing during sleep [2]. Three different data streams

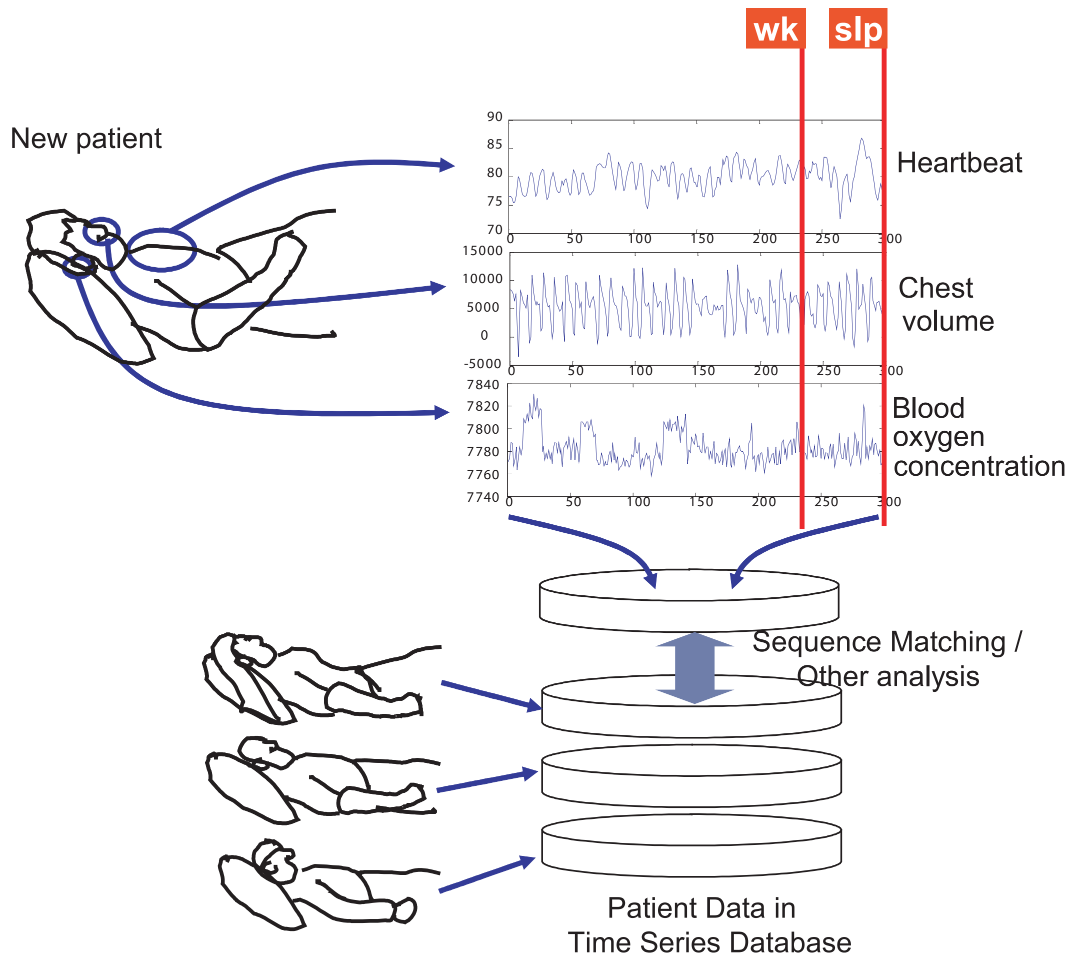

wk
slp
New patient
Heartbeat
Chest volume
Blood oxygen concentration
Sequence Matching / Other analysis
Patient Data in Time Series Database

Figure 1.1: Sleep Apnea: An Example of Time Series Analysis [1], wk :wake, slp:sleep

- heartbeat, chest volume, and blood oxygen concentration - are recorded in the time series storage system [3]. There are many open problems regarding the apnea patient datasets. One problem is to find different signals affect each other. Another problem is to determine how episodes of sleep apnea can be predicted from the preceding data recorded by other patients. Searching or comparing a new patient data against the previous dataset is a basic problem for all analysis steps. It is hard to locate an input pattern within datasets because each dataset collected from individual patients greatly varies in size and pattern.

In this paper, we will first begin with subsequence matching of a single channel time series and then extend it to subsequence matching of a multivariable time series. One of the basic problems in handling time series data is locating a pattern of interest from the long sequence of input data [4–6]. The sequence matching problem has two major components: whole sequence matching and subsequence matching. Whole sequence matching involves finding, within the dataset, all sequence entries whose lengths are equal to the query length within the similarity threshold specified by the user. For example, Figure 1.2 illustrates the way the whole sequence matching works to find out how the orientation of the palm of the Australian Sign Language signers is traced for the duration of several different words [7]. Each word of a different signer is of the same length and is searched for a given query.

Subsequence matching includes finding all subsequences in a longer data sequence that matches with the query. In Figure 1.3, which locates a query, the data sequence is explored from the beginning to the end of the data. Subsequence matching is a more general problem than the whole sequence matching problem. However, most of the previous work has focused

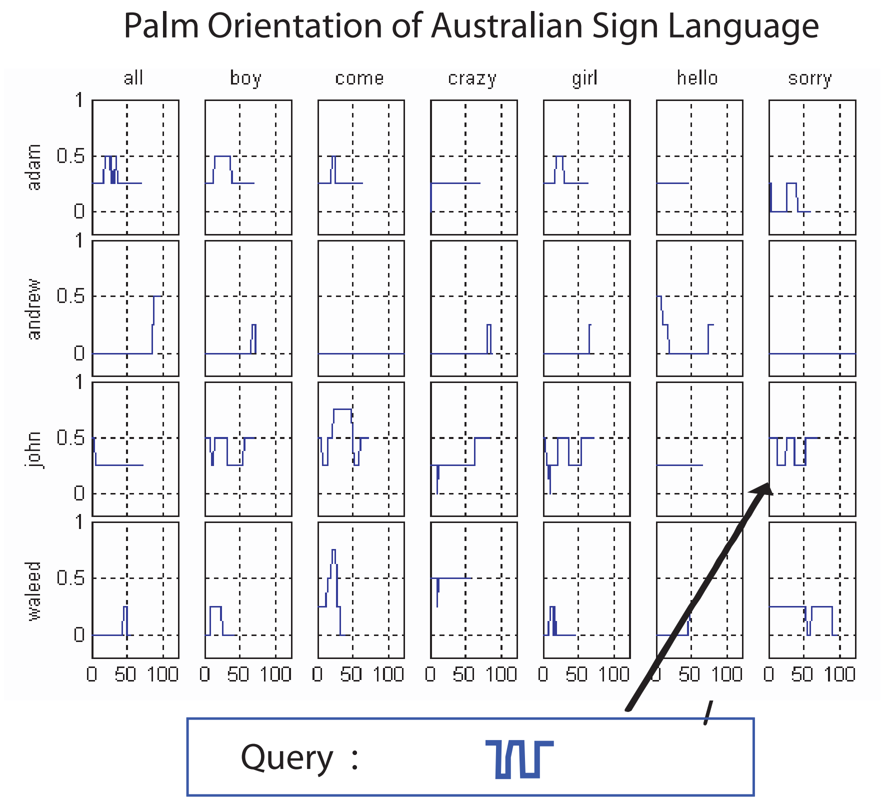


Figure 1.2: Whole Sequence Matching in a Single Channel

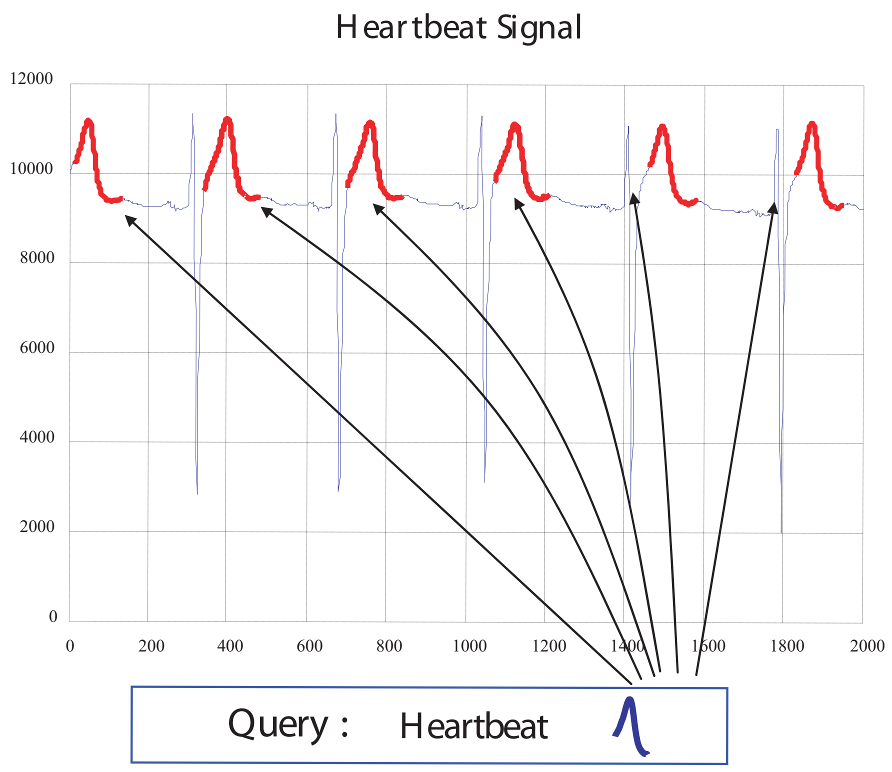


Figure 1.3: Subsequence Matching in a Single Channel

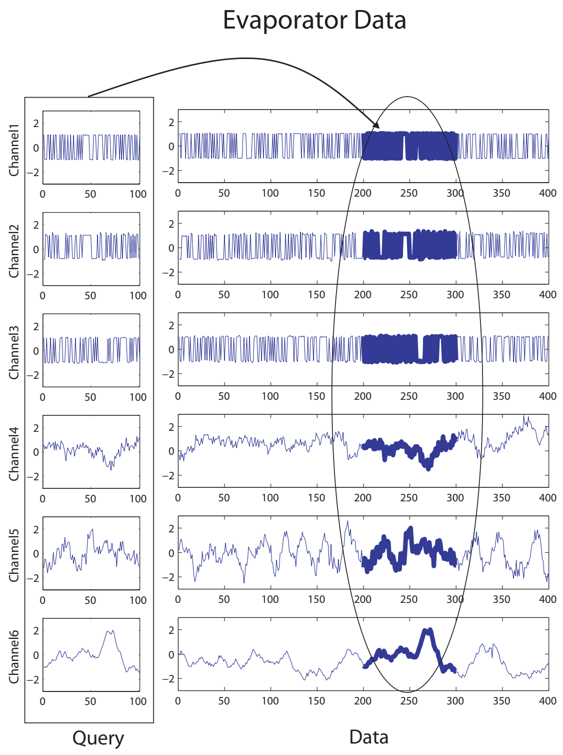


Figure 1.4: Subsequence Matching in a Multivariable Data

on the whole sequence matching problem [4, 8, 9]. While applying whole sequence matching techniques to the subsequence matching can be possible through the GEMINI [5] framework, the application is not straightforward when non-Euclidean distance measures are used. The Euclidean distance measure is sensitive to noise and, due to the irregular nature of the data in sequence applications (e.g., moving object trajectories, query-by-humming, and etc.), non-Euclidean measures are often more desirable. The non-Euclidean distance measures such as Dynamic Time Warping (*DTW*) and Longest Common Subsequence (*LCS*) address some of the problems that are characteristic of the Euclidean distance [8, 10].

In this work, we propose an efficient index searching framework for subsequence matching using *LCS*. We choose *LCS* because it is known to be more robust against the noise in the data than *DTW* [11, 12]. Furthermore, no separate normalization process is needed to overcome the difference of base unit of multivariable time series. To the best of our knowledge, no previous work has considered *LCS* in the context of subsequence matching. We make the following contributions:

- We have proposed a subsequence matching framework that employs a non-Euclidean distance measure using *LCS*. The result is a more intuitive matching performance.
- We have formally introduced criteria to prune the search space when we use a time series index with the *LCS* similarity function.
- We have introduced a new index query scheme, multiple window sliding, where several

adjacent windows are queried and aggregated in order to improve the pruning power of the index.

- We have proposed a new index search scheme that enables us to skip unnecessary similarity computations of the consecutive matching subsequences.
- We have extended the technique to subsequence matching of multivariable time series data.

Chapter 2

Background and Related Work

2.1 Notational Conventions

In order to state the problem and concepts clearly, we define some notations and terminologies in Table 2.1. In our work, we assume that a time series is a sequence of real numbers and each real number element is collected from a sensor device. A subsequence is a subset of a time series in contiguous time stamps.

Table 2.1: The Basic Notation

B	A time series data sequence, $< b_1, b_2, \ldots, b_n >$, each b_i is a real number at the i^{th} time stamp.
$\lvert B \rvert$	Length of the sequence B
B_i	A subsequence of B when B is divided into disjoint subsequences of an equal length
Q	A query sequence, usually $\lvert Q \rvert \ll \lvert B \rvert$
$B[i:j]$	A subsequence of B from time stamp i to j

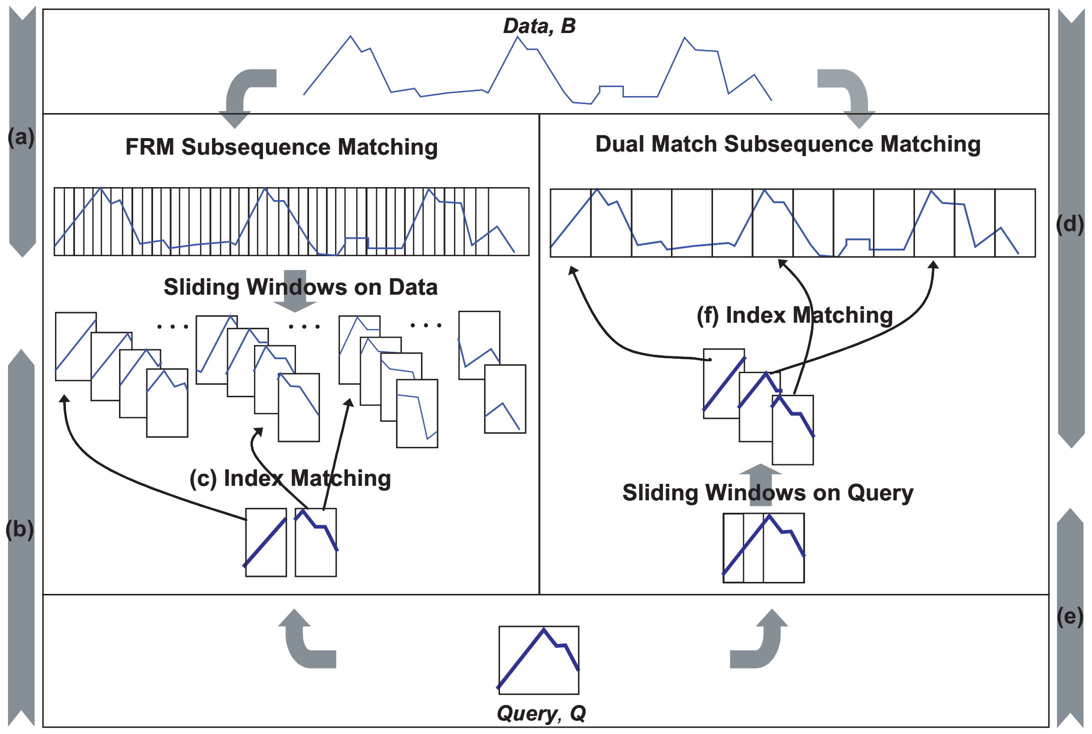


Figure 2.1: Two Subsequence Matching Frameworks

2.2 Subsequence Matching Framework (Dual Match vs. FRM)

There are two subsequence matching frameworks: FRM [5] [1] and Dual Match [6]. Both of the matching processes are illustrated in Figure 2.1. Let n be the number of data points and w be the size of an index window. In FRM, the data sequence is divided into $n - w + 1$ sliding windows. Figure 2.1 (a) shows the FRM indexing step. Every window overlaps with the next window except for the first data point. Query Q is divided into disjoint windows, Figure 2.1 (b), and each window is to be matched against the sliding windows of the data sequence, Figure 2.1 (c). In the Dual Match framework, the data sequence is divided into disjoint windows, like in Figure 2.1 (d), and part of the query in its sliding window is matched to the data indexes,

[1] It is named after its authors.

Figure 2.1 (e) and (f). Since the Dual Match does not allow any overlap of the index windows, it needs less space for an index, and consequently spends less index searching time than FRM. Through the index matching, we get a set of candidate data for the matching, and the actual similarity or distance is computed. Since the length of the data is usually very long, the Dual Match framework reduces the indexing efforts. We employ the Dual Match as our indexing scheme.

2.3 Dual Match Subsequence Matching with Euclidean Distance

Dual Match consists of the following three steps:

- First, in the indexing step, data is decomposed into disjoint windows and each window is represented by a multi-dimensional vector. They are stored in a spatial index structure like an R-tree [13] or R*-tree [14].
- Second, the query sequence is transformed into the same dimensional representations in the sliding windows. The size of the sliding window is the same as that of the index. It is proven that if the length of the query is at least twice as long as the index length, one of the sliding windows in the query is matched to a data index [6]. The index matching always returns a superset of the true matching intervals because the similarity of the index and query sliding window is always greater than or equals to the similarity of the true match.

- Lastly, depending on the positions of the matching sliding windows, whole matching intervals are determined and actual similarities are computed.

2.4 Non-Euclidean Distance *DTW* and *LCS*

Non-Euclidean similarity measures, such as *DTW* [12, 15–18] and *LCS* [10, 19], are useful when we are comparing two time series data sequences that share patterns similar in shapes but irregular in size. Both use dynamic programming algorithms to compute optimal value based on a recursive definition of the solution [16–18]. *DTW* is an algorithm used to find warping paths of the two time series by computing minimum accumulative distance. The cumulative distance of the two time series sequence is defined as below.

Definition 1 *[10] Let Q=$< q_1, q_2, ...q_n >$ be a query and B=$< b_1, b_2, ...b_n >$ be a data subsequence of time series. The cumulative distance $\rho_{i,j}$ is defined as $\rho_{i,j}(Q, B) = d(q_i, b_j) + min(\rho_{i-1,j}, \rho_{i,j-1}, \rho_{i-1,j-1})$. Then, $DTW(Q, B) = \rho_{|Q|,|B|}$.*

DTW was introduced to the time series research community by [15]. Figure 2.2 is an example of *DTW* computation. It also compares *DTW* to the Euclidean distance of two sequences. This original *DTW* algorithm has a greater time complexity than the popular Euclidean distance function. *DTW* has $O(n^2)$ time complexity when two time series are of the same length n. It is reduced $O(\delta n)$ by restricting the greedy algorithm to search minimum distance within $\pm\delta$ range of each time stamp. The distance is only computed within the diagonal band, the Sakoe-Chiba band, of the width δ in the computation matrix [20]. The restricted area for opti-

Example sequences: b : <0,0,1,1,0,0>, q : <0,0,0,1,1,0>

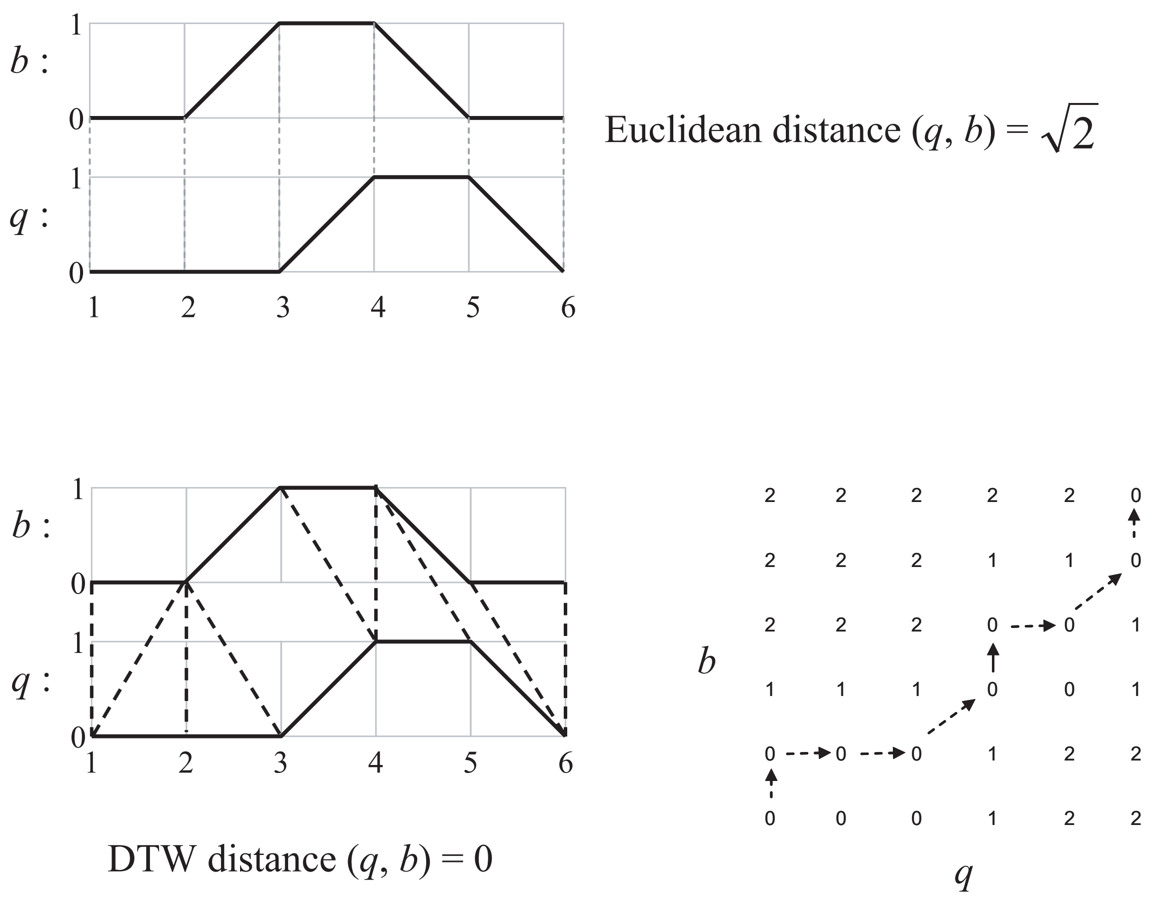


Figure 2.2: An Example of *DTW* Computation

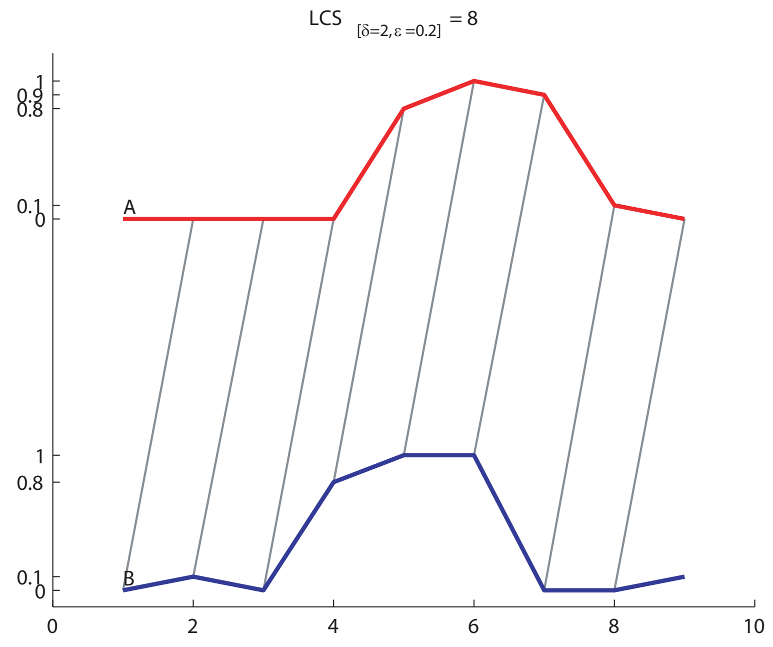
mal warping path is decided by δ. Another popular shape of the δ restriced area is the Itakura band [21]. It has a diagonal diamond shape [21]. We chose the Sakoe-Chiba band for the ease of computation. *LCS* and *DTW* share the same dynamic programming procedure to compute the optimal warping path within the δ time interval. We chose *LCS* as our distance function and the definition is given below.

Definition 2 *[10] Let Q=$< q_1, q_2, ...q_n >$ be a query and B=$< b_1, b_2, ...b_n >$ be a data subsequence of time series. Given an integer δ and a real number 0 $< \epsilon <$1, we define the cumulative similarity $\gamma_{i,j}(Q, B)$ or $\gamma_{i,j}$ as follows:*

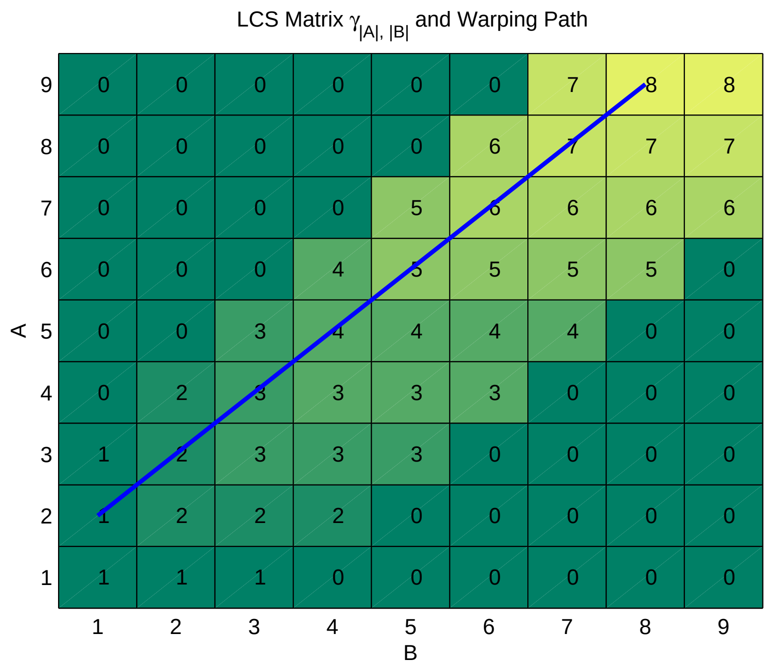
$$\gamma_{i,j} = \begin{cases} 0, & \text{if } i, j = 0 \\ 1 + \gamma_{i-1,j-1} & \text{if} |q_i - b_j| \leq \epsilon \\ & \text{and } |i - j| \leq \delta \\ max(\gamma_{i,j-1}, \gamma_{i-1,j}) & \text{otherwise} \end{cases}$$

$$LCS_{\delta,\epsilon}(Q, B) = \gamma_{|Q|,|B|}$$

LCS of the two given data sequences is computed by dynamic programming. $LCS(Q, B)$ returns an integer from 0 to $max(|Q|, |B|)$. δ is the allowable matching interval in the time dimension and ϵ is the allowable error bound in the data value dimension. Here is an example of *LCS* match for the two sequences A and B of the same length, where A = [0, 0, 0, 0, 0.8, 1, 0.9, 0.1, 0] and B = [0, 0.1, 0, 0.8, 1, 1, 0, 0, 0.1]. Figure 2.3(a) shows the *LCS* warping path. Figure 2.3(b) shows the *LCS* computation process in the *LCS* warping path matrix. It is constructed by dynamic programming of the cumulative similarity $\gamma_{|A|,|B|}$. The non-zero



(a) Sequence A, B and Warping Path



(b) Sakoe-Chiba Band and an Optimal Warping Path in *LCS* Computation Martix

Figure 2.3: An Example of *LCS* Computation

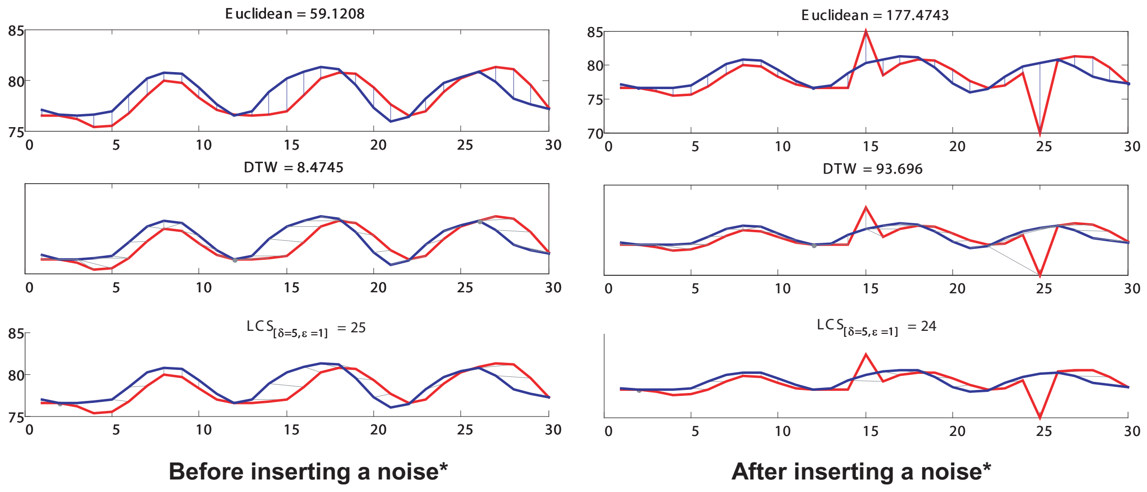


Figure 2.4: Euclidean, *DTW* and *LCS* When Noise Involved

boxes in light color in the *LCS* warping path matrix of Figure 2.3(b) represent a Sakoe-Chiba band [20].

LCS is known to be robust to the noise since it does not count the sequence values out of the range, ϵ. In Figure 2.4, three distance functions are compared by an example. *LCS* was not affected by noise as much as the other two distance functions. An alternative approach to the noise problem is to use outlier detection algorithms in the pre-processing stage. It helps subsequence matching by removing extreme values even though we use distance functions that are not strong against the noise. However, extra time is required to scan the data in order to get a correct statistics or analysis to identify outliers. [22–24]

2.5 Optimal Bounding for Index Matching

Many researchers did their best to find the optimal bound for efficient indexing of time series data [25–27]. In [8, 28], the Euclidean distance between *MBR*s of the data sequence and the query *MBE* in PAA (Piecewise Aggregate Approximation) representation is a lower bound for the *DTW* distance between the data and the query. *MBE* is a Minimum Bounding Envelope that covers all the possible matching points. Enhancing this indexing method of [8], a more efficient index matching scheme was developed by representing a query of the average values of the MBRs in [9]. [10] introduced *LCS* to the whole sequence matching problem. The number of intersecting time stamps of *MBRs* is an upper bound for the *LCS* similarity. This work, however, is proposed not just for the subsequence matching problem but also for the whole sequence matching using *LCS* .

2.6 Sequence Alignment

Subsequence matching is similar to the sequence alignment in bioinformatics in that both compare two different sequences. The sequence alignment is used to arrange two DNA or RNA sequences which consist of a small number of characters such as A,T,C and G. By identifying similar regions of two different sequences, researchers try to explain functional or evolutionary relationships of sequence owners. Depending on the number of sequences in a comparison, sequence matching is categorized into piecewise alignment and multiple sequence alignment. In the piecewise alignment, there are two approaches: global alignment and local alignment. The Needleman-Wunsch algorithm [29] is a global sequence alignment method. It is a dynamic programming algorithm that computes the similarity of two sequences. Different from *LCS* and *DTW*, it gives penalty for the unmatched regions. In local alignment, gaps of unmatched sequences reset the alignment. Matching (or alignment) is restarted whenever the algorithm encounters unmatched subsequences. The Smith-Waterman algorithm [30] is a popular local alignment method that employs negative penalty scoring system. *Basic Local Alignment Search Tool (BLAST)* [31, 32] is one of the most useful algorithms for genomists to compare amino-acid sequences or DNA sequences. *BLAST* is based on the Smith-Waterman algorithm and it is modified by heuristics to enhance computational performance.

2.7 Other Related Work

To query by content is to find a portion of the data stream similar to a given sequence in terms of a certain distance measure. [5, 9–11, 33, 34]. Classification and clustering are classic data mining problems used to label an unknown instance of data based on the previously known dataset [35, 36]. Motif discovery is a problem to identify frequently recurrent subsequences in a given data sequence [37–43]. Rule discovery is an application in the next stage of the motif or basic pattern discovery. Rule discovery finds relationships among subsequences in a given time series data [44].

Chapter 3

Subsequence Matching with *LCS* Using Dual Match Index in Single Channel

3.1 Problem Statement

The purpose of the subsequence matching is to find subsequences similar to the given query sequence. A subsequence matching framework with the Euclidean distance has been already developed as we stated in the previous section. However, to the best of our knowledge, many things have not yet been considered when applying a non-Euclidean function to the subsequence matching. We need to improve the index search performance, and we need to provide an index matching criteria that avoids expensive computations caused by non-Euclidean measures.

In order to describe what the output of the subsequence matching should be, we define

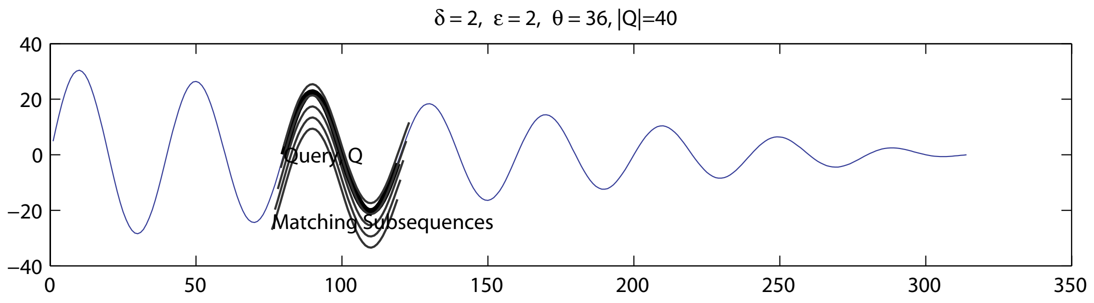


Figure 3.1: Matching Subsequences in Subsequence Matching

matching subsequences for a query sequence Q in terms of $LCS_{\delta,\epsilon}$.

Definition 3 *Let Q=*$< q_1, q_2, ...q_m >$ *be a query and B=*$< b_1, b_2, ...b_n >$ *be a data subsequence of time series. Given an integer* δ*, a real number 0* $<\epsilon<$*1 and user defined similarity threshold* θ*, we define the* ***matching subsequences****,* $M = \{B[i:j] \mid LCS_{\delta,\epsilon}(Q, B[i:j]) \geq \theta\}$

We restrict the scope of our work to searching for the longest possible matching subsequences of the length $|Q| + 2\delta$. Finding all the matches of all the lengths with a non-Euclidean measure is time-consuming. It makes sense to find the longest matching subsequences since they also include matching subsequences shorter than $|Q| + 2\delta$. It is possible to search for shorter matching subsequences after the search process for the longest ones has been completed. In Figure 3.1, all of the matching subsequences of the longest length, $|Q| + 2\delta$, are demonstrated in grey lines.

Problem : Find all matching subsequences $B[i : j]$ of the length $|Q| + 2\delta$ for data sequence B, and query Q such that the similarity $LCS_{\delta,\epsilon}(Q, B[i : j])$ is no less than s% of the $|Q|$, $\frac{s}{100}|Q|$.

Solution Road Map : Here is a road map of solutions to the problem:

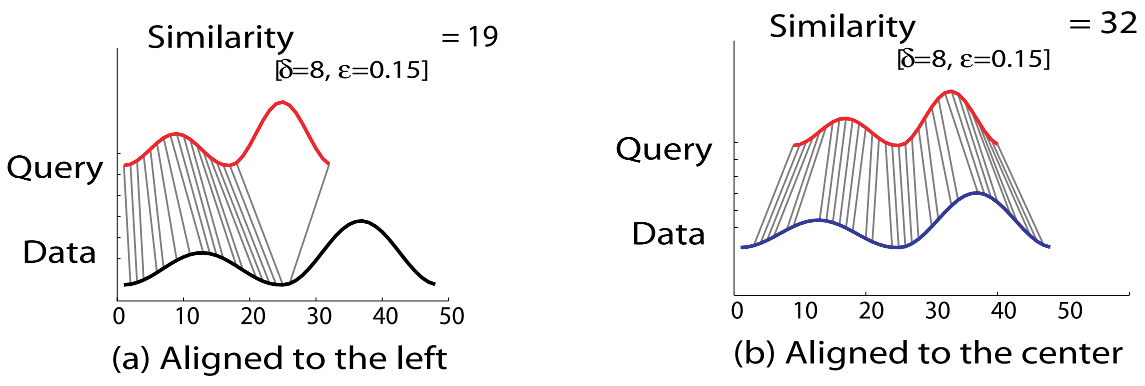


Figure 3.2: Alignment with *LCS* when $|Query|$ = 32 and $|Data|$ = 48

- Index pruning criteria (bounding value) is computed to obtain candidates with *LCS* without missing correct matches.
- When it comes to the index use, the number of candidates is decreased by summing up the index search results.
- Adjacent matching intervals are efficiently skipped by observing the *LCS* matrix, which allows more expensive similarity computations to be avoided.

3.2 Linear Search and Skipping *LCS* Computation

An intuitive approach to the subsequence matching is comparing the query sequence Q to all of the candidate subsequences of the data sequence B in a sequential manner. All the candidates are chosen by sliding a fixed size window along the data sequence.

3.2.1 Alignment in *LCS*

When we compare the query Q to a candidate data subsequence of the length $|Q| + 2\delta$, we align the query in the middle of each candidate as illustrated in Figure 3.2 (b). In the case of the whole sequence matching, alignment is not a problem since the query and data are of the same length. However, in our subsequence matching, we need to locate the query in the candidate subsequence. If we align the query to the left side of a candidate, we cannot find a correct subsequence. In Figure 3.2(a), a shorter query is not matched well to the longer data when aligned to the left. The right side of the query cannot be compared with the data since the δ is not big enough to cover the matching points of the data. A larger δ needs a heavier similarity computation. Figure 3.2 (a) shows that the query is correctly matched with the same δ when aligned to the center.

3.2.2 Skipping *LCS* Computation

We can avoid expensive similarity computations of the adjacent subsequences by observing the *LCS* warping path and the local constraint, such as the Sakoe-Chiba band. In the subsequence matching, we can think of the computation matrix as a moving window along the data sequence, as in Figure 3.3.

Let us take a look at the following example. Let's assume that $|Q| = 4$, and that a user wants to find all the subsequences whose similarity is larger than or equal to 3. Figure 3.3(a) shows that an *LCS* warping path is found and represented as a set of arrows. $LCS(Q, B[1 : 6]) = 4$. Darker cells represent the Sakoe-Chiba band. In Figure 3.3 (b), we move a sliding window

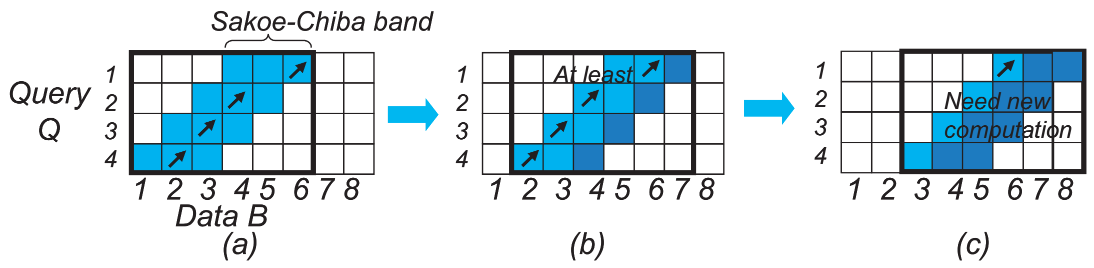


Figure 3.3: An Example of Skipping *LCS* Computation when $|Q| = 4$ and $\delta = 1$

by a time stamp. The Sakoe-Chiba band still includes the warping path. In this case, we do not have to compute the $LCS(Q, B[2:7])$ since the dynamic programming finds a maximum warping path in the Sakoe-Chiba band and the $LCS(Q, B[2:7])$ must be larger than or equal to 4. In Figure 3.3 (c), we need to compute $LCS(Q, B[3:8])$ since only one warping path remains there.

We can skip a computation of a sliding window by tracing the warping path. If we find that the Sakoe-Chiba band of the current *LCS* matrix includes the previous warping path greater than or equal to the user-defined threshold, then we can skip the *LCS* computation. The skipping goes until the Sakoe-Chiba band includes a warping path whose similarity is smaller than the user-defined threshold. It is a useful asset to be used in order to reduce the expensive similarity computation in the subsequence matching where the adjacent window usually has a similar value.

Indexing enables us to avoid a number of false candidate subsequences for matching. We compute the pruning criteria in order to choose candidate matching subsequences with *LCS*. We also propose a new framework to search for the index in this section.

3.3 Indexing

Data is divided into equi-length disjoint windows for the index. Each window is represented by a multi-dimensional vector. That is, data sequence B is divided into equi-length disjoint windows $< w_i >$. Each window w_i consists of N *MBR*s. Let N be the dimensionality of the space we want to index. An *MBR* represents a dimension. N *MBR*s for a w_i is transformed into $\overrightarrow{w_i} =< (u_{i1}, \ldots, u_{iN}), (l_{i1}, \ldots, l_{iN}) >$, where u_{ij} and l_{ij} represent the maximum and minimum values in the j^{th} interval of w_i. $\overrightarrow{w_i}$ is stored in an N dimensional R-tree. An example is illustrated in Figure 3.4 (a). In the figure, the data in the first window, $w_1 =< b_1, ..., b_9 >$ is transformed into $\overrightarrow{w_1} =< (u_{11}, u_{12}, u_{13}), (l_{11}, l_{12}, l_{13}) >$. It is stored in an R-tree as in Figure 3.4 (b).

3.4 Index Matching with *LCS*

A query Q is compared first to the index. Q is transformed into an *MBE* with the $LCS_{\delta,\epsilon}$ function as illustrated in Figure 3.4 (d). Let MBE_Q be an MBE for Q. Let the i^{th} sliding window of Q be v_i. It is transformed into $\overrightarrow{v_i} =< (\hat{u}_{i1}, \ldots, \hat{u}_{iN}), (\hat{l}_{i1}, \ldots, \hat{l}_{iN}) >$, where $\hat{u}_{ij}$ and $\hat{l}_{ij}$, respectively, are the maximum and minimum values in MBE_Q of the j^{th} MBR of the v_i. This is illustrated in Figure 3.4 (e). Since MBE_Q covers the entire possible matching area, any point that lies outside the MBE_Q is not counted for the similarity. The number of intersecting points between B and MBE_Q overestimates $LCS_{\delta,\epsilon}(B, Q)$ [10]. The number of intersections is counted through an R-tree operation as in Figure 3.4 (b), which is a intersection of Figure

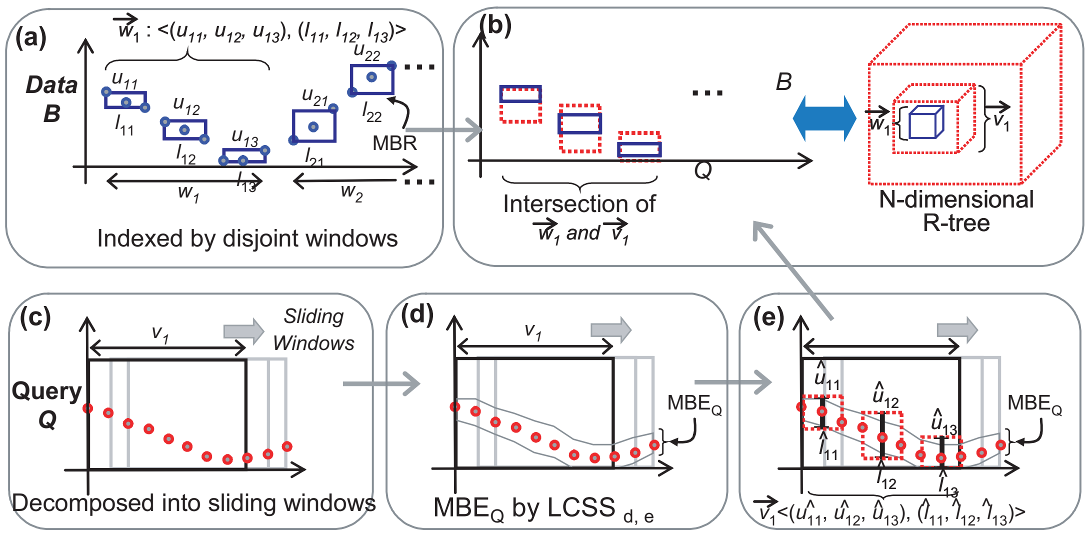

(a)
$\vec{w}_1$: <(u_{11}, u_{12}, u_{13}), (l_{11}, l_{12}, l_{13})>
Data B
MBR
w_1
w_2
Indexed by disjoint windows
(b)
B
Q
Intersection of $\vec{w}_1$ and $\vec{v}_1$
N-dimensional R-tree
(c)
Sliding Windows
Query Q
v_1
Decomposed into sliding windows
(d)
MBE_Q
MBE_Q by LCSS $_{d,e}$
(e)
$\vec{v}_1$<($\hat{u}_{11}$, $\hat{u}_{12}$, $\hat{u}_{13}$), ($\hat{l}_{11}$, $\hat{l}_{12}$, $\hat{l}_{13}$)>

Figure 3.4: Indexing and Index Matching where w=9 and N=3

3.4 (a) and Figure 3.4 (e).

3.5 Window Sliding Schemes in Index Matching

There are three ways to slide query windows and choose the candidate matching subsequences: Simple Single Window Sliding, Single Window Sliding, and Multiple Window Sliding. We explain each window sliding scheme and show how the the bounding similarity is computed.

3.5.1 Simple Single Window Sliding

In this scheme, as illustrated in Figure 3.5(a), we compare a sliding window of a query to the index, which is first introduced in the Dual Match [45]. This overestimation method cannot be applied to the *LCS*-based subsequence matching since it is based on the Euclidean distance.

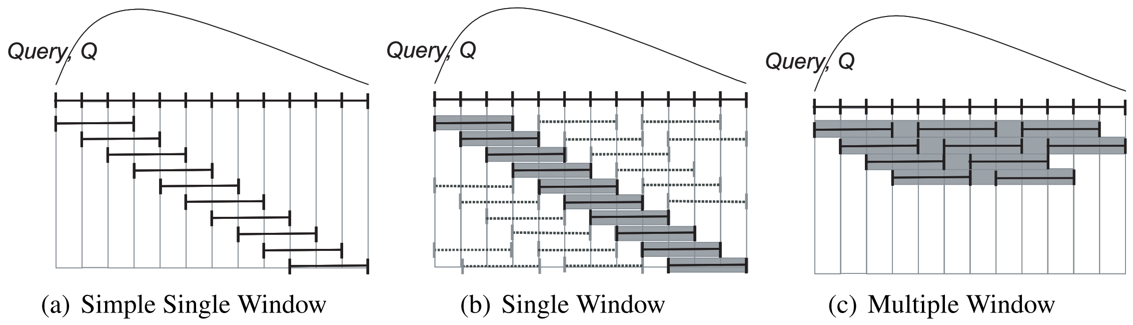


Figure 3.5: Window Sliding Schemes when $|v|$=4.

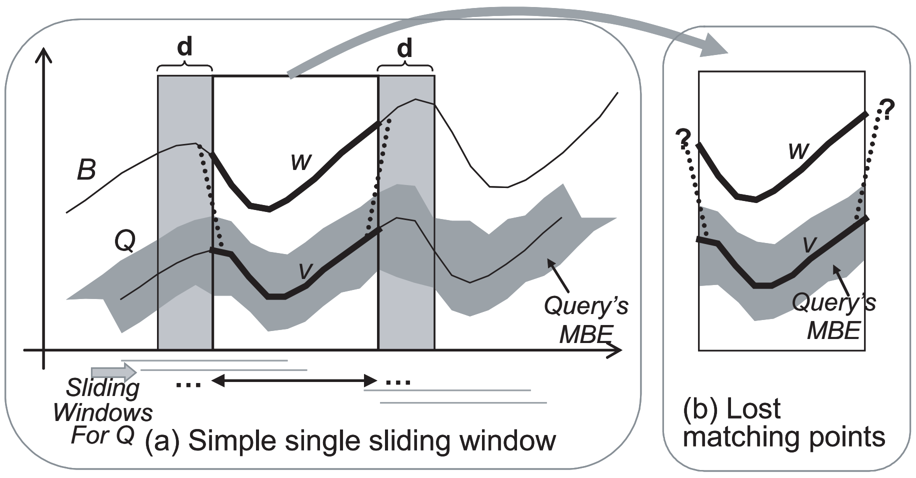


Figure 3.6: Matching points (connected by dotted lines) are not captured in the index matching using LCS

We should consider δ on both ends of the query sliding window. In Figure 3.6 (a), a sliding window v of a query Q is matched to a window w of the data sequence B. In actual index matching, some points near the start and end of the query Q cannot be matched to those of w as in Figure 3.6 (b). The data is just indexed by *MBR* that does not consider δ time shift.

We newly compute the similarity threshold for the simple single window sliding method. Let us be reminded of our subsequence matching problem: find all the matching subsequence $B[i:j]$ of the length $|Q|+2\delta$ in data sequence B such that the similarity $LCS_{\delta,\epsilon}(Q, B[i:j])$

is no less than s% of the $|Q|$, $\frac{s}{100}|Q|$. Let v be a sliding window of Q. The minimum similarity, θ is

$$\theta = |v| - (|Q| - \frac{s}{100}|Q|) - 2\delta \tag{3.1}$$

The term, $(|Q| - \frac{s}{100}|Q|)$, for the Equation (3.1) is subtracted from $|v|$ when all the mismatches can be found in the current window v. The last term 2δ is the maximum possible number of the lost matching points.

3.5.2 Single Window Sliding

When the query length is long enough to contain more than one sliding window, we can use the consecutive matching information as in Figure 3.5(b). Let us assume query Q and matching subsequence B have M consecutive disjoint windows, B_i's and Q_i's. If some Q_i and B_i pairs are not similar, then the other Q_j and B_j pairs should be similar, and we can recognize the B and Q pair as a candidate because of B_j and Q_j. When all B_i and Q_i pairs have the same similarities, we should have the minimum value to establish the candidate for comparison. The *multiPiece* search [5] is proposed to choose candidates through this process. It is the same for the Euclidean distance measure. In the *multiPiece*, the two subsequences, B and Q, of the same length are given, and each can be divided into p subsequences, each of which is of the length l. $d(B,Q) < \epsilon \Rightarrow d(B_i,Q_i) < \frac{\epsilon}{\sqrt{p}}$ for some $1 \leq i \leq p$ where B_i and Q_i are i^{th} subsequences of length l and $\epsilon > 0$. In the case of the Dual Match using Euclidean distance, we can count a candidate if the distance is less than or equal to $\frac{\epsilon}{\sqrt{p}}$.

Similarly, in the case of *LCS*, $LCS_{\delta,\epsilon}(B,Q) > \frac{s}{100}|Q| \Rightarrow LCS_{\delta,\epsilon}(v, Q[i:j]) > \frac{M|v|-(|Q|-\frac{s}{100}|Q|)-2\delta}{M}$

for some $j - i + 1 = |v|$. So the similarity threshold for single window sliding, θ_s, is

$$\theta_s = |v| - \frac{(|Q| - \frac{s}{100}|Q|) + 2\delta}{M} \tag{3.2}$$

As illustrated in Figure 3.5(b), M consecutive sliding windows are thought to be one large sliding window that might lose the warping path at both ends. The threshold for the M sliding windows is $M|v| - (|Q| - \frac{s}{100}|Q|) - 2\delta$, and it is divided by M for one sliding window. If one of the sliding windows among consecutive M sliding windows in Q is larger than or equal to θ_s, we can obtain a candidate, and we do not have to do index matching for the remaining consecutive sliding windows at the same candidate location.

3.5.3 Multiple Window Sliding

In this new window sliding scheme, as illustrated in Figure 3.5(c), the matching results of consecutive sliding windows in a query are aggregated. If we sum up the index matching result from M consecutive sliding windows, we can obtain fewer false candidates than when we use only one window. Let M be the number of consecutive windows fitted in a query Q. We vary M to contain the maximum number of sliding windows depending on the left-most window.

The index matching results of each sliding window for all disjoint data windows are added up to get M consecutive sliding windows. In Figure 3.7, the aggregation is done by accumulating the results in a vector A of the size $\frac{|B|}{w}$. B is the data sequence and w is the length of an index window. Let's assume that $< v_1 \ldots v_M >$ is a series of consecutive windows in the query

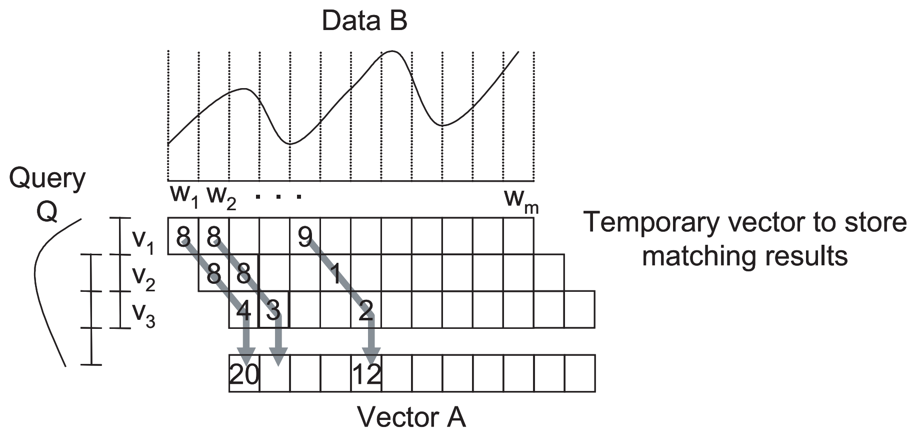


Figure 3.7: Index Matching Result

Q. The index matching results of a query window v_j are placed in a temporary row vector in Figure 3.7. It is added to A, and A is shifted right. The next matching result for v_{j+1} is placed in the temporary row vector. It is added to A, and A is shifted right. In Figure 3.7, we get A such that

$$A[1] = LCS_{\delta,\epsilon}(\overrightarrow{v_1}, \overrightarrow{w_1}) + LCS_{\delta,\epsilon}(\overrightarrow{v}_2, \overrightarrow{w}_2) + LCS_{\delta,\epsilon}(\overrightarrow{v}_3, \overrightarrow{w}_3),$$

$$A[2] = LCS_{\delta,\epsilon}(\overrightarrow{v_1}, \overrightarrow{w_2}) + LCS_{\delta,\epsilon}(\overrightarrow{v}_2, \overrightarrow{w}_3) + LCS_{\delta,\epsilon}(\overrightarrow{v}_3, \overrightarrow{w}_4),$$

$$\vdots$$

$$A[m] = LCS_{\delta,\epsilon}(\overrightarrow{v_1}, \overrightarrow{w_{m-2}}) + LCS_{\delta,\epsilon}(\overrightarrow{v}_2, \overrightarrow{w}_{m-1}) + LCS_{\delta,\epsilon}(\overrightarrow{v}_3, \overrightarrow{w}_m).$$

The shift operations aggregate the consecutive index matching results.

The similarity threshold for multiple sliding windows, θ_m, is computed as if the consecutive M windows moved as one.

$$\theta_m = M|v| - (|Q| - \frac{s}{100}|Q|) - 2\delta \tag{3.3}$$

θ_m is for an aggregate comparison of M consecutive sliding windows, while θ_s is for one sliding window.

Using the aggregation of the consecutive index matching information, we can enhance the pruning power of the index. That is, we have fewer false alarms than the single window sliding scheme does. In Figure 3.7, the diagonal sum illustrates the aggregatation of the consecutive index matching results. If $\theta_s = 8$, the first, second, and the fifth diagonals are selected as the candidates since one of the matches is greater than or equal to 8. However, in the case of the multiple window sliding, if $\theta_m = 20$, the fifth diagonal is not a candidate since the sum 12 is less than 20, so it has fewer false alarms than the single window sliding scheme does.

3.6 Post-Processing and Skipping

3.6.1 Post-Processing

Post-processing is the final procedure to determine the whole length of the matching subsequence depending on the position of the matching sliding window in a query. The actual similarity computation should be performed for the whole interval of the subsequence against the query. Figure 3.8 demonstrates the post-processing. We intentionally omit the adjacent matching subsequences and show only one that matches. Through the index matching process, matching indexes for each sliding window ①, ②, ③ are to be found and then the whole length of the candidate subsequence is computed including 2δ area. In Figure 3.8, one candidate subsequence has an index matching area and the entire length of the match is determined

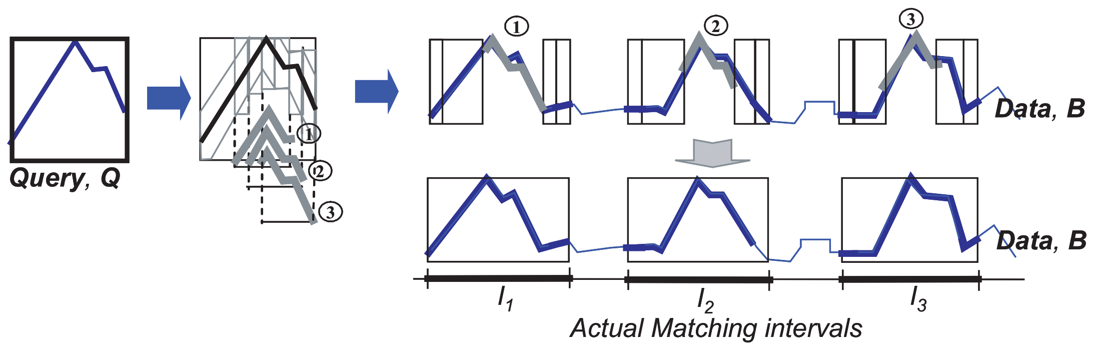


Figure 3.8: Postprocessing determines the entire lengths of the candidate subsequences depending on the location of the sliding window in the query.

3.6.2 Skipping the *LCS* Computation

After determining the whole length of the candidate subsequences, skipping the *LCS* computation is applied to reduce the computational load. Subsequence matching cannot avoid many adjacent matching subsequences where one subsequence is found. By tracing the warping path of the matching subsequences in its *LCS* warping path matrix, we can reduce the *LCS* computation.

3.7 Experiment of Single Channel Time Series Dataset

Experiments were conducted on a machine with a 2.8 GHz Pentium 4 processor and 2GB memory using Matlab 2006a and Java. Here are the parameters to run the tests:

- **Dataset**: We have used 48 different time series datasets[1] for evaluation. Each dataset

[1]http://www.cs.ucr.edu/ eamonn/TSDMA/UCR, The UCR Time Series Data Mining Archive

has a different data length and a different number of channels. We set the length of each to 100,000 by attaching the beginning to the end so that all the datasets have the same length.

- **Index**: We set the dimension to 8 and *MBR* size to 4. Determining the sizes of the dimension, *MBR* and R-tree requires domain knowledge.
- **Query**: We choose 4 fixed lengths of queries, 100, 150, 180, and 200, so that each length includes 3, 4, 5 and 6 windows. Ten queries for each length are randomly selected from the data sequence.
- **Similarity**: ϵ is set to 1 $\%$ of the data range, and δ is set to 2.5 $\%$ of the $|Q|$. Similarity threshold S is set to 99% of the $|Q|$.

3.7.1 Different Sliding Schemes and Candidates

We compare the performance of the two different index sliding schemes, namely, the single window sliding and the multiple window sliding scheme. Figure 3.9 shows the ratios, $\frac{\text{\# of candidates by single windows sliding}}{\text{\# of candidates by multiple windows sliding}}$ for different lengths of queries of each dataset. Ratios greater than one means the multiple window sliding scheme generates fewer candidates than those of the single window sliding scheme. The multiple window sliding scheme has fewer false alarms than the single window sliding scheme in the tests. The ratio varies from 1 to 140. The multiple sliding window scheme generates candidates only $\frac{1}{140}$ of the single window sliding scheme in the Fluid dynamics dataset.

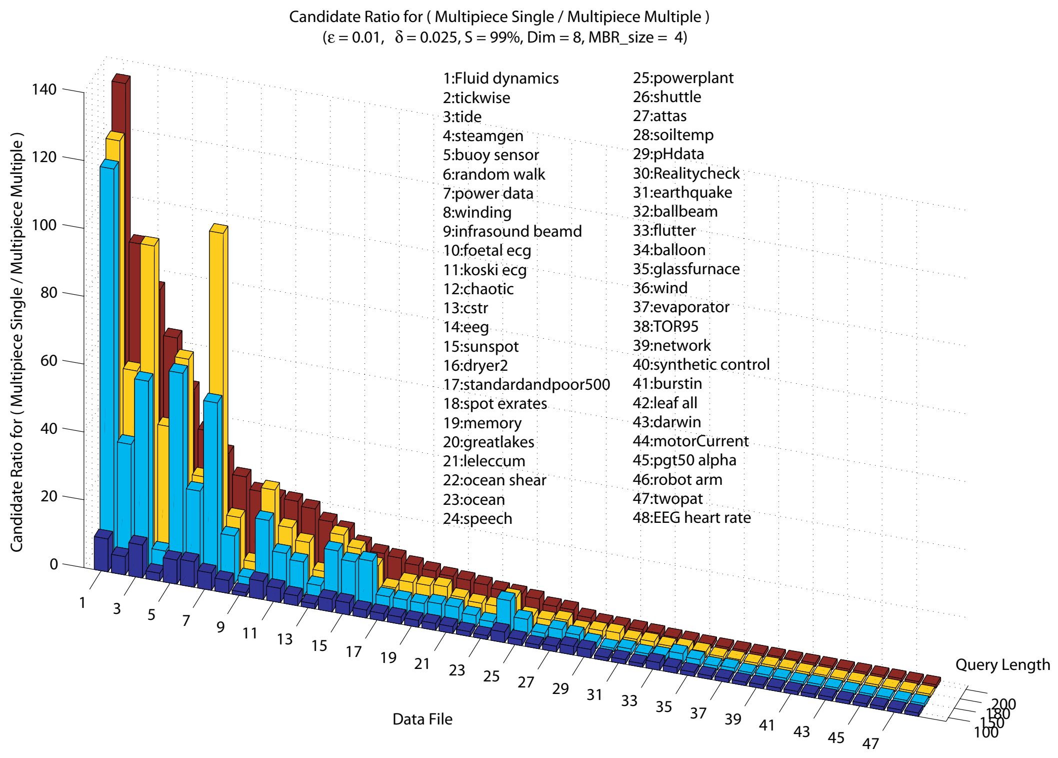


Figure 3.9: Candidate Ratio $= \frac{\text{\# of candidates by single windows sliding}}{\text{\# of candidates by multiple windows sliding}}$. The same color indicates queries of the same length.

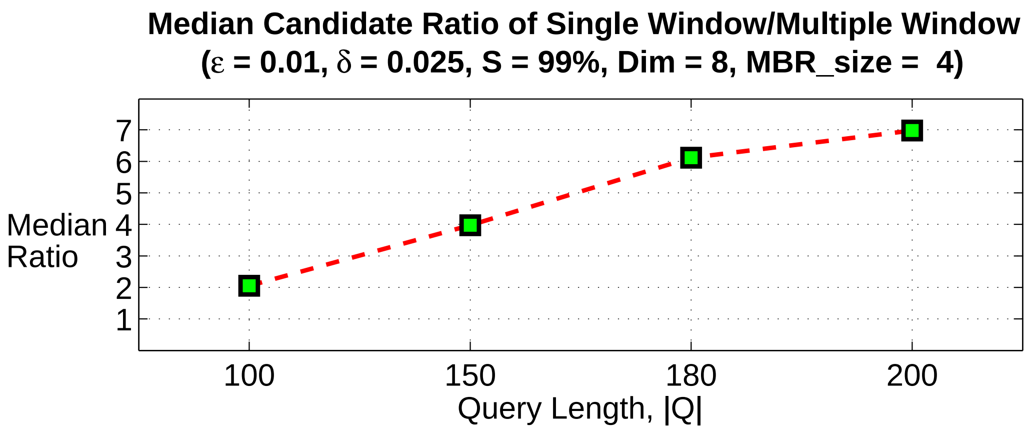


Figure 3.10: Summary of Candidate Ratio in Figure 3.9

Figure 3.10 shows the median values from the Figure 3.9 for each length of the queries. Figure 3.10 summarizes how much the performance is improved as the length of the query gets longer in all of the datasets. It demonstrates that as the length of a query gets longer to include more index windows, fewer false alarms occur in the multiple window sliding than in the single window sliding.

However, in the datasets, such as an EEG heart rate, two pat, or robot arm, there is not much difference between the two methods. We can explain it in terms of the index. For these datasets, all of the disjoint data windows are very similar to each other. Figure 3.11 shows the first 500 points index of the best and the worst three datasets regarding the candidate generation. Comparing the index of the best three datasets to the worst three, we cannot easily distinguish a set windows from any other set of windows. This is the reason why the index search is difficult even though the multiple index information is used.

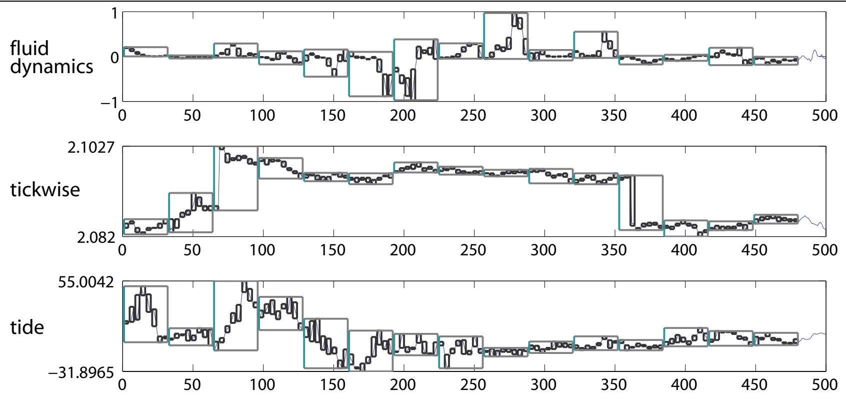
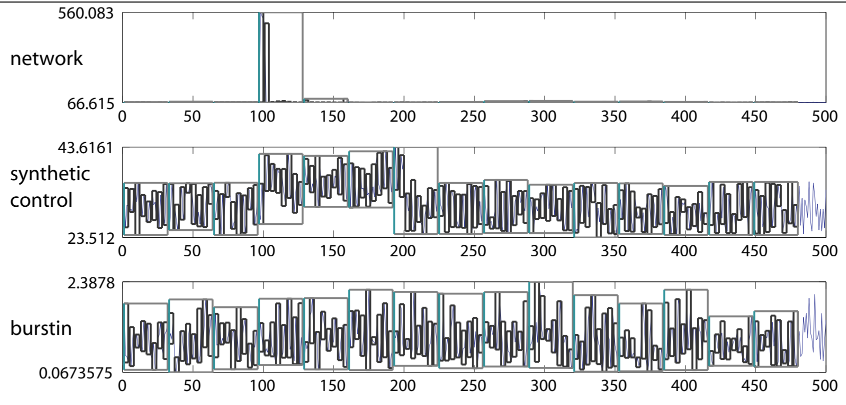
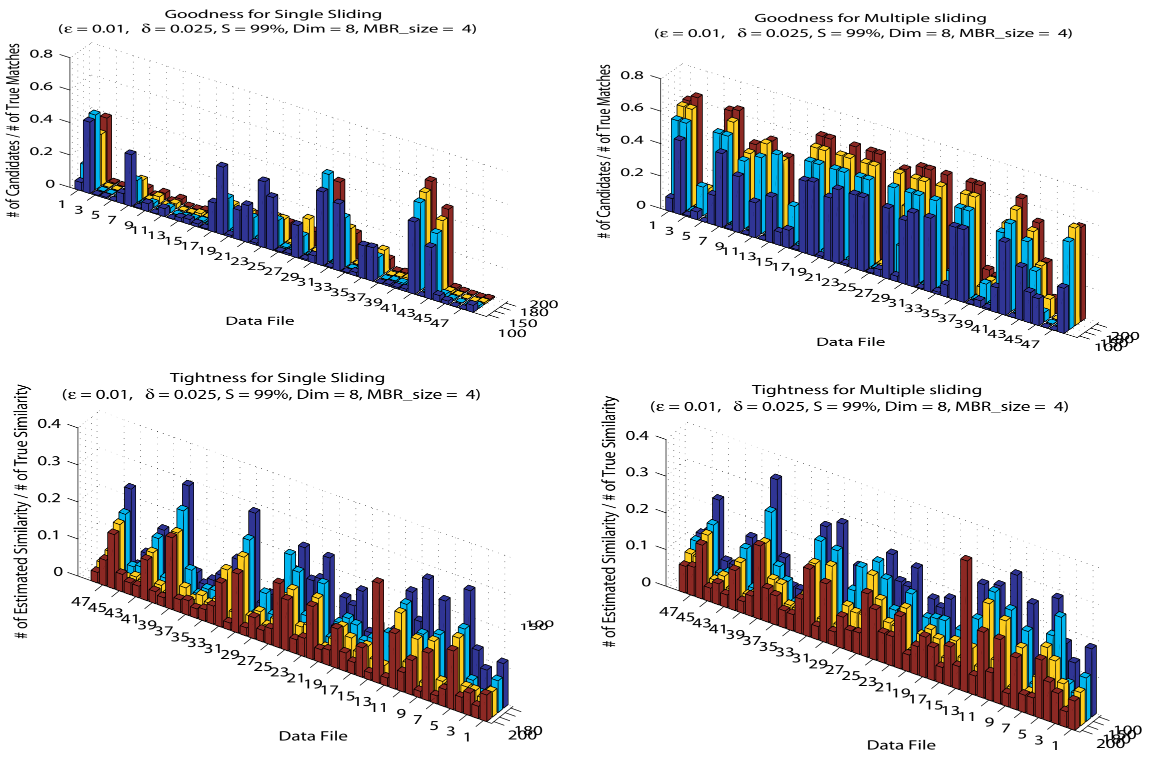
Best 3	
Worst 3	

Figure 3.11: Index



1:attas	13:Fluid dynamics	25:pgt50 alpha	37:steamgen
2:ballbeam	14:flutter	26:pHdata	38:sunspot
3:balloon	15:foetal ecg	27:powerplant	39:synthetic control
4:buoy sensor	16:glassfurnace	28:power data	40:tide
5:burstin	17:greatlakes	29:random walk	41:TOR95
6:chaotic	18:infrasound beamd	30:Realitycheck	42:twopat
7:cstr	19:leaf all	31:robot arm	43:wind
8:darwin	20:leleccum	32:shuttle	44:winding
9:dr er2	21:memor	33:soiltemp	45:koski ecg

Figure 3.12: Goodness and Tightness

3.7.2 Goodness and Tightness

Goodness and tightness are metrics that show how well the index works [8].

$$Goodness = \frac{\text{\# of all true matches}}{\text{\# of all candidates}} \tag{3.4}$$

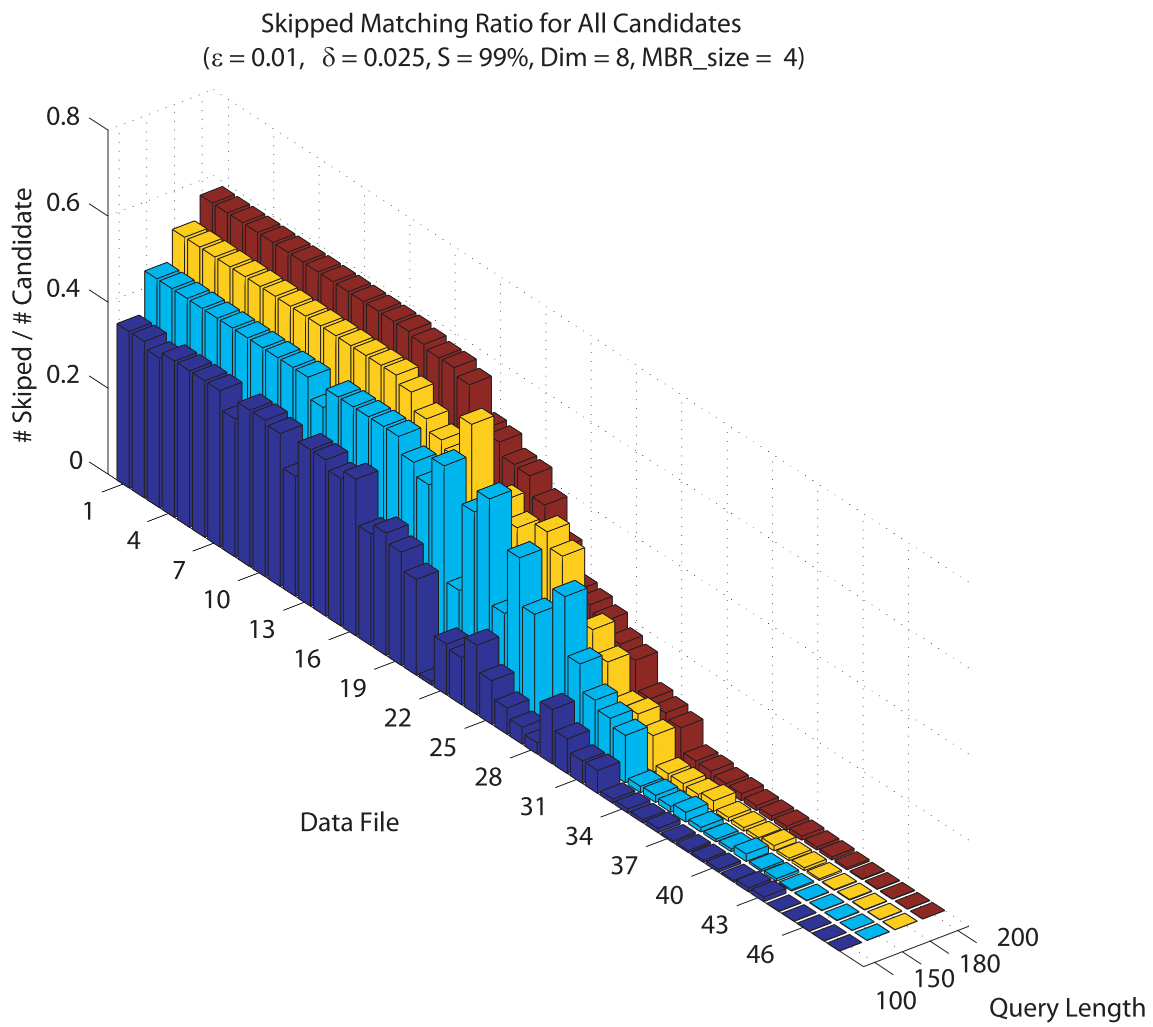
$$Tightness = \frac{\text{Sum of all true similarity}}{\text{Sum of all estimated similarity}} \tag{3.5}$$

Goodness shows how much the index reduces the expensive computations. Tightness shows how close the estimated values are to the actual values in indexing [8]. If the tightness is 1.0, then it means that the estimation is perfect. In Figure 3.12, the multiple sliding window scheme shows greater goodness and tightness than that of the single window sliding scheme.

3.7.3 Improving Performance by Skipping Similarity Computations

Figure 3.13 shows how effective the skipping of the similarity computation is. The chart demonstrates that we can avoid many similarity computations as the length of the query gets longer.

However, it also shows that the skipping mechanism does not work well for the datasets that cannot be properly indexed because the index parameter captures all of the windows in the data as well as the ones similar to the *LCS* matrix.



1:ballbeam
2:cstr
3:greatlakes
4:infrasound beamd
5:memory
6:ocean
7:ocean shear
8:random walk
9:Realitycheck
10:shuttle
11:standardandpoor500
12:tickwise
13:powerplant
14:TOR95
15:spot exrates
16:attas
17:leleccum
18:chaotic
19:wind
20:Fluid dynamics
21:pHdata
22:eeg
23:tide
24:winding
25:dryer2
26:power data
27:steamgen
28:buoy sensor
29:foetal ecg
30:koski ecg
31:sunspot
32:speech
33:motorCurrent
34:leaf all
35:synthetic control
36:flutter
37:glassfurnace
38:pgt50 alpha
39:robot arm
40:soiltemp
41:darwin
42:balloon
43:earthquake
44:twopat
45:evaporator
46:EEG heart rate
47:network
48:burstin

Figure 3.13: Skipping Similarity Computations

3.7.4 Runtime

We compare the performance of FRM, single window sliding dual match and the multiple sliding window Dual Match in CPU time. 10 randomly selected queries are searched against 48 datasets. Experiments are done for four different lengths: 100, 150, 180 and 200. Figure 3.14 shows median CPU times of 10 runs of subsequence matching using three different methods for each query length. It shows that in most cases, multiple sliding window scheme performs better than FRM and single sliding windows. FRM spent much more time on searching an R-tree than Dual Match method did. In the case the data size is n and index window size is w, FRM has $n\text{-}w\text{+}1 \approx n$ index elements in the R-tree while Dual Match has only n/w. FRM has more candidates than the Dual Match and FRM needs more operations to merge index information as the query length gets longer [5]. Figure 3.15 summarizes the Figure 3.14 by choosing median CPU time of all data sets for each length. It clearly demonstrates that as the query length gets longer, the multiple sliding window scheme based on the Dual Match method shows better performance in CPU time than the other two methods.

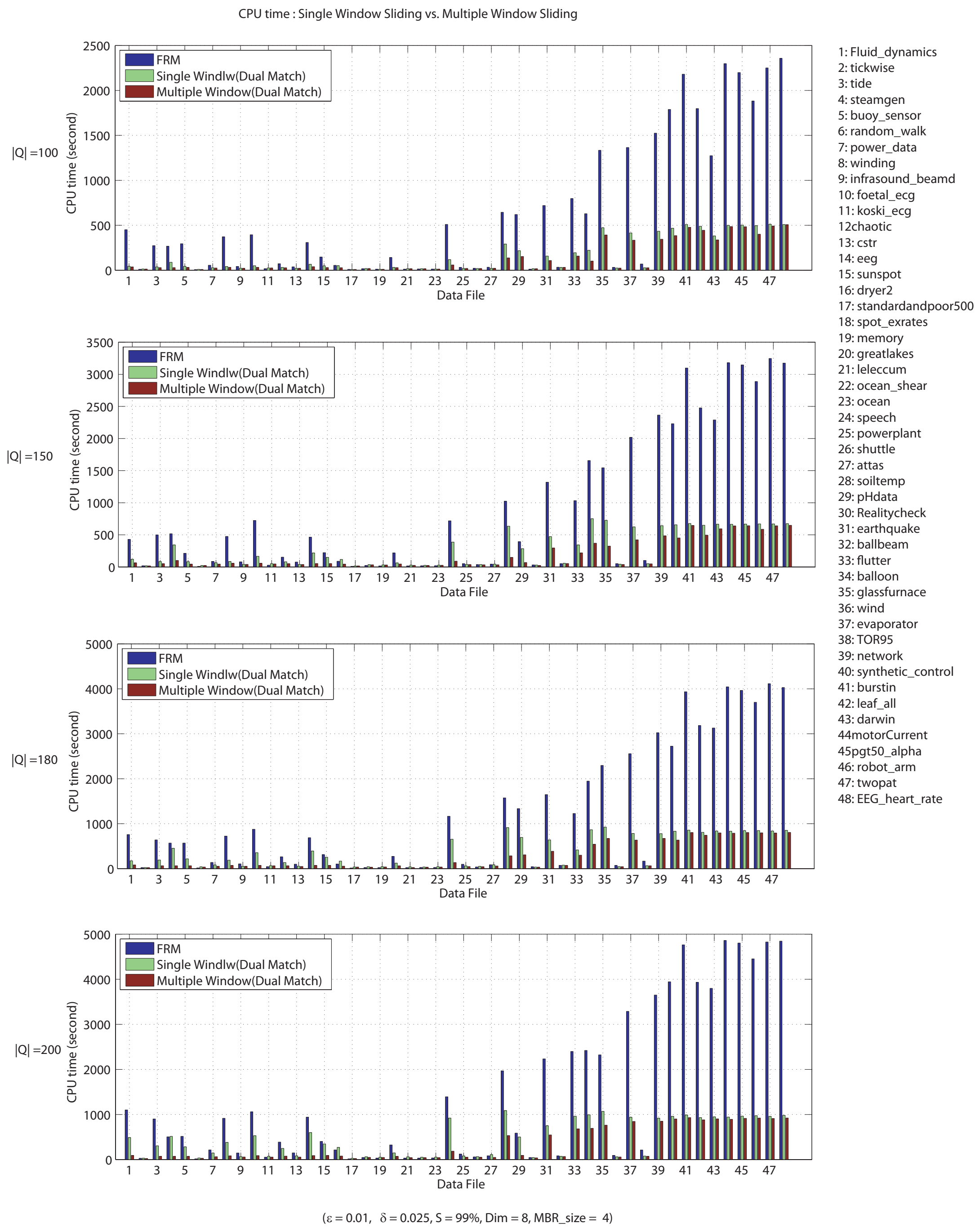


Figure 3.14: CPU Time for FRM, Single Window Sliding and Multiple Window Sliding Scheme

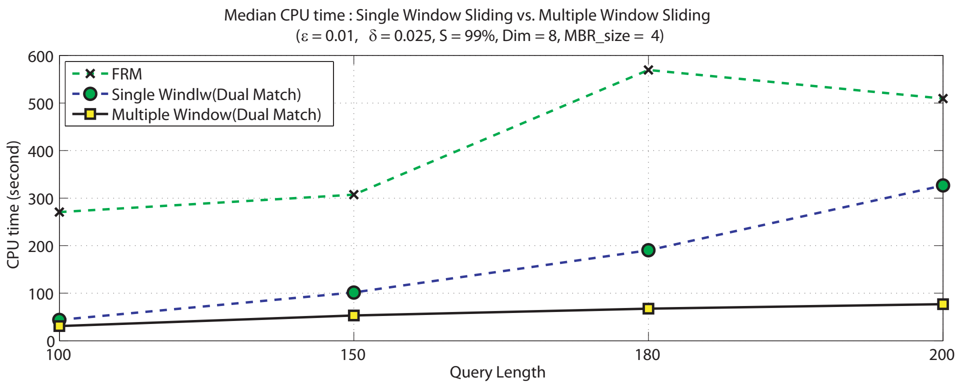


Figure 3.15: Median CPU Time for FRM, Single Window Sliding and Multiple Window Sliding Scheme

Chapter 4

Multivariable Subsequence Matching

4.1 Introduction

Multivariable stream data is ubiquitous today. Advances in sensors and wireless network technologies have produced many sensor network applications, such as object tracking, surveillance, object guarding, and structural integrity monitoring of large constructions , to name a few. In a typical sensor network application scenario, a large number of different types of sensors are deployed over a wide area. Each sensor generates a continuous sensor stream. Sensor streams are collected by a sink node and relayed to a server node for analysis. The combined data collected in the server node naturally forms a multivariable time series data. Multivariable time series data is a set of time series that shares the same time stamps. We call an individual time series a channel. A multivariable time series data consists of multiple numbers of channels. Multivariable time series are more popular and useful in the real world than the single

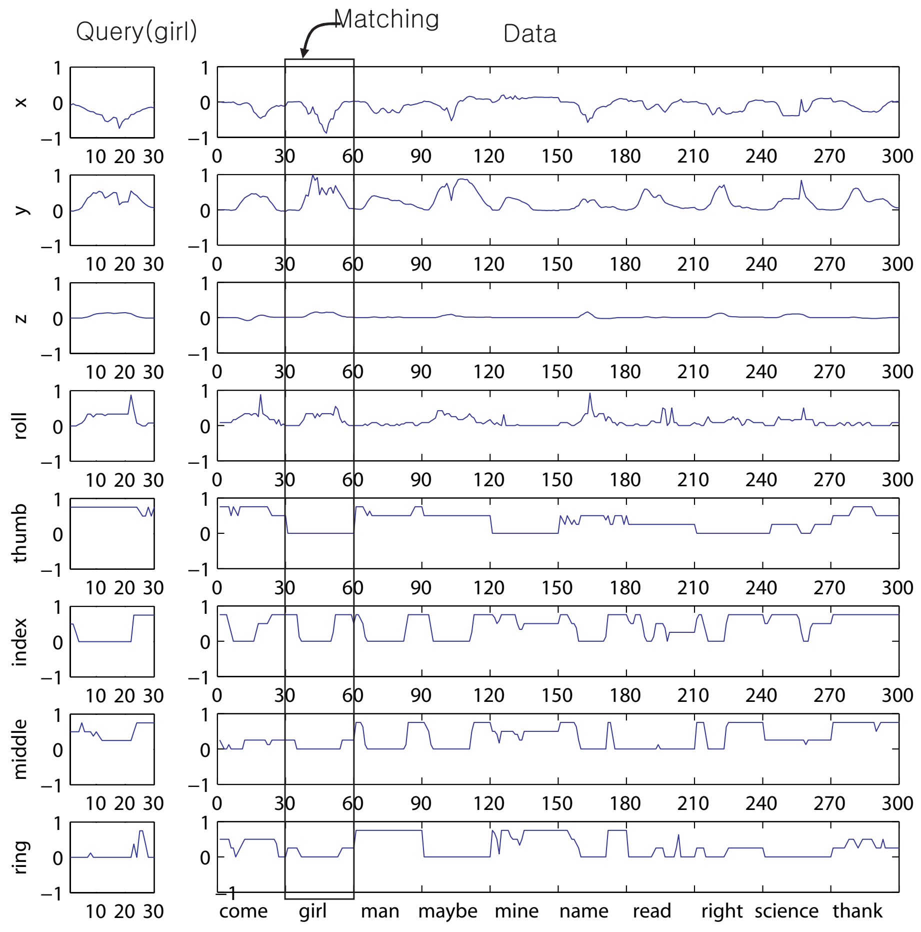


Figure 4.1: Multivariable Subsequence Matching

channel time series data. With the advancement of the sensor technology, many applications require tracing a large number of channels from various sources.

Time series data is usually stored in a back-end intelligence module to analyze the properties of the data. The multivariable time series data is also fed as input to some online applications monitoring real-time events based on the back-end analysis. Unfortunately, however, traditional data mining solutions are not directly applicable to the sensor network application framework. Traditional techniques focus on rather static collections of records containing mainly discrete values, while in sensor network applications, data is an unbounded stream of continuous values. Furthermore, characteristics of data instances are also different. A record in sensor stream data typically contains values representing particular sensor readings at a given point in time. Unlike the traditional databases, the records in this environment have temporal locality; for example, the current sensor readings are likely to be similar to the ones observed recently rather than to the ones monitored a long time ago. These differences pose substantial challenges to the traditional data management techniques, enough to force a paradigm shift in the long-established data management standards.

In this work, we will attempt to address some of the challenges, especially in subsequence matching in multivariable time series data. We extend the subsequence matching technique to multivariable time series with *LCS*. Most of the existing solutions focus on searching subsequence matching of a single channel data. [4, 5, 8, 9, 45]. Although these techniques are successful in many application domains, they fail to search efficiently interesting patterns that may span over multiple interdependent sensor readings in a multivariable time series data.

The use of multiple channels of signals would increase the accuracy of the security system even though multivariable time series requires complex analysis. For example, in sign language recognition, movements of the body parts are captured in a multivariable time series data. The orientation of a palm, its angles as well as the positions of fingers, wrists, and arms are represented in a time series for recognition. Figure 4.1 shows an example of a search of multivariable time series in Australian Sign Language [7]. The data has 8 channels (x, y, z position of a hand, orientation of a palm, and the folding degree of 4 fingers). In the figure, data (to be searched) is a time series of 10 different sign words. Each word comes from one of the 20 different signers. The query is the sign word, ”girl.” Subsequence matching in this multivariable time series is to locate the query sign word in the data. In this example, data is a kind of a sign language sentence, although the actual one is more complicated than this one, since there is a pause between two words and the length of each word is not as regular as the provided example.

We chose *LCS* in definition 2 as our non-Euclidean measure to run similarity matching to overcome the Euclidean measure. If we apply *LCS* to each channel of a multivariable time series data, we do not have to perform extra computations for normalization or weighting to avoid problems from different basis units of each channel.

We avoid many expensive *LCS* computations in subsequence matching by observing the computation matrix. *LCS* requires expensive computation; $O(|Q|^2)$, when query Q is compared to the data of the same length. [10,12,20,21,28,46]. To reduce the number of computations for every instance of the matching subsequence, data is indexed by spatial structure, and candidate

Table 4.1: The Notation for Multivariable Time Series

B	A multivariable time series data sequence, $< b_{1,1}, b_{1,2}, \ldots, b_{2,1}, b_{2,2}, \ldots >$, each $b_{i,j}$ is a real number at the i^{th} channel and j^{th} time stamp.
$B[i,:]$	i^{th} channel of data
$B[:, j : k]$	a window of data from j^{th} frame to k^{th}frame
Q	A query sequence, we assume that all channels in Q are of the same length.

subsequences are computed through an inexpensive operation. In the following sections, we will explain how to compute and reduce candidate matching subsequences by applying our proposed method to the multivariable time series data. We will also validate our proposed method through the experiments carried out on 14 multivariable datasets.

4.2 Notational Conventions

Multivariable time series data is expressed in two dimensions, value and time, in several streams. Each stream is called a *channel* and it represents a feature or attribute of a temporal event in real numbers. These values are recorded in regular time intervals. A *time stamp* is a time point when a set of values of all channels is recorded. The set of values for all channels at a time stamp is a *frame*. A *window* is a set of frames. The number of frames in a window is the size of the window. All terms are illustrated in Figure 4.2.

We define some notations and terminologies in Table 4.1 for multivariable time series.

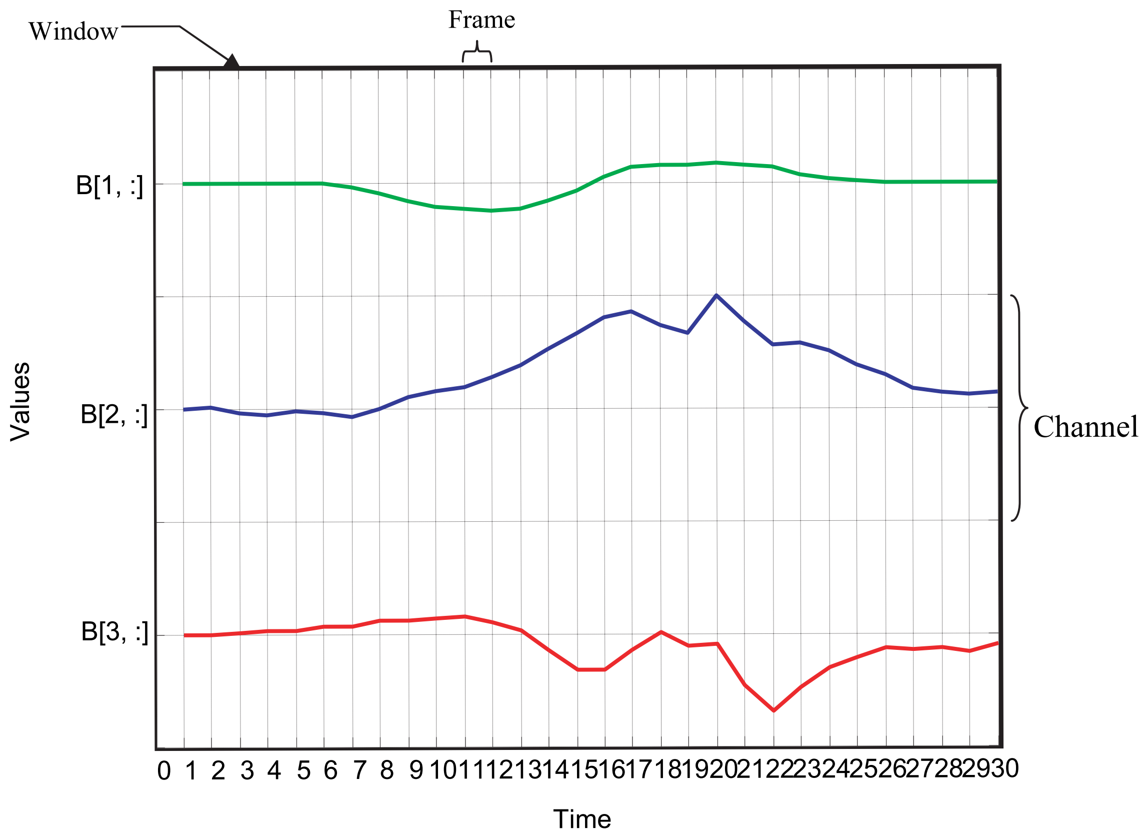


Figure 4.2: Windows, Frames and Channels in a Multivariable Time Series

4.3 Problem Statement

We apply the proposed subsequence matching method in a single channel dataset to the multivariable time series data. We generalize the definition of matching subsequences in Definition 3 for multivariable time series with $\overline{LCS_{\delta,\epsilon_i}}$.

Definition 4 *Let Q=* $< q_{1,1}, q_{1,2}, ...q_{m,n} >$ *be a query and B=* $< b_{1,1}, b_{1,2}, ... >$ *be a data subsequence of time series of* m *channels of finite length. Given an integer* δ*, a real number 0* $<$ ϵ_i *<1 for each* i_th *channel and user defined similarity threshold* θ*, we define the* ***matching subsequences****,* $M = \{B[:, j : k] \mid \overline{LCS_{\delta,\epsilon_i}(Q, B[:, j : k])} \geq \theta\}, where\ \overline{LCS_{\delta,\epsilon_i}(Q, B[:, j : k])} = \frac{\sum_{1<i<n} LCS_{\delta,\epsilon_i}(Q[i,:], B[i,j:k])}{m}\}$

Problem: Find all matching subsequences $B[:, j : k]$ of the length $|Q| + 2\delta$ for data sequence B and query Q such that the similarity $\overline{LCS_{\delta,\{\epsilon_i\}}(Q, B[:, j : k])}$ is no less than S% of the $|Q|$, $\frac{S}{100}|Q|$.

Solution Road Map Here is a road map of solutions to the problem:

- Index pruning criteria (bounding value) is computed and applied to each channel.
- Candidates from the index matching process are chosen by summing up the contiguous index search results.
- Adjacent matching intervals are efficiently skipped by observing the *LCS* matrix of all channels.

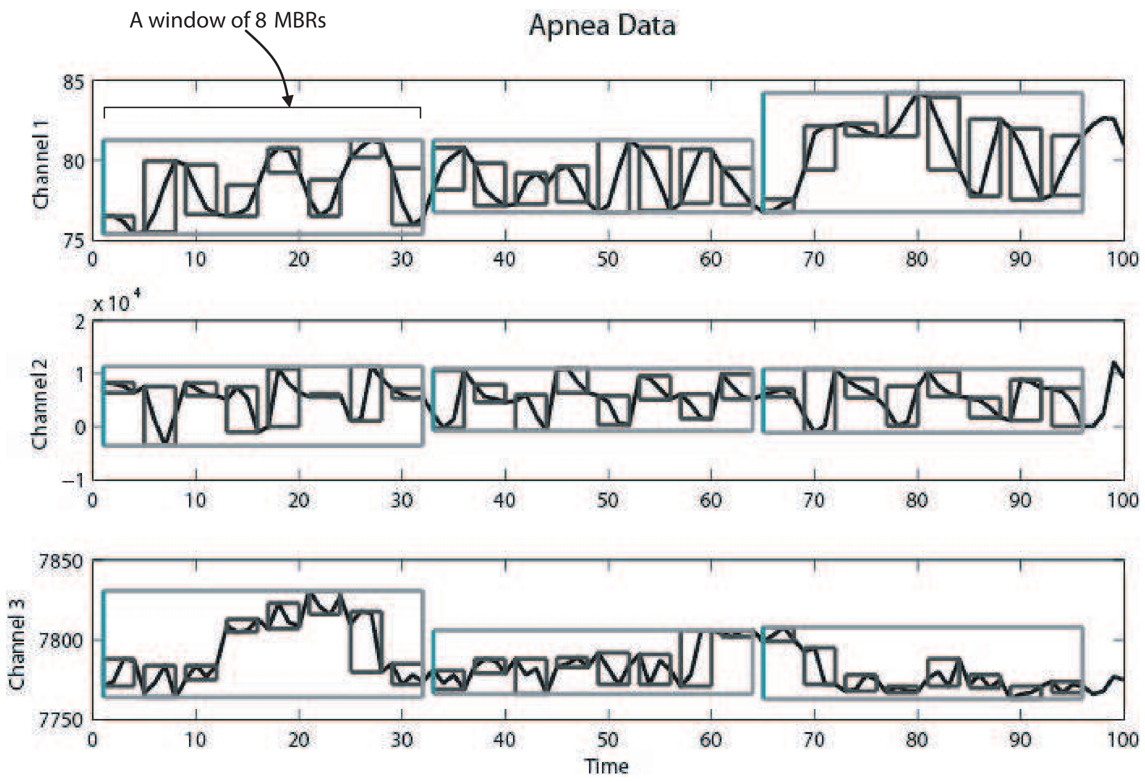


Figure 4.3: An Example of an Index that Shows *MBRs* and Windows

4.4 Indexing for Multivariable Time Series

Subsequence matching in multivariable time series data begins by indexing each channel of data into separate R-trees. If there are m channels, we need m separate R-trees. Each channel has its own error range to determine similarity. That is, i^{th} channel has its own ϵ_i for *LCS* similarity depending on the application context. All channels share the same δ.

Data is divided into equi-length disjoint windows for the index. Each window is represented by a multi-dimensional vector. That is, data sequence B is divided into equi-length disjoint windows $< w_i(k) >$, i^{th} window of the channel k. It consists of N *MBR*s. Let N be the dimensionality of the space we want to index. An *MBR* represents a dimension. N *MBR*s for a $w_i(k)$, is transformed into $\overrightarrow{w_i(k)} =< (u_{i1}, \ldots, u_{iN}), (l_{i1}, \ldots, l_{iN}) >_k$, where u_{ij} and l_{ij} represent the maximum and minimum values in the j^{th} interval of $w_i(k)$. $\overrightarrow{w_i}$ is stored in an N dimensional R-tree for channel k, R-tree(k).

4.5 Index Matching in Multivariable Time Series

A query is represented by *MBE-MBRs*. Each channel of the query Q is transformed into an *MBE*, and one or more sliding windows are chosen depending on window sliding schemes.

We obtain *m MBEs* from the *m* channels of the query Q, and each *MBE* is divided into separate *MBR*s of the chosen sliding window. We compare this transformed query against the data using the Dual Match index. Figure 4.4 illustrates index matching steps in multivariable time series.

Formally, let $MBE_Q(j)$ be an MBE for $Q[j,:]$, j^{th} channel of Q. Let the i^{th} sliding window of $Q[j,:]$ be $v_i(j)$. It is transformed into $\overrightarrow{v_i(j)} =< (\hat{u}_{i1}, \ldots, \hat{u}_{iN}), (\hat{l}_{i1}, \ldots, \hat{l}_{iN}) >_j$, where $\hat{u}_{ik}$ and $\hat{l}_{ik}$, respectively, are the maximum and minimum values in $MBE_Q(j)$ of the k^{th} MBR of the $\overrightarrow{v_i(j)}$.

$MBE_Q(j)$ covers whole possible matching areas using *LCS*. Any point that lies outside the $MBE_Q(j)$ is not counted for the similarity. The number of intersecting points between $B[j,:]$ and $MBE_Q(j)$ overestimates $LCS_{\delta,\epsilon_j}(B[j,:], Q[j,:])$ [10]. So, $\sum_j LCS_{\delta,\epsilon_j}(B[j,:], Q[j,:])$ overestimates actual similarity by counting all intersections of *MBE-MBRs* of Q and data index in all channels. The intersections are counted through the R-tree operation.

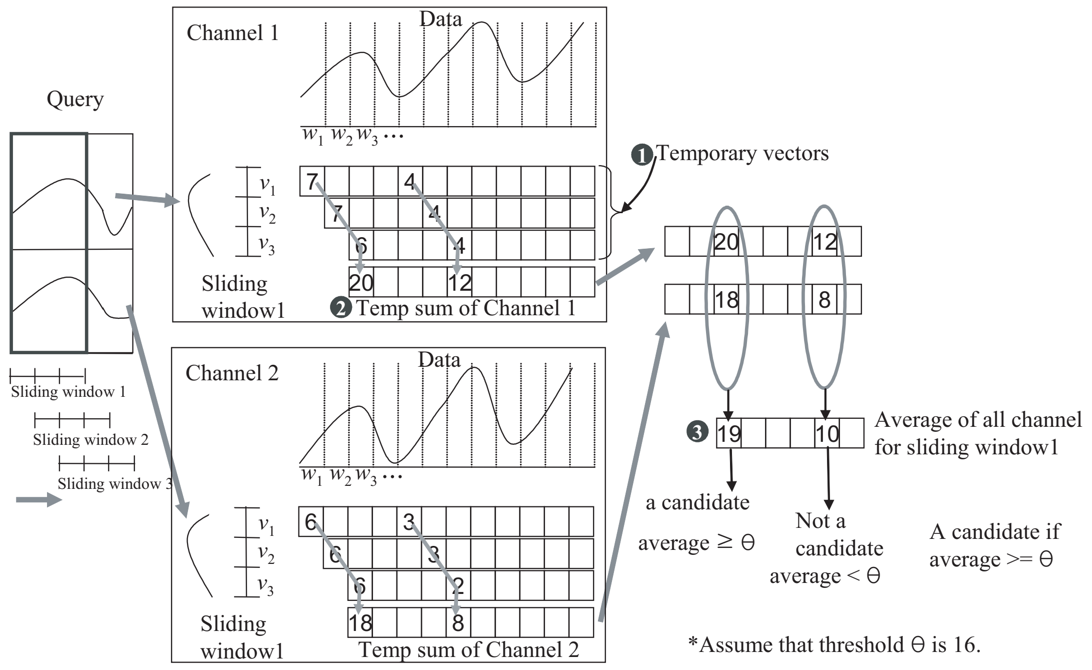


Figure 4.4: An Example of Multivariable Subsequence Matching

We apply one sliding window to all channels of query. In the figure, a window q_i of query Q is searched in R-tree indexes to find estimated matching values. These values are stored in a temporary vector like ① in figure 4.4. The individual matching process of each channel is the same as the one for the single channel data.

We use the same θ_s and θ_m depending on the window sliding schemes. These were computed in the previous chapter.

Temporary vectors of a channel are diagonally aggregated into a vector, temp sum of a channel, labeled ② in Figure 4.4. In the 2-channel example of Figure 4.4, the first element of the temporary vector in channel 1 is a sum of 7 + 7 + 6.

Index matching results of all channels in the temp sum vectors are aggregated and divided by the number of channels as ③ in Figure 4.4. This final vector contains index matching results. Only the vector elements greater than or equal to the matching threshold, θ_m or θ_s, are considered candidates.

4.6 Post-processing

Post-processing determines the whole length of the matching subsequence by actual *LCS* computation. Candidate subsequences are chosen based on this index matching. If the average value of *LCS* similarities of all channels are greater than or equal to the threshold , θ_s or θ_m, a candidate is chosen. If not, the window is not chosen for a candidate. In Figure 4.4, a window that has (12 + 8) / 2 in the result of index matching will be rejected since it is less than the computed threshold 16.

4.7 Skipping with *LCS* for Multivariable Time Series

We can skip one or more *LCS* computations in the neighboring candidates by observing the *LCS* warping path. In the case of the multivariable time series data, this observing process is done for each channel. We obtain all possible skippings for all channels. A minimum number among these possible skippings of all channels is chosen to skip $\overline{LCS}$ computations. This minimum number of skippings prevents us from computing the actual *LCS* for all channels.

4.8 Experiment of Multivariable Time Series Dataset

Here is a brief introduction to the multivariable datasets used in our experimental study and its parameters chosen to run our tests:

- **Dataset**: We have used 14 different time series datasets [1] for evaluation. Each dataset has a different length of data and a different number of channels from 2 to 15. We set the length of each to 10,000 by attaching the beginning to the end so that all the datasets are of the same length.
- **Index**: We set the dimension to 8 and *MBR* size to 4. Determining the sizes of the dimension, *MBR* and R-tree requires domain knowledge.
- **Query**: We chose 4 fixed-lengths of queries, 100, 150, 180, and 200, so that each length includes 3, 4, 5 and 6 windows. Ten queries for each length are randomly selected from the data sequence.
- **Similarity**: ϵ is set to 1 $\%$ of the data range, and δ is set to 2.5 $\%$ of the $|Q|$. Similarity threshold S is set to 99% of the $|Q|$.

4.8.1 Different Sliding Schemes and Candidates in Multivariable Time Series Data

In the first experiment with multivariable time series datasets, we compare the performance of two different index sliding schemes, single window sliding, and multple window sliding,

[1] http://www.cs.ucr.edu/ eamonn/TSDMA/UCR, The UCR Time Series Data Mining Archive

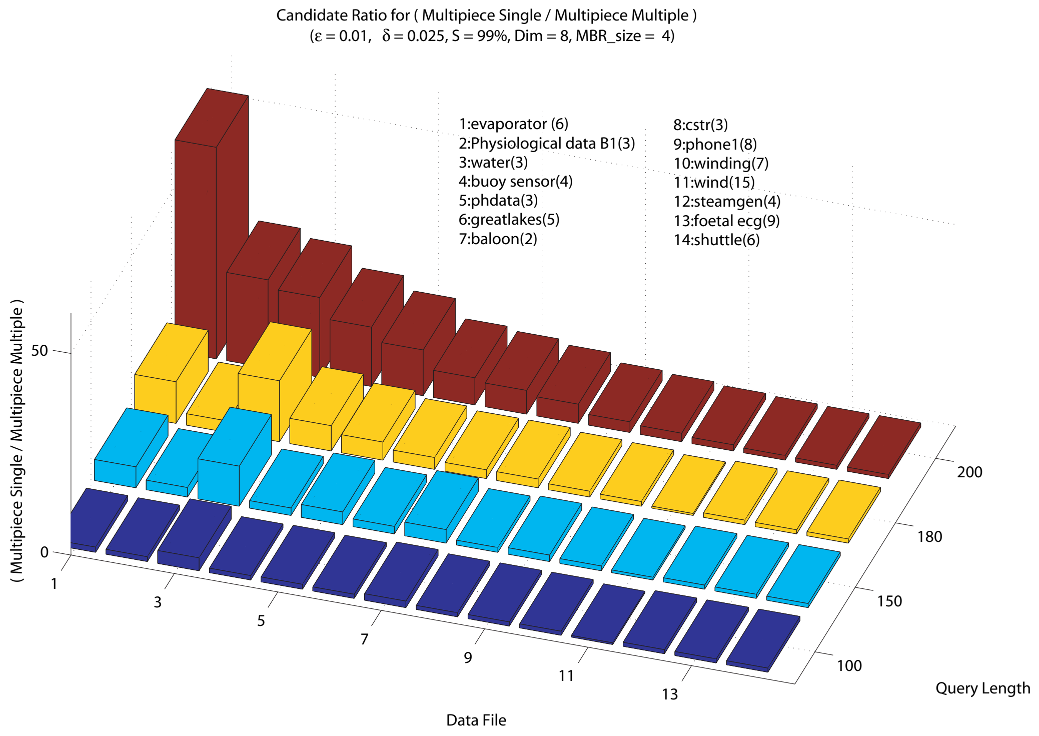


Figure 4.5: Candidates Ratio= $\frac{\text{\# of candidates by single windows sliding}}{\text{\# of candidates by multiple windows sliding}}$. The same color indicates queries of the same length.

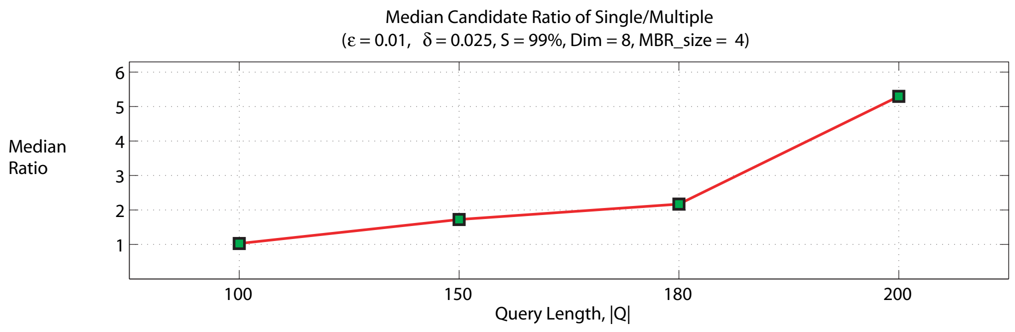


Figure 4.6: Summary of Candidate Ratio in Figure 4.5

in the multivariable time series data. By counting the number of candidates, we can correctly compare the performance of two methods independently. For ease of comparison, a ratio of the candidates is computed. Figure 4.5 shows that ratio, $\frac{\text{\# of candidates by single windows sliding}}{\text{\# of candidates by multiple windows sliding}}$ increases as the length of queries increases for most datasets. The number of channels of a dataset is specified in the legend of the figure. In this experiment, the multiple window sliding scheme has fewer false alarms than the single window sliding scheme. The ratio varies from 1 to 50. In the best case, the multiple sliding window generates candidates only $\frac{1}{50}$ of the single window sliding scheme.

Figure 4.6 shows the median values from Figure 4.5 for each length of the queries. It summarizes the improvement that occurs as the length of the query gets longer when we use the multiple window sliding method. We have fewer false alarms in the multiple window sliding than in the single window sliding as the length of a query gets longer. For the longer query, we have more index windows than for the shorter one, which reduces the chances of getting wrong candidates.

We can explain this difference in candidates in terms of the index. For these datasets, all

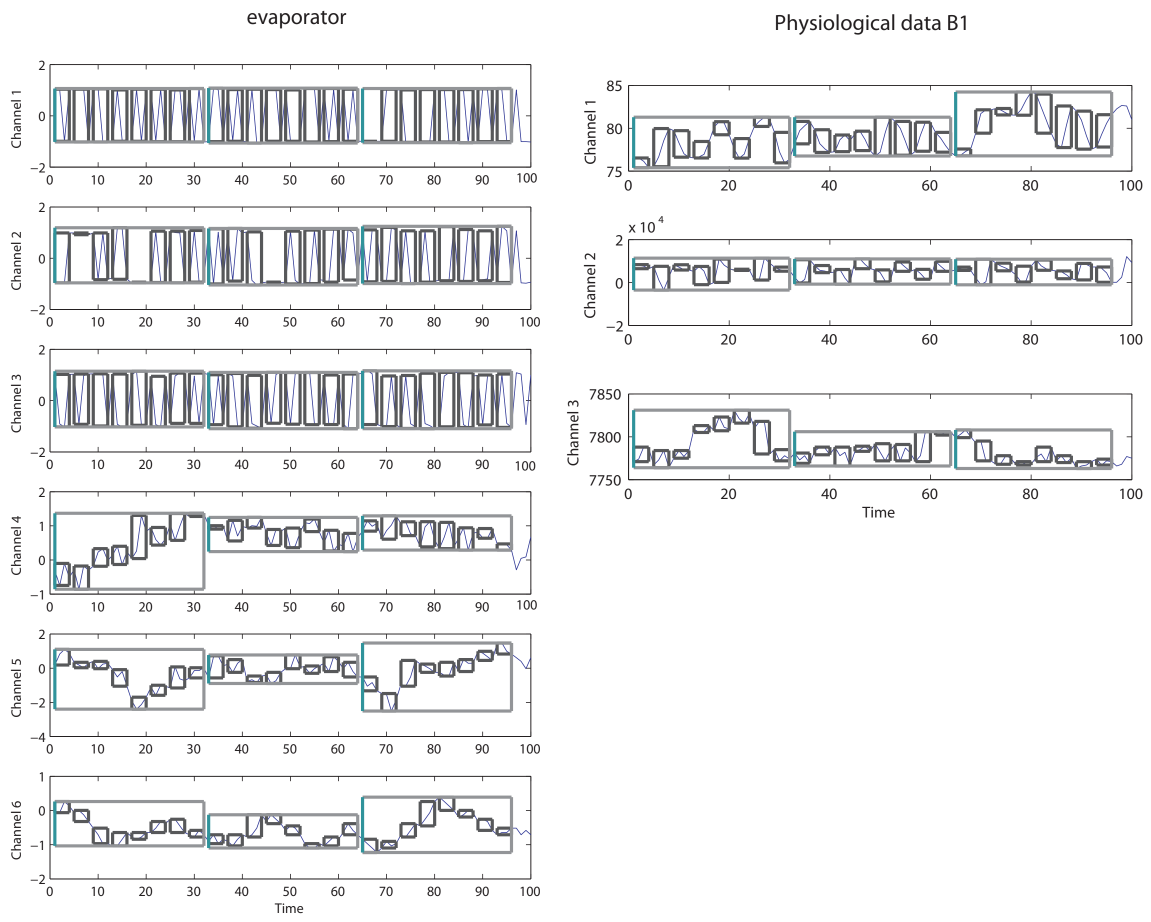


Figure 4.7: Index of Best 2 Multivariable Time Series

of the disjoint data windows are very similar to each other. Figure 4.7 and Figure 4.8 show the first 100 points index of the best and the worst two datasets in terms of the candidate generation. When we compare the index of the best two datasets to the worst two, we cannot easily distinguish a set of windows from another set of windows. This makes it hard to search the index quickly even though multiple index information is used.

We tested the effect of the number of channels in use. We chose the best data set, evaporator dataset, and applied the same experiment while increasing the number of channels from 1 to 6. Multiple sliding windows scheme greatly reduces the number of false candidates. Fig-

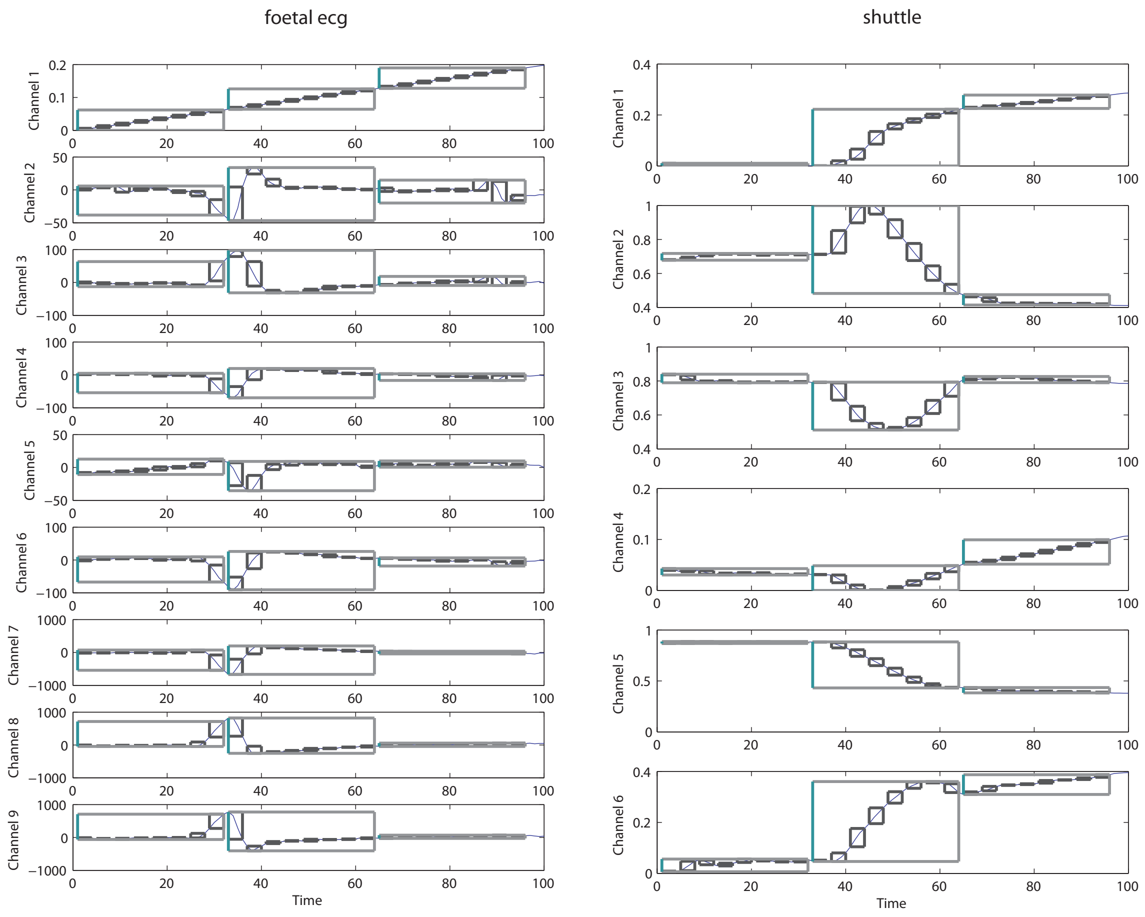


Figure 4.8: Index of Worst 2 Multivariable Time Series

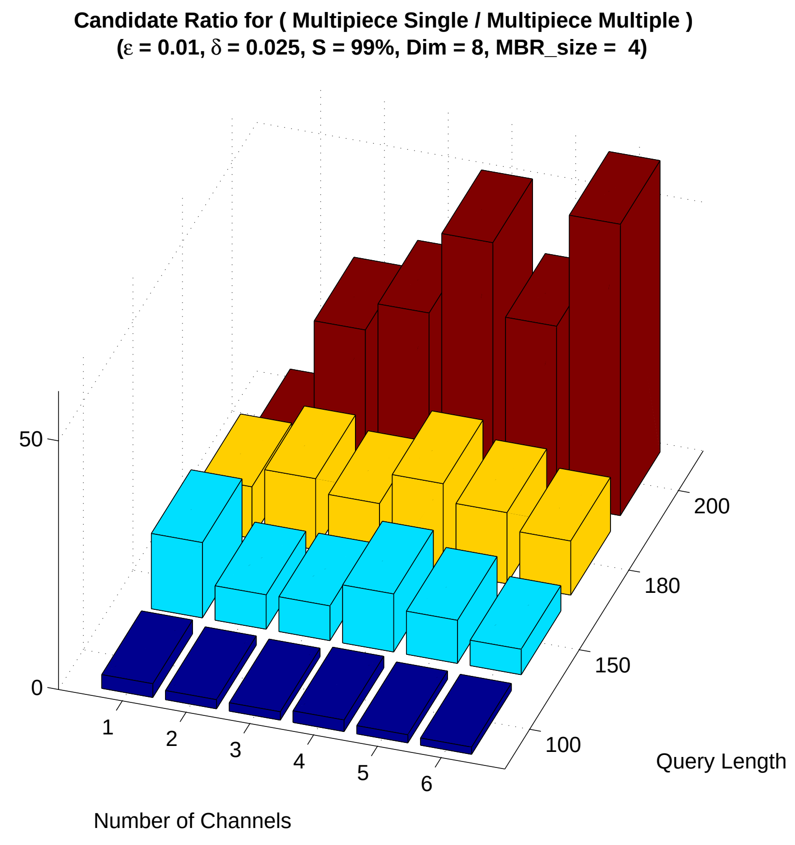


Figure 4.9: Candidate Ratio by Increasing the Number of Channels

ure 4.9 shows that, as the number of channels increases, the number of candidates decreases when multiple window sliding scheme is used. As the query length gets longer, the number of windows in the query increases. The increase of the number of windows in a query enables us to use more information in index matching process and as a result we have fewer false alarms than we have in a shorter query. From Figure 4.9, we can conclude that the number of channels affects candidate computation because it increases the amount of information for correct index matching.

4.8.2 Goodness and Tightness

We compute the goodness and the tightness as given in the equations 3.4 and 3.5. In Figure 4.10, the multiple sliding window scheme shows better goodness and tightness than the ones of the single window sliding scheme. We can see from the figure, as the length of query gets longer, goodness and tightness improve. A long query includes more windows than a short one. This increases the accuracy of the index matching process using multiple numbers of index windows.

4.9 Improving Performance by Skipping Similarity Computations

Figure 4.11 shows how the skipping strategy effectively reduces expensive $\overline{LCS}$ computations. As we found in the experiment with the single channel dataset in the previous chapter, this experimental result shows that we can avoid many similar $\overline{LCS}$ computations as the length of the query gets longer in the multivariable time series data.

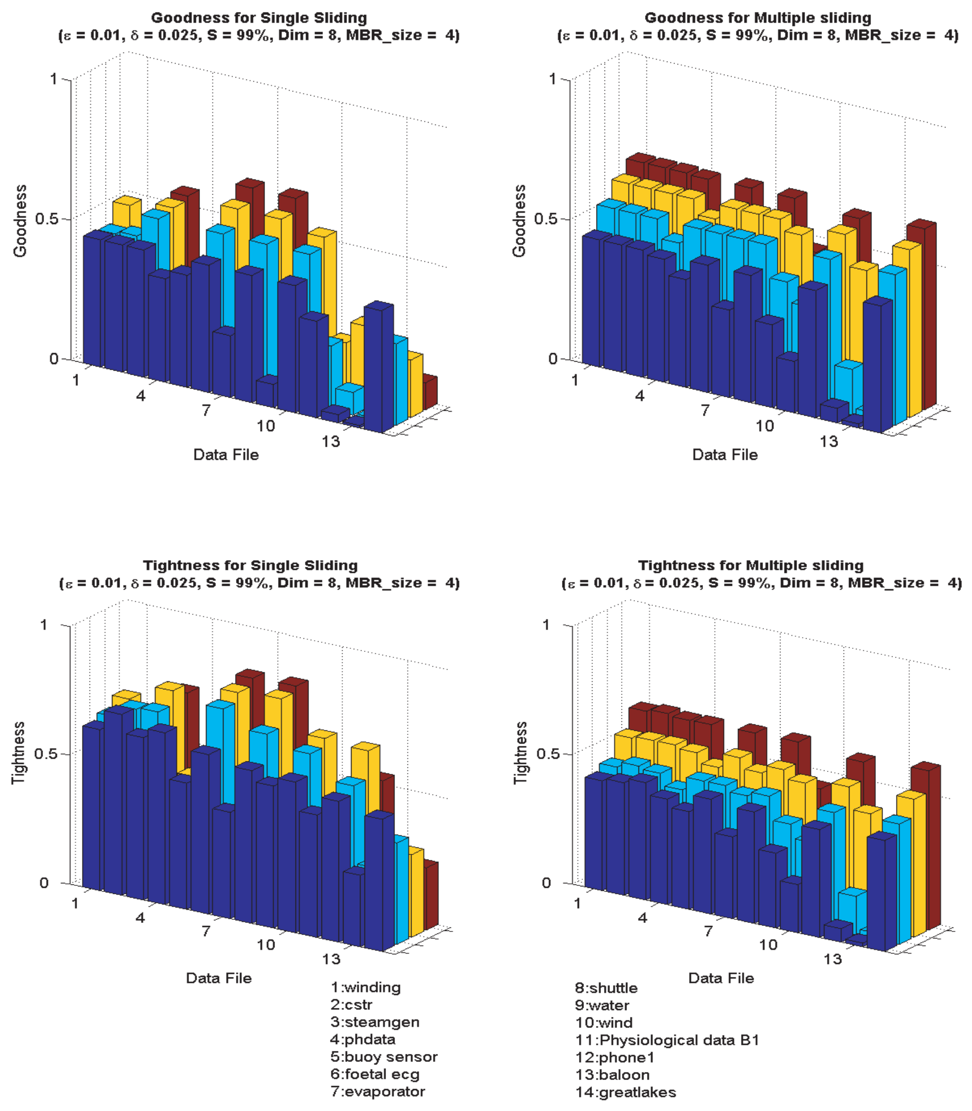


Figure 4.10: Goodness and Tightness from multivariable time series experiment

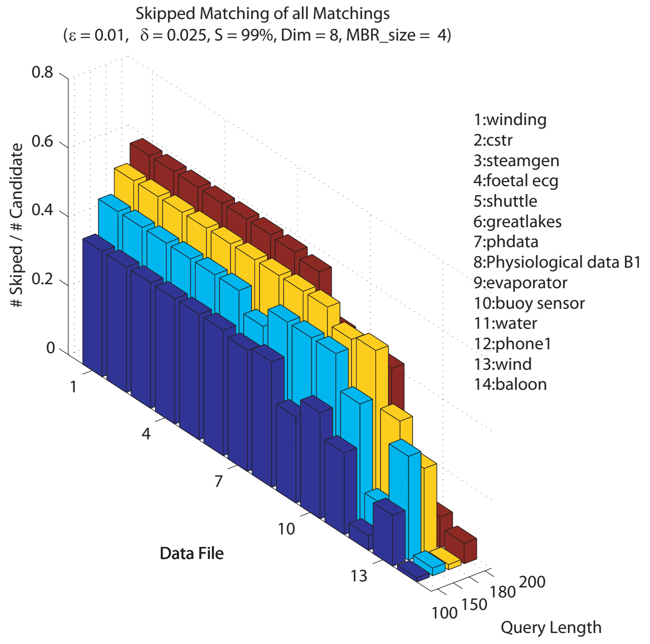


Figure 4.11: Skipping Similarity Computations

Chapter 5

Conclusion

We have proposed a novel subsequence matching framework that employs a non-Euclidean distance, in particular, *LCS*, a multiple window sliding scheme and a similarity skipping idea. Subsequence matching is used to find a query sequence out of a long range of data sequence. Our framework is based on the Dual Match subsequence matching where the data sequence is divided into a series of disjoint equi-length subsequences and then indexed in an R-tree while the query is searched in sliding windows. We newly computed the similarity bounds in the index matching framework with *LCS*. The proposed query matching scheme reduces many false alarms present in the previous approaches. We developed an algorithm to skip expensive *LCS* computations by observing the warping paths in a computation matrix. We extended our proposed subsequence matching scheme of a single channel data to multivariable time series data. An index is constructed in individual channels. A query is matched using a multiple sliding window scheme for each channel. The index matching results from all channels are aggregated and divided by the number of channels to determine candidates.

We have validated our new framework through experiments with 48 time series datasets for single channels and 15 for multivariable time series datasets. The proposed methods enabled us to have more intuitive and efficient subsequence matching algorithms. The multiple window sliding scheme was more efficient than the single window sliding scheme for the longer query in candidate generation, along with goodness and tightness. Skipping of the *LCS* computation greatly reduced expensive similarity computations in a single channel and multivariable time series data.

Regarding future work, the next research direction of the proposed work is to apply the subsequence matching method to other time series mining problems. Motif finding typically needs efficient subsequence matching algorithms. It requires a great amount of searching in order to guess basic patterns to explain events because we do not know ahead of time the lengths of frequent patterns. This implies that all patterns of all lengths of subsequences should be counted. Cluster analysis also includes several comparisons of neighboring subsequences and group similar patterns. All time series analysis work requiring heavy searching is a candidate for the next research topic.

Chapter 6

Appendix: First 500 Points of Datasets

Table 6.1: Time Series Data Used in the Experiments

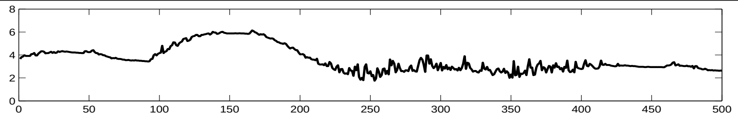
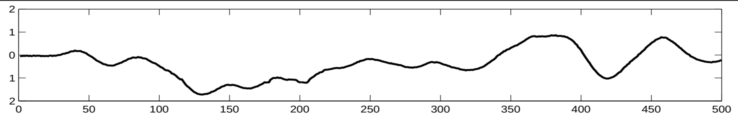
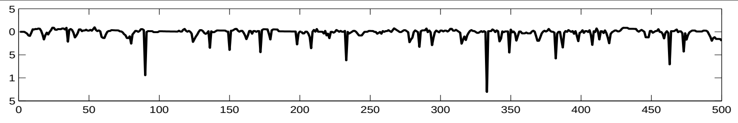
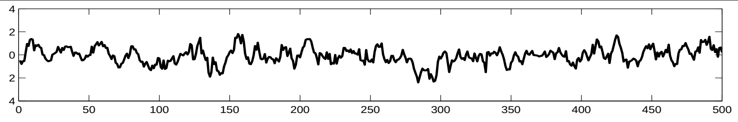
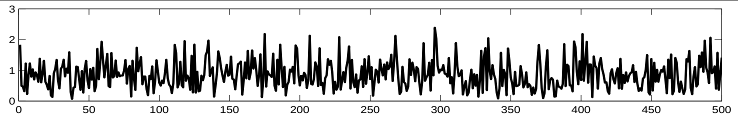
Title	**Data**	**Cite**
attas		[47]
ballbeam		[48]
balloon		[49]
buoy sensor		[50]
burstin		[51]

Table 6.1 continued

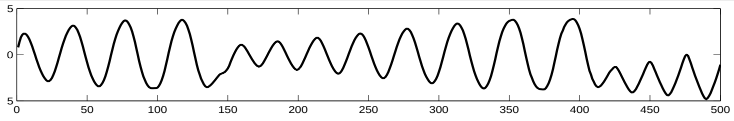
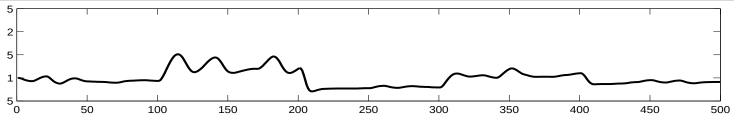
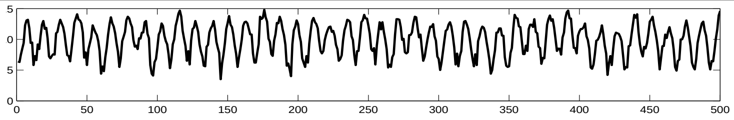
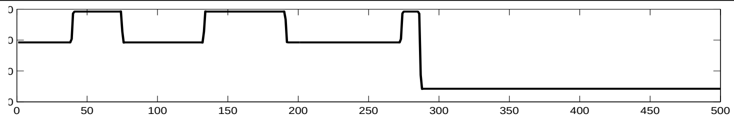
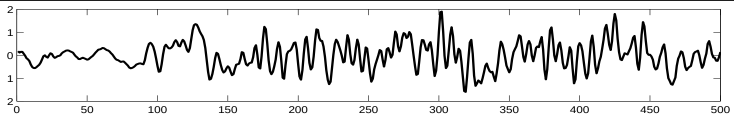
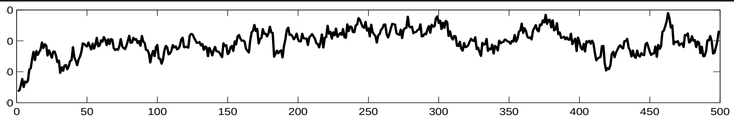
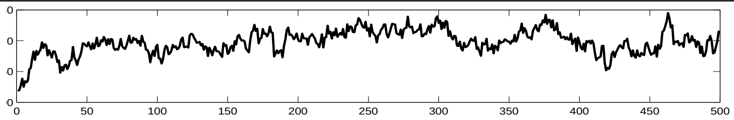
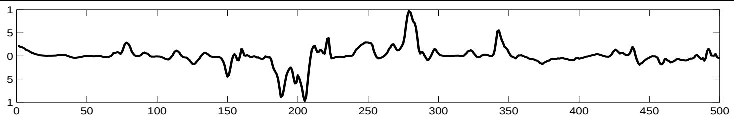
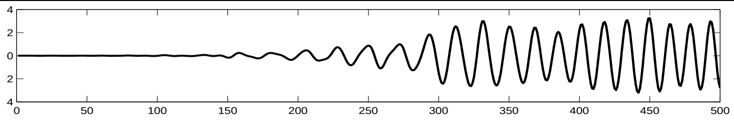
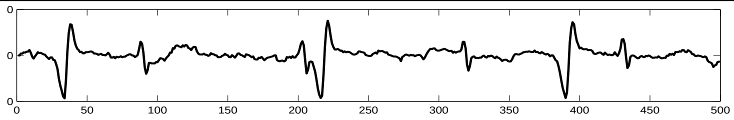
Title	**Data**	**Cite**
chaotic		[52]
cstr		[48]
darwin		[53]
dryer2		[48]
earthquake		[54]
eeg		[55]
evaporator		[48]
Fluid dynamics		[56]
flutter		[57]
foetal ecg		[48]

Table 6.1 continued

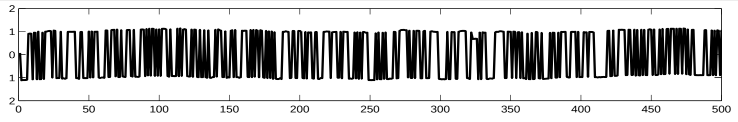
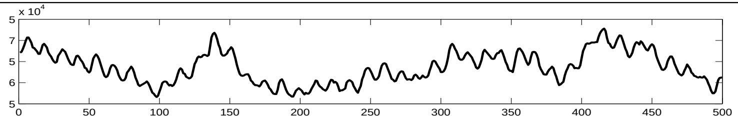
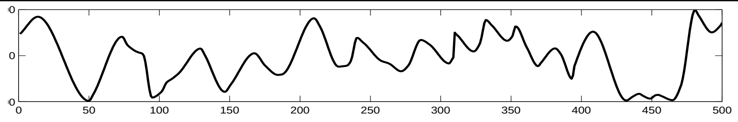
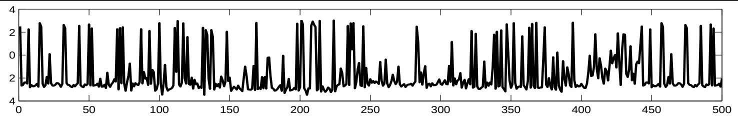
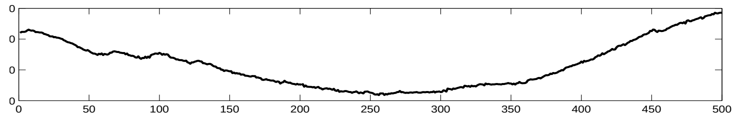
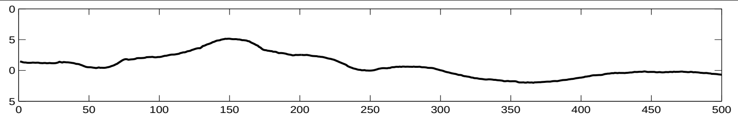
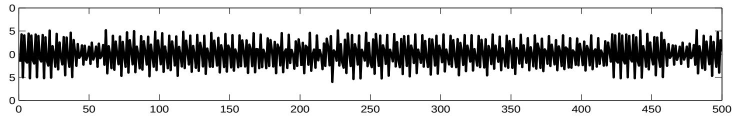
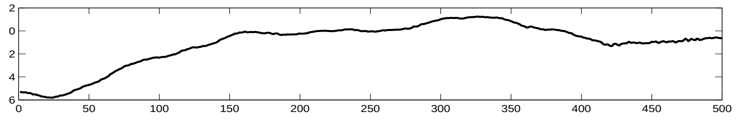
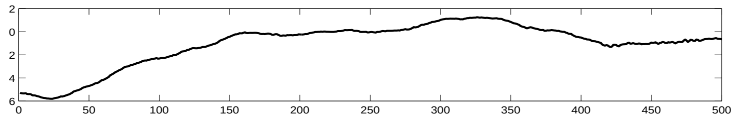
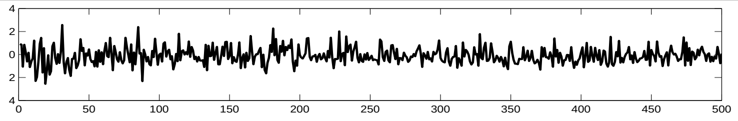
Title	**Data**	**Cite**
glassfurnace		[48]
greatlakes		[55]
infrasound beamd		[55]
leaf all		[55]
leleccum		[55]
memory		[55]
motorCurrent		[58]
ocean		[55]
ocean shear		[55]
pgt50 alpha		[1]

Table 6.1 continued

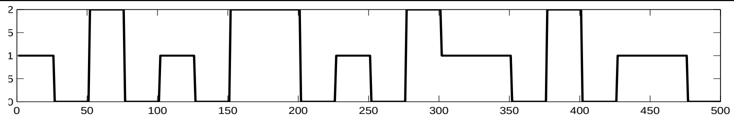
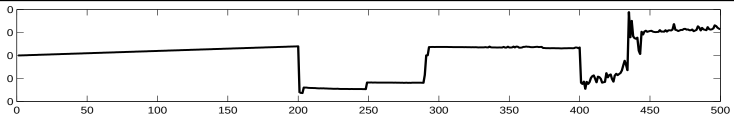
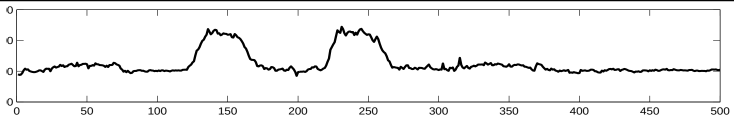
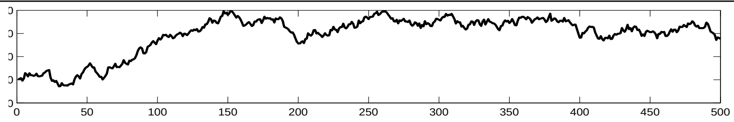
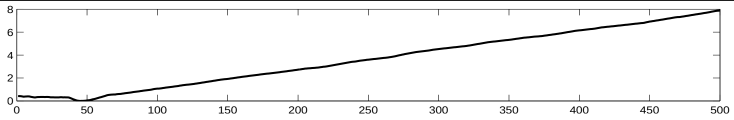
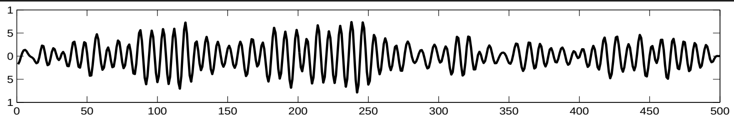
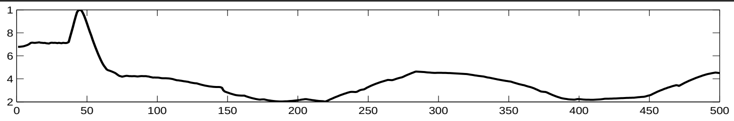
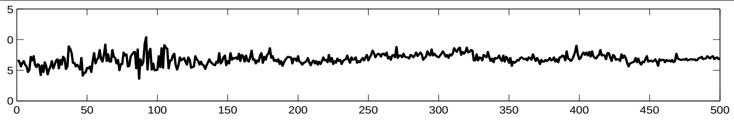
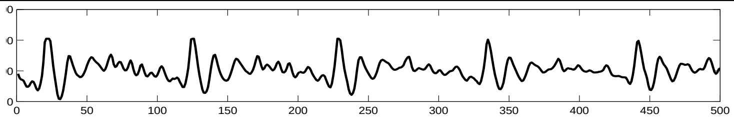
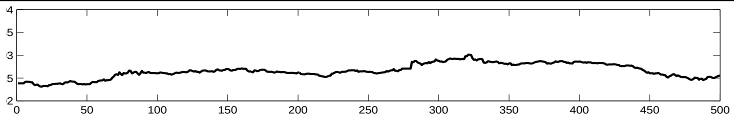
Title	**Data**	**Cite**
pHdata		[48]
powerplant		[48]
power data		[59]
random walk		[55]
Realitycheck		[55]
robot arm		[55]
shuttle		[60]
soiltemp		[54]
speech		[54]
spot exrates		[61]

Table 6.1 continued

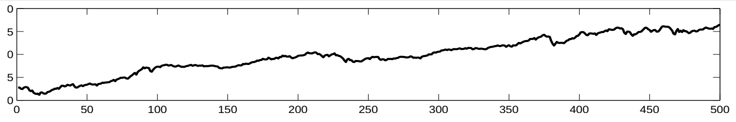
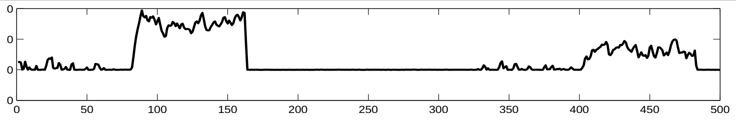
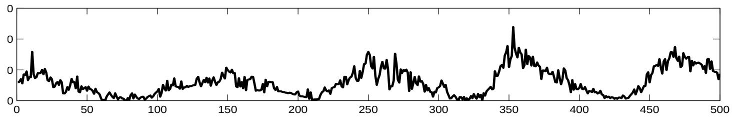
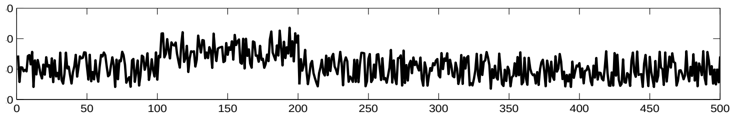
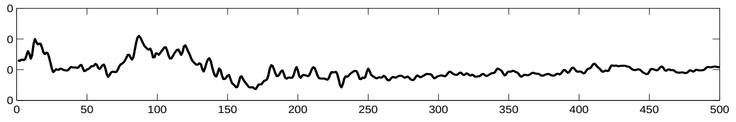
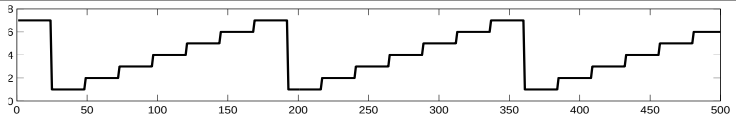
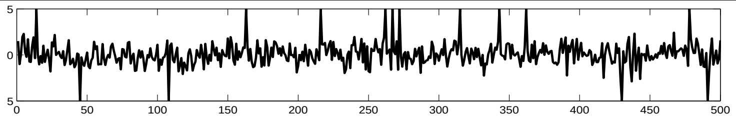
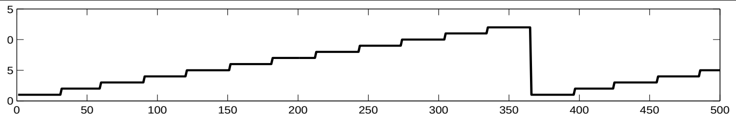
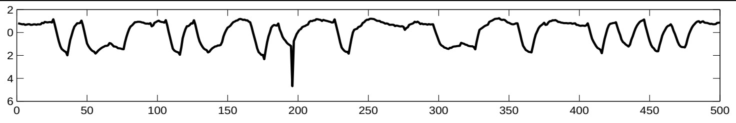
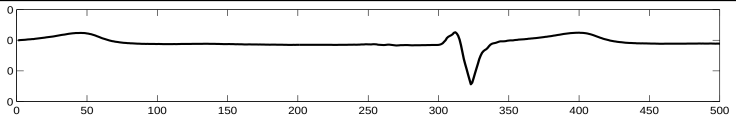
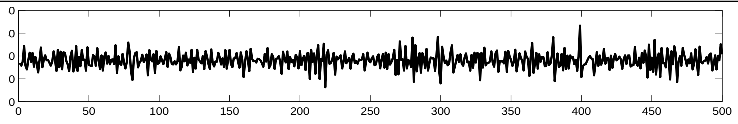
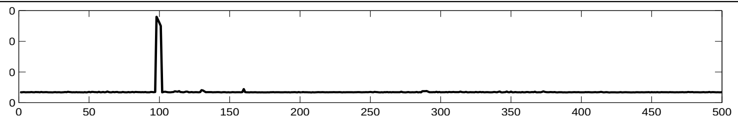
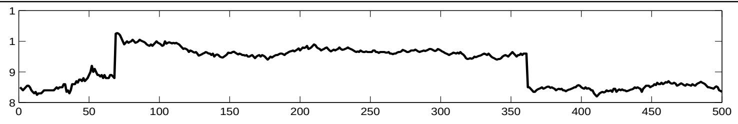
Title	Data	Cite
standardandpoor500		[49]
steamgen		[48]
sunspot		[62]
synthetic control		[63]
tide		[64]
TOR95		[55]
twopat		[48]
wind		[49]
winding		[48]
koski ecg		[65]

Table 6.1 continued

Title	**Data**	**Cite**
EEG heart rate		[55]
network		[55]
tickwise		[66]

Bibliography

[1] J. Aach and G. Church, “Aligning gene expression time series with time warping algorithms,” in *Bioinformatics*, pp. 17:495–508, 2001.

[2] “Apnea in wikipedia.” http://en.wikipedia.org/wiki/Apnea.

[3] “Tickwise data.” http://www-psych.stanford.edu/ andreas/Time-Series/SantaFe.html.

[4] R. Agrawal, C. Faloutsos, and A. N. Swami, “Efficient similarity search in sequence databases,” in *FODO ’93*, (London, UK), pp. 69–84, Springer-Verlag, 1993.

[5] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, “Fast subsequence matching in time-series databases,” in *Proceedings 1994 ACM SIGMOD Conference, Mineapolis, MN*.

[6] Y.-S. Moon, K.-Y. Whang, and W.-K. Loh, “Efficient time-series subsequence matching using duality in constructing windows,” *Information Systems*, vol. 26, no. 4, pp. 279–293, 2001.

[7] M. Kadous, “Grasp: Recognition of australian sign language using instrumented gloves,” 1995.

[8] E. J. Keogh, “Exact indexing of dynamic time warping,” in *VLDB*, pp. 406–417, 2002.

[9] Y. Zhu and D. Shasha, “Warping indexes with envelope transforms for query by humming,” in *SIGMOD ’03*, (New York, NY, USA), pp. 181–192, ACM Press, 2003.

[10] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh, “Indexing multi-dimensional time-series with support for multiple distance measures,” in *KDD ’03*, (New York, NY, USA), pp. 216–225, ACM Press, 2003.

[11] D. Gunopoulos, “Discovering similar multidimensional trajectories,” in *ICDE ’02: Proceedings of the 18th International Conference on Data Engineering*, (Washington, DC, USA), p. 673, IEEE Computer Society, 2002.

[12] D. Sankoff and J. Kruskal, *Time warps, string edits, and macromolecules : the theory and practice of sequence comparison.* Addison-Wesley Pub. Co., 1983.

[13] A. Guttman, “R-trees: a dynamic index structure for spatial searching,” in *SIGMOD ’84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 47–57, ACM Press, 1984.

[14] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, “The r*-tree: an efficient and robust access method for points and rectangles,” *SIGMOD Rec.*, vol. 19, no. 2, pp. 322–331, 1990.

[15] D. Berndt and J. Clifford, “Using dynamic time warping to find patterns in time series,” in *AAAI-94 Workshop on Knowledge Discovery in Databases*, (Seattle, WA, USA), pp. 229–248, 1994.

[16] R. L. R. Thomas H. Cormen, Charles E. Leiserson and C. Stein, *Introduction to Algorithms*. The MIT Press, 2001.

[17] S. Baase, *Computer Algorithms*. Addison Wesley, 1991.

[18] R. P. Grimaldi, *Discrete and Combinatorial Mathematics - An Applied Introduction*. Addison Wesley, 1989.

[19] L. Bergroth, H. Hakonen, and T. Raita, “A survey of longest common subsequence algorithms,” in *SPIRE ’00: Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE’00)*, (Washington, DC, USA), p. 39, IEEE Computer Society, 2000.

[20] H. Sakoe and S. Chiba, “Dynamic programming algorithm optimization for spoken word recognition,” pp. 159–165, 1990.

[21] F. Itakura, “Minimum prediction residual principle applied to speech recognition,” in *Readings in Speech Recognition* (A. Waibel and K.-F. Lee, eds.), pp. 154–158, San Mateo, CA: Kaufmann, 1990.

[22] S. Basu and M. Meckesheimer, “Automatic outlier detection for time series: an application to sensor data,” *Knowl. Inf. Syst.*, vol. 11, no. 2, pp. 137–154, 2007.

[23] G. R. George Box, Gwilym M. Jenkins, *Time Series Analysis: Forecasting and Control (3rd Edition)*. Prentice Hall, 1994.

[24] M. K. Jiawei Han, *Data Mining, Second Edition : Concepts and Techniques*. Morgan Kaufmann, 2005.

[25] B.-K. Yi, H. V. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *ICDE '98: Proceedings of the Fourteenth International Conference on Data Engineering*, (Washington, DC, USA), pp. 201–208, IEEE Computer Society, 1998.

[26] S.-W. Kim, S. Park, and W. W. Chu, "Efficient processing of similarity search under time warping in sequence databases: an index-based approach," *Inf. Syst.*, vol. 29, no. 5, pp. 405–420, 2004.

[27] R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. Shim, "Fast similarity search in the presence of noise, scaling, and translation in time-series databases," in *Twenty-First International Conference on Very Large Data Bases*, (Zurich, Switzerland), pp. 490–501, Morgan Kaufmann, 1995.

[28] E. J. Keogh and C. A. Ratanamahatana, "Exact indexing of dynamic time warping.," *Knowl. Inf. Syst.*, vol. 7, no. 3, pp. 358–386, 2005.

[29] S. Needleman and C. Wunsch, "A general method applicable to the search for

similarities in the amino acid sequence of two proteins.," *J Mol Biol*, vol. 48, no. 3, pp. 443–53, 1970.

[30] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195–197, 1981.

[31] "Blast." http://www.ncbi.nlm.nih.gov/BLAST/.

[32] "Blast tutorial." http://www.ncbi.nlm.nih.gov/Education/BLASTinfo/tut1.html.

[33] A. W. chee Fu, E. Keogh, L. Y. H. Lau, and C. A. Ratanamahatana, "Scaling and time warping in time series querying," in *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pp. 649–660, VLDB Endowment, 2005.

[34] S.-L. Lee, S.-J. Chun, D.-H. Kim, J.-H. Lee, and C.-W. Chung, "Similarity search for multidimensional data sequences," in *ICDE*, pp. 599–608, 2000.

[35] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "Segmenting time series: A survey and novel approach," 1993.

[36] E. Keogh, J. Lin, and W. Truppel, "Clustering of time series subsequences is meaningless: Implications for previous and future research," in *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, (Washington, DC, USA), p. 115, IEEE Computer Society, 2003.

[37] F. Höppner, "Discovery of core episodes from sequences," in *Proceedings of the ESF*

Exploratory Workshop on Pattern Detection and Discovery, (London, UK), pp. 199–213, Springer-Verlag, 2002.

[38] J. Lin, E. Keogh, S. Lonardi, and P. Patel, "Finding motifs in time series," in *Proceedings of the Second Workshop on Temporal Data Mining*, (Edmonton, Alberta, Canada), July 2002.

[39] P. Patel, E. J. Keogh, J. Lin, and S. Lonardi, "Mining motifs in massive time series databases," in *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, (Washington, DC, USA), p. 370, IEEE Computer Society, 2002.

[40] Y. Tanaka, K. Iwamoto, and K. Uehara, "Discovery of time-series motif from multi-dimensional data based on mdl principle," *Mach. Learn.*, vol. 58, no. 2-3, pp. 269–300, 2005.

[41] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth, "Rule discovery from time series," in *Knowledge Discovery and Data Mining*, pp. 16–22, 1998.

[42] B. Chiu, E. J. Keogh, and S. Lonardi, "Probabilistic discovery of time series motifs."

[43] J. Buhler and M. Tompa, "Finding motifs using random projections," in *RECOMB*, pp. 69–76, 2001.

[44] F. Hoppner, "Discovery of temporal patterns – learning rules about the qualitative behavior of time series," 2001.

[45] Y.-S. Moon, K.-Y. Whang, and W.-K. Loh, "Duality-based subsequence matching in time-series databases," in *Proceedings of the 17th ICDE*, (Washington, DC, USA), pp. 263–272, IEEE Computer Society, 2001.

[46] D. Gunopoulos, "Discovering similar multidimensional trajectories," in *ICDE '02: Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, (Washington, DC, USA), p. 673, IEEE Computer Society, 2002.

[47] J. Tikka and J. Hollmen, "Learning linear dependency trees from multivariate time-series data," in *Proceedings of the Workshop on Temporal Data Mining: Algorithms, Theory and Applications (in conjunction with The Fourth IEEE International Conference on Data Mining)*, (Brighton, U.K.), November 2004.

[48] "Data of the ball-and-beam setup in sista." http://www.esat.kuleuven.ac.be/ tokka/daisydata.html.

[49] "Data from a balloon about 30 kilometres above the surface of the earth." http://lib.stat.cmu.edu/datasets/.

[50] "Data for the camp current meter moorings." http://ccs.ucsd.edu/zoo/.

[51] "Data for elastic burst detection." http://cs.nyu.edu/cs/faculty/shasha/papers/burst.d/burst.html.

[52] "Chaotic data." http://cns-web.bu.edu/pub/cn550/project98/data/time_series_specification.html.

[53] “Monthly climate values of the darwin slp series from 1882 to 1998.” http://www.stat.duke.edu/ mw/ts_data_sets.html.

[54] “Earthwuake data.” http://lib.stat.cmu.edu/general/tsa/tsa.html.

[55] “Ucr dataset.” http://www.cs.ucr.edu/

[56] G. L. J. K. Brijesh Garabadu, Cindi Thompson, “Fast and accurate nn approach for multi-event annotation of time series,” vol. UUCS-03-021, 2003.

[57] F. Kuleuven, “Time-frequency analysis for transfer function estimation and application to flutter clearance,” *AIAA*.

[58] N. A. O. Demerdash and J. F. Bangura, “Characterization of induction motors in adjustable-speed drives using a time-stepping coupled finite element state-space method including experimental validation,” p. 790, IEEE Transactions On Industry Applications, 1999.

[59] “Cluster and calendar-based visualization of time series data.” http://www.win.tue.nl/ vanwijk/clv.pdf.

[60] “Robot arm data.” http://www-aig.jpl.nasa.gov/public/mls/time-series/.

[61] “spot prices (foreign currency in dollars)and the returns for daily exchange rates.” http://www.stat.duke.edu/data-sets/mw/ts_data/all_exrates.html.

[62] “Monthly sunspot data from jan 1749 to july 1990.” http://xweb.nrl.navy.mil/timeseries/multi.diskette.

[63] “Synthetic data.” http://kdd.ics.uci.edu/.

[64] “Tioed data.” http://lib.stat.cmu.edu/jasadata/percival-m.

[65] “Koski’s ecg data.” http://www2.cs.utu.fi/staff/antti.koski/abs.html.

[66] “Tickwise data.” http://www.stern.nyu.edu/ aweigend/Time-Series/Data/.