2002

Techniques to explore time-related correlation in large datasets

Sumeet Dua
Louisiana State University and Agricultural and Mechanical College

TECHNIQUES TO EXPLORE TIME-RELATED CORRELATION IN LARGE DATASETS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Sumeet Dua
B.E., Thapar Institute of Engineering and Technology (Deemed University), 1997
M.S. in Systems Science, Louisiana State University, 2000
May 2002

Dedicated to my late grandfather Shri. Bihari Lal Dua
for his vision and direction.

Acknowledgements

Many people were involved in the completion of this dissertation and my studies at the Department of Computer Science, Louisiana State University, Baton Rouge. I am grateful to my major professor, Dr. S. S. Iyengar for his expert guidance and support. His assistance and ideas on my research are really appreciated. Without his skilled direction, this dissertation would not have taken this shape.

I would also like to thank Dr. Jerry Trahan for his interest in my work and valuable discussions. His professional assistance and support were inestimable to the completion of this dissertation. I would like to express my appreciation to Dr. Aiichiro Nakano and Dr. Bush Jones for their professional apparition and sustained guidance. I am also grateful to Dr. Lynn R. LaMotte for agreeing to be a part of my committee and his tremendous support.

I consider this work as a joint effort in which my family shouldered the maximum responsibility. Thanks to sisters and brother-in-laws, Sonia Malhotra, Sunny Malhotra, Simmi Grover and Sanjeev Grover for their patience, blessings and encouragement. Last, but the most, my parents for whom there is so much to say but no words to say it in.

Table of Contents

Dedication……..……..ii

Acknowledgements…..……..iii

List of Tables..……..……..vi

List of Figures..……..……..vii

Glossary of Symbols and Terms ……..x

Abstract…….……xi

Chapter

1 Introduction……..…….……..1

1.1 Computational Structure of the Problem…….……..2

1.2 Scope of the Dissertation…….……..3

2 Related Work and Preliminaries…….…..……..7

2.1 Related Work…….7

2.1.1 Approximate String Similarity Matching……..……..7

2.1.2 Approximate String Matching in Databases..……..8

2.1.3 Searching Similarity in Protein Sequences……..9

2.2 Similarity Search in Data Mining……..9

2.3 Limitation of Previous Research……..10

2.4 Our Focus…….12

2.5 Organization of the Dissertation……..16

3 Whole Matching……..……..….17

3.1 Dynamic Overlay and Summation Technique (*DOLS*)…….. …17

3.1.1 Normalized Crosscorrelation…….22

3.1.2 Window Length…….24

3.1.3 Recursive Addition Updating…….25

3.1.4 Bias due to Re-occurrence of Input Data……...25

3.1.5 Global versus Local Maximization of Crosscorrelation……28

3.1.6 Actual Time Scale Modification Performed……..29

3.1.7 Computational Complexity…….29

3.1.8 An Example ……..…30

3.2 Whole Matching using *DOLS*……..…32

3.2.1 Notion of Similarity……..32

3.2.2 Normal Form of Sequences……..….34

3.2.3 Whole Matching…….……37

3.3 Summary……..39

4 Subsequence Matching……..40
4.1 Preliminaries……41
4.1.1 Notion of Similarity……..41
4.1.2 Discrete Fourier Transform……42
4.1.3 Normal Form of Sequences…..…….44
4.2 Outline of Proposed Approach……45
4.3 Piecewise Fourier Transform……...47
4.4 Grouping of Feature Points……..50
4.4.1 Answering Whole Match Query of size *w*…….51
4.4.2 Answering *Longer* Queries……..…..…52
4.4.3 Mahalanobis Distance……55
4.4.4 Grouping Algorithm……..….56
4.5 The *FG*-Tree……59
4.6 Summary……..61

5 Building the *FG*-index……..……62
5.1 Shape Quantification from Time Sequences……..……..62
5.2 The Minimum Enclosing Rectangle Algorithm……..63
5.3 Summary……..66

6 Similarity Search using *FG*-Index……67
6.1 Building the *FG*-Index…….67
6.2 Whole Match in Subsequences Query…..……69
6.3 *Similar Pairs* Subsequence Matching..……..……...70
6.4 Summary……..71

7 Experimental Results……72
7.1 Effect of *DOLS* Scaling on Harmonic Behavior……..…72
7.2 Running Similarity Queries using *DOLS*……..…78
7.3 Our Proposal versus Previous Work: A Case Study……82
7.4 Subsequence Matching using *FG*-Index……..…85

8 Conclusions and Future Directions…..……...91
8.1 Merits and Contributions…….91
8.1.1 Approach to Answer Similarity Queries in Unequal-Sized Sequence Databases...……91
8.1.2 Approach to Answer *Similar-pairs* Query in Sequences.……. …92
8.2 Applications…….93
8.3 Future Directions…….94

References…….……95

Vita……..100

List of Tables

4.1 Table of notations for groups…….57

6.1 *Distance*-table for 5 objects……...69

7.1 Database *DB* of 20 sequences……78

7.2 Query sequences for range query……..79

7.3 Summary of experimental parameters……...83

List of Figures

1.1 Subsequence matching problem can be converted into a whole matching problem by sliding a "window"(of length = L_Q) across a database sequence (length $\geq L_Q$) and making copies of data within windows……..3

2.1 Exchange rate data between Swiss franc and US dollar (Source: ftp://ftp.santafe.edu) and reconstructed sequence after employing Fourier transform, selecting the first k(=10) coefficients and then taking the inverse Fourier transform……..11

2.2 Algorithm employing whole matching linearly to obtain solution set for unequal-subsequence matching problem……...15

3.1 Control parameters for *DOLS* algorithm……..18

3.2 Dynamic Overlay and Summation Algorithm…….20

3.3 Overlay when $D_S < T$ …….26

3.4 Overlay when $D_S > T$ …….27

3.5 Overlay when $D_S = T$ …….27

3.6 (a) Plot of normal form of exchange rate data between Swiss Franc-USD (Source: ftp://ftp.santafe.edu) for 4624 days (b) Plot for time scale modified stream (a = 0.25)…… 31

3.7 (a) R: Exchange rate data between Swiss Franc and USD (Source: ftp://ftp.santafe.edu) (b) $\mathcal{N}(R)$: Normal form of Exchange rate data……..37

3.8 Whole matching algorithm for range query of threshold $\in$ …….38

4.1 Time series of length T, N values and its corresponding discrete Fourier transform representation……..43

4.2 (a) Algorithm *Extract_Subsequence* to extract all subsequences of size w (=5) from a sequence (b) Extraction of subsequences from a reference sequence……...….47

4.3 Amplitude log-plot of real exchange rate data versus increasing frequency……...….49

4.4 Answering whole matched range queries of size w…….51

4.5 Feature groups of fixed size (X = 3) each....……...….52

4.6 The grouping of segments using Euclidean distance as distance measure. Feature points are grouped together such that no two points have a distance greater than a threshold l_T …..……..53

4.7 Feature groups of fixed size of 3 each……..…54

4.8 Algorithm *group_feature* to form groups in feature space……..58

4.9 *FG*-tree for feature trail *RF*……..60

5.1 Process of derivation of MERs from a time sequence and extraction of geometric attributes from a MER…….65

6.1 *FG*-Index organization- the class identifiers are in shaded boxes....…….. 68

7.1 (a) Normal form of *random walk* (b) Its a -scaled Fourier transform amplitudes…..73

7.2 (a) *DOLS* scaled *random walk* (w = 31, a =0.5) (b) Its corresponding Fourier transform amplitudes……...…73

7.3 Plot of normalized mean square error between first *K* Fourier amplitudes for different values of *w* for random walk……..……..74

7.4 Plot of normalized mean square error between first *K* Fourier amplitudes for different values of a for random walk…….....75

7.5 (a) Normal form of exchange rate data (b) Its corresponding a -scaled Fourier transform …….75

7.6 (a) *DOLS*-scaled exchange rate (w = 31, a =0.5) (b) Its corresponding Fourier transform……..76

7.7 Plot of normalized mean square error between strong Fourier amplitudes for different values of *w* for real exchange rate data……..……..76

7.8 Plot of normalized mean square error between first *K* Fourier amplitudes for different values of a for real exchange rate data……..……..77

7.9 Euclidean distances of *DB* elements from *Q*1…….80

7.10 Euclidean distances of *DB* elements from *Q*2……..80

7.11 Euclidean distances of *DB* elements from *Q*3...……81

7.12 Euclidean distances of *DB* elements from *Q*4……...…81

7.13 Time per query versus number of Fourier coefficients…….84

7.14 Time per query versus length of each sequence……84

7.15 Percentage of false dismissals versus length of sequence……..……84

7.16 (a) – (e): # of matched subsequences at different levels of FG-tree versus window size for FG_{naive}, $FG_{Euclidean}$ and FG_{mahal} techniques employed for random walk; (e): Total # of matched subsequences versus window size for different methods……..…86

7.17 Total # of false alarms versus window size for FG_{naive}, $FG_{Euclidean}$ and FG_{mahal} techniques employed for random walk……88

7.18 (a) – (d): # of matched subsequences at different levels of FG-tree versus window size for FG_{naive}, $FG_{Euclidean}$ and FG_{mahal} techniques; (e): Total # of matched subsequences versus window size for different methods……88

7.19 Total # of false alarms versus window size for FG_{naive}, $FG_{Euclidean}$ and FG_{mahal} techniques employed for real exchange rate data……90

Glossary of Symbols and Terms

Symbol/term	Meaning
a	Time scale modification factor
$\hat{I}$	Threshold value for range query
$\subseteq$	Subset of or equal to
$\forall$	For all
$C_{p,q}$	Crosscorrelation between data sequences p and q of same length
D_A	Analysis displacement
DB	Database
DFT	Discrete Fourier transform
d_k	Shift determined for k^{th} window
DOLS	Dynamic overlay and summation technique algorithm
D_S	Synthesis displacement
FG-index	Feature group index
FG-tree	Feature group tree data structure
L_Q	Length of query sequence
L_R	Length of reference sequence
MER	Minimum enclosing rectangle (an enclosing rectangle of minimum area)
N	Number of sequences in a database of time sequences
Q	Query sequence
R	Reference sequence
S_{max}	Displacement shift in search interval
T	Accuracy overlap
w	Overlay weight

Abstract

The next generation of database management and computing systems will be significantly complex with data distributed both in functionality and operation. The complexity arises, at least in part, due to data types involved and types of information request rendered by the database user.

Time sequence databases are generated in many practical applications. Detecting similar sequences and subsequences within these databases is an important research area and has generated lot of interest recently. Previous studies in this area have concentrated on calculating similitude between (sub)sequences of equal sizes. The question of unequal sized (sub)sequence comparison to report similitude has been an open problem for some time. The problem is an important and non-trivial one.

In this dissertation, we propose a solution to the problem of finding sequences, in a database of unequal sized sequences, that are similar to a given query sequence. A paradigm to search pairs of similar, equal and unequal sized, subsequences within a pair of sequences is also proposed. We put forward new approaches for sequence time-scale reduction, feature aggregation and object recognition. To make the search of similar sequences efficient, we put forward indexing technique to index the unequal-sized sequence database. We also introduce a unique indexing technique to index *identified* subsequences within a reference sequence. This index is subsequently employed to report pairs of similar subsequences, when presented with a query sequence.

Our experimental results have depicted that relative amplitudes of first few frequencies (which are then employed for indexing) tend to behave similarly after time-scale reduction by using the proposed technique. The implementation has also exhibited about 27% reduction in query processing time, on a database of equal sized time sequences, of proposed approach over previous approach in this area.

Chapter 1

Introduction

Time series data is generated and recorded in various scientific, financial and medical databases. These time series could represent exchange rates, daily sales volume, average temperatures, bio-medical measurements, musical scores etc. The study of efficient algorithms for the analysis of this time series data has generated lot of interest recently [1,2,3,10,37] and numbers of significant advances in this field have been made. These advances have ranged from the development of algorithms to model time series data (see [10]), to the startling discovery of certain natural problems in the design of languages to query these data (see [37]). These results have kindled considerable interest in the study of approaches for similarity search in time series data.

There are many statistical tests one can perform on this data such as determining auto-correlation coefficients, measuring linear trends etc. Much of the utility of collecting this data also comes from determining similarity (for example, to a previously observed behavior) between these sequences. Some examples include the following.

- Financial analysts study stock market data, searching for certain behavior, which is perceived as indicative of stock's future performance (see [46]).
- Geophysicists constantly monitor seismic energy recorded by vibrator arrays at periodic intervals to look for certain regularities that may be an indication of new seismic event such as an earthquake (see [27]).
- Musicians might be interested in looking for musical scores similar to a given copyrighted score (see [42]).

In the above examples it is necessary to search a time series database for those series that are similar to a given query sequence. Similarity search is useful in its own right as an independent tool for exploratory analysis in time series databases.

This primitive is also needed, for example, for prediction [43], clustering [11] and classification [30, 38] purposes.

An important feature of time series data is that two sequences are rarely identical (same); therefore approximate matching is a promising means of comparing two sequences.

One of the major goals of this dissertation is to extend existing problem definitions (proposed in [2]) of detecting pairs of similar equal-sized (sub)sequences, to the problem of efficiently detecting pairs of similar unequal-sized (sub)sequences.

1.1 Computational Structure of the Problem

To analyze the general problem of similarity search in time-sequenced data, it is necessary to develop a good understanding of the existing problem definitions developed to search these data. Our thesis begins by presenting these definitions, which would establish our motivation to generalize them to accurately reflect, to the maximum extent possible, salient features of practical time-sequence data. Majority of this work has specifically originated from the scientific community in the area of data mining. Most of the research has focused on the following problems.

1. **Whole Matching:** The sequences to be compared have the same length *n*. Given a query sequence *Q* of length *n* and a database[1] *DB* of *N* data sequences, all of same length *n*, find first (or all) data sequence(s) in *DB* that match *Q* approximately. The following example illustrates the problem.

   Example 1.1: Given a database of daily average temperatures of ten regions for the month of July 1998 and a query sequence of daily average temperature of a query region for the month July 1998. User might be interested to find in this history (of 10 regions), the behavior of a region whose temperature was similar to the given sequence for the query region.

2. **Subsequence Matching:** The query sequence is smaller and we look for subsequence(s) (of same size as query sequence) in a database of larger sequences that best matches the query sequence. Given a query sequence *Q* of

[1] Term "database", here, is used in a broader sense not necessarily involving a restricted model (e.g. a relational model).

length L_Q and a database DB of N sequences, each of length L_i ($1 \le i \le N$), find first (or all) occurrences of a contiguous subsequence within DB that matches Q approximately. The following example illustrates the problem.

Example 1.2: Given a database of time sequences of daily average temperature for ten regions (region1 thru region10) from September 1967 to August 1999. User might be interested in finding if in this history the behavior of query region in a period of June 2, 1991 to June 27, 1991 (25 days) was similar to any region's behavior (for 25 days) in the database.

Note that it is possible to convert subsequence-matching problem to whole matching sequence problem by sliding a "window" of length L_Q across each sequence (of length L_i; $1 \le i \le N$, $L_i \ge L_Q$) in DB, making copies of (L_i-L_Q+1) windows within this sequence and congregating them in a database DB`. Figure 1.1 illustrates the idea.

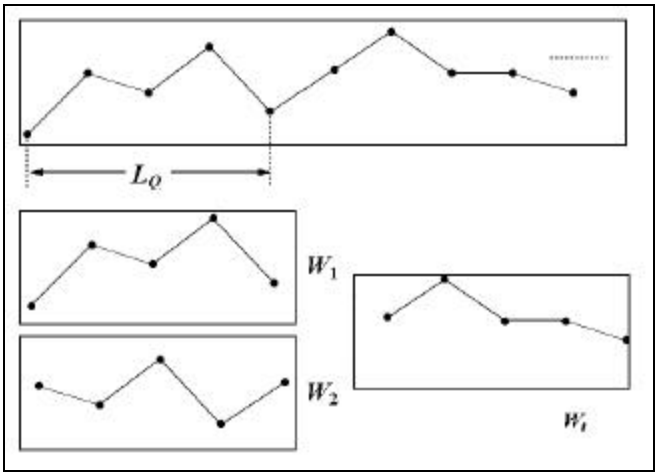


Figure 1.1: Subsequence matching problem can be converted into a whole matching problem by sliding a "window"(of length = L_Q) across a database sequence (length $\ge$ L_Q) and making copy of data within windows.

1.2 Scope of the Dissertation

An important feature of whole matching problem, as defined above, is that the sequences in the database and the query sequence are assumed to be of equal sizes. However in reality, we frequently encounter sequences of unequal sizes. Examples include but are not limited to the following.

- Daily closing stock prices of a 2-year-old company versus historical data of a 5-year-old company to determine whether these behaviors resemble each other.
- Different sampling rates used for discretization of musical scores while detecting possible copyright infringement (see [42]).
- Satellite sensor data recorded at different rates to measure a same phenomenon (see [8]).

One might be interested in whole matched queries for these unequal sized sequences.

This brings us to the focus of the first part of this dissertation. We propose a solution to the whole matching problem for unequal sized time sequence database. In this case, given a query sequence Q of length L_Q and a database DB of N data sequences, each of length L_i ($1 \le i \le N$), we are interested in finding first (or all) data sequence(s) in DB that match Q approximately.

In many applications it is also desired to find pairs of similar subsequences within two sequences. These pairs can indicate an embedded similar behavior within the sequences, occurring at previously unknown positions and can be of unknown (unequal) sizes. The problem is challenging because these pairs of unequal-sized subsequences can be overlapping (within each sequence). We call this problem as a *similar-pairs* query in subsequence matching problem, where "similar pairs" refers to pairs of similar subsequences in given reference and query sequences.

Previous work (see [16]) have presented this problem as a problem that can be solved by applying whole matching techniques sequentially[2] over all possible subsequences[3] in reference and query sequence. This solution is not sufficiently efficient since it sequentially scans over all possible subsequences in reference sequence, which can be very large. It also may not be required to compare subsequences less than a certain specified size. Moreover, users might be interested in finding subsequence pairs of unequal sizes. This problem is an interesting one and needs an independent solution not necessarily built upon sequential scanning techniques.

[2] Section 2.4 presents this approach and discusses its demerits.
[3] The approach presented by authors in [16] does not find subsequence pairs of unequal sizes.

In the second part of the dissertation, we propose a solution to the above-mentioned problem defined as follows. Given a reference sequence R of size L_R and a query sequence Q of size L_Q, find pairs of subsequences ($[R_i \ldots R_n]$, $[Q_j \ldots Q_m]$ where $1 \le i \le n \le L_R, 1 \le j \le m \le L_Q$) in R and Q respectively, that match each other approximately. The approximate match is defined by a degree of closeness.

An approach to handle the above mentioned problems should contain a paradigm to calculate similarity within pairs of unequal-size (sub)sequences. However, time sequence databases are often extremely large. Given the magnitude of these databases, the techniques for scanning the complete whole sequences and entire database for similitude can be both time and computationally expensive. Hence, it is desired to reduce the dimensions, and consequently search space of the database for efficient query processing.

In order to achieve a faster search methodology, we propose index structures to index the database (of unequal sequences, in the whole-matching problem) and subsequences in a reference sequence (for *similar-pairs* query problem), such that one can search the index, having lesser dimensions than the database, instead of searching the sequence database in its entirety (or all candidate subsequences in a reference sequence) for retrieval of similar pairs.

Our proposed indexing technique for similar-pairs query can detect embedded subsequences when their positions are previously unknown. Also, the number of locations in which matching is done depends on the number of objects discovered in a sequence and not on its size.

We have tested our proposed framework on real and synthetic data. In one experimental setup, the proposed indexing technique for an equal-sized sequence database has shown about 27% reduction in query processing time in comparison to the technique proposed by authors in [2]. In another experimental routine, our indexing technique for *similar-pairs* query has shown to detect 75% of embedded

similar pairs in synthetic (random walk) and 85% similar subsequence pairs in real exchange rate data with almost equal number of false alarms[4].

[4] False alarms in index search refer to those records that are retrieved during the index search and are not a part of the answer set.

Chapter 2

Related Work and Preliminaries

Similarity search has gathered interest from many researchers in both academic and industrial communities, especially in recent years. In this chapter we will discuss different approaches in this area. It includes traditional techniques on similarity search and similarity search in the area of data mining, which forms the background of our proposed approach. We conclude the chapter with our proposal, which lays groundwork for our discussion in subsequent chapters.

2.1 Related Work

There are many different similarity search algorithms that are determined by the kind of search space to deal with, that is, specific domain data and type of interest. The work in text retrieval and pattern recognition that deals with matching characters and patterns is usually considered to be searching in discrete space. The problem of searching similarity in a database of time series of real numbers is considered to be searching in continuous space. We begin our discussion with the problem of similarity search in text data.

2.1.1 Approximate String Similarity Matching

There has been a lot of work on finding text subsequences that approximately match a given string [20,25]. Text sequences normally consist of few discrete symbols as opposed to continuous numbers that makes the similarity measures and the search methods quite different.

The algorithmic problem of finding all occurrences of a given string, usually called the pattern, in another larger text string is called the *exact string-matching problem*. Sometimes, it is difficult to find an occurrence of exact match to a pattern within the larger text string. In these cases, it is desired to find an *approximate* match to a pattern rather than an exact match. Generally the concept of *edit distance* is employed to measure the goodness of approximate occurrence(s) of a pattern. The

edit distance between two strings, A and B in alphabet S, can be defined as the minimum number of editing steps needed to convert A to B. Each editing step is a rewriting step of the form $a \rightarrow \varnothing$ (a deletion), $\varnothing \rightarrow b$ (an insertion), or $a \rightarrow b$ (a modification) where a, b are in Σ and $\varnothing$ is the empty string. Assuming the cost of each of these operations is unity, the edit distance is the minimum number of operations needed to obtain a pattern from a text. The *k-differences approximate string matching* problem is to find the occurrences of substrings of a text string T whose edit distance from a pattern string P is less than k.

A dynamic programming solution for this problem is given in [29] that computes the solution in O(mn) time, where m and n are the size of text and pattern respectively. There are other algorithms (see [20, 25]) similar to one in [29] with running time O(pn) ($p<n$). However, there is no single method, which is always the fastest. The speed of these algorithms varies according to the alphabet size, the series length and the value of p. The number of possible pairs for similarity calculations is gigantic in case of subsequences within large sequences. There does not exist a direct approach exists to extend these methods to employ a faster methodology, such as, indexing of subsequences within a larger text string or text sequences within a large database of sequences to prune the search space.

2.1.2 Approximate String Matching in Databases

Approximate string matching in one-dimension for a database of text strings has been attracting increasing interest lately. Motro described a user interface for vague queries [37]. The author presents a design of an interface for a relational database management system (RDBMS) to accommodate fuzzy and *non-exact* queries directly, but does not discuss the similarity notions applied at the database level. Shasha and Wang [51] proposed a database indexing method that uses the triangular inequality and some precomputed distances to prune the search. However, the space overhead of the method is quadratic on the number of objects, which makes it prohibitive to use for large databases.

Related effort, in this category but not directly applicable to numerical sequences include, algorithms for approximate string searching with full text scanning [5] and clustering algorithms in information retrieval and library science [48].

2.1.3 Searching Similarity in Protein Sequences

In molecular biology, a distance metric called evolutionary distance is aimed to measure the distance between two protein sequences. This evolutionary distance is similar to the edit-distance defined for the text string matching problem (Section 2.1.1).

If the evolutionary distance between two sequences is less than a threshold, they are more likely homologous, which means a historical path of evolutionary steps join them. To discover the function of a new sequence[1], biologists usually need to search the database of all previously known sequences for possible homologies. These discovered similar pairs might share similar functions and play similar roles. Protein similarity search finds these sequences, providing new clues and insights to the function of a query protein.

Protein Similarity Search employs a multitude of different search methods, depending on the nature of the query. The computational algorithms used, and their specific configurations are pre-set according to the specific type of research question being posed (see [34, 33, 4]).

2.2 Similarity Search in Data Mining

Discovering similar patterns in large time-series datasets has been generating a lot of interest from the scientific community in the area of data mining. The problem is an important and non-trivial one, on which a lot of work has been done [1,2,16,50].

To the best of our knowledge, [2] is the first work that proposes a solution for similarity matching of sequences in time series data. In [2], it is assumed that all

[1] The function of a protein sequence is related to their secondary structure, which in turn depends on to the amino acid sequence they contain. Each amino acid sequence is a unique combination of characters. Also see [29,4].

sequences are of the same length (n), and each sequence is considered as a point in an n-dimensional space. Then, two sequences are considered similar when the Euclidean distance between them is less than a threshold value[2] ϵ. The Discrete Fourier Transform (DFT) is used for extract features from these sequences since it preserves the Euclidean distance[3]. Sequences are then represented as k-dimensional points using k features for each sequence. The first k terms after the transformation are chosen to represent a sequence. There are other related works on similarity queries over time-series data. Faloutos *et al.* [16] describe an extension of this work for subsequence matching. There are more related works on querying time series data, but they usually do not consider similarity comparison as a query language operator. For example, Seshadri *et al.* [50] develop a data model and a query language for sequences in general. A specific query language for time series data in the stock market domain is developed by Roth [46].

2.3 Limitation of Previous Research

The existing research to calculate similitude in time series datasets as discussed above suffers from the following limitations.

1. Similarity is measured for sequences of the same length. For whole matched queries (Section 1.1) the database is assumed to consist of sequences of the same length as that of the query sequence. Similarly, for the subsequence matching problem the retrieved sequences (answer set) are of the same size as the query sequence (see [2], [16]).
2. The technique presented in [2] for whole matching is not well suited for sequences that have large amount of short-term peaks and valleys. As indicated before, authors in [2] propose to extract first k features from the whole sequence in a database and use these k features to index them[4]. Now, consider a sequence of exchange rates between Swiss franc and US dollar depicted in Figure 2.1.

[2] ϵ is user specified here.
[3] Distance preservation is true for any orthonormal transformation. Other examples of orthonormal transformations include DCT(discrete cosine transform) and wavelet transform.
[4] The typical values of k recommended by the authors in [2] are 2,3,4 and 5.

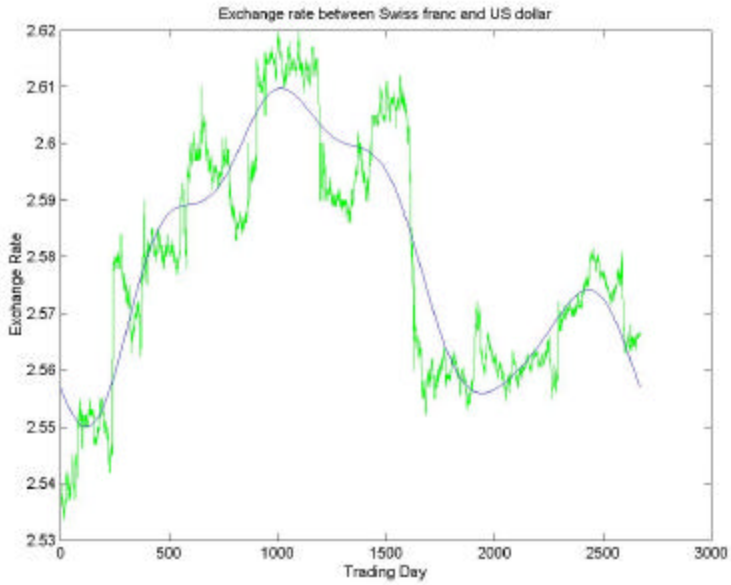


Figure 2.1: Exchange rate data between Swiss franc and US dollar (Source: Public site at ftp://ftp.santafe.edu) and reconstructed sequence after employing Fourier transform, selecting the first k(=10) coefficients and then taking the inverse Fourier transform.

The reconstructed sequence after selecting the first k (k=10) coefficients in DFT and employing inverse Fourier transform is superimposed. Note that most short interval peaks and valleys are lost in the reconstructed signal. We believe many practical sequences contain such transient behavior. For example, short interval jumps or valleys in stock market data and turbulent behavior in sensor data (see [8]). This short-term transient behavior in long sequences cannot be well represented in first few (k) Fourier coefficients.

Time series databases traditionally are huge in sizes. The computational load for processing similarity queries is thus high. This load is escalated by the fact that these data sets extend themselves in high dimensions. Research in multi-dimensional spatial data structures (see [19] for a survey) has exhibited that query time on data organized in a spatial data structure, such as state of the art R* trees (see [19]), scale exponentially for high dimensions, eventually reducing to sequential scanning within

this data. For example, for a spatial data structure such as linear quadtrees[5], the computation is proportional to the area of hypersurface of the query region. And this hypersurface grows exponentially with increasing dimensionality (see [16]). This paradigm is commonly known as "dimensionality curse" in time series databases.

Hence, it is desired to reduce the dimensions of these sequences to reduce the computational cost of processing similarity queries. A feature extraction technique, such as discrete Fourier transform, can be employed to extract features, such that few features are sufficient to differentiate between objects.

We propose two techniques for dimensionality reduction. One is called dynamic overlay and summation technique (abbreviated *DOLS*). Another technique that we propose is piecewise Fourier transform followed by a minimum enclosing rectangle (MER) approximation. Eventually, we aim at developing a technique to handle the limitation of discovering same length similar (sub)sequence pairs. We consequently employ our framework to a variety of real stock and synthetic data to discuss experimental results.

2.4 Our Focus

Let us consider a generalized similarity query defined for a database of *Objects*. The *Objects* of our interest are time sequences.

Let *distance* be a metric distance function for a pair of *Objects*, that is, *distance*: $Objects \times Objects \rightarrow R_0^+$ and *distance* satisfies the following conditions $\forall\ O_1, O_2, O_3 \in Objects$.

1. $distance(O_1, O_2) = 0 \Leftrightarrow O_1 = O_2$;(identity).
2. $distance(O_1, O_2) = distance(O_2, O_1)$; (symmetry).
3. $distance(O_1, O_3) \leq distance(O_1, O_2) + distance(O_2, O_3)$; ($\Delta$ – inequality).

Definition 2.1: Similarity Query

Let $DB \subseteq Objects$ be a database and let $Q \in Objects$ be a query object.

[5] A tree where each node is split along all d dimensions, leading to 2^d children (see [19] for a detailed description).

The specification of the query type *T* on *DB* consists of three components.

1. *T.range*: A real number specifying a maximum *distance* between *Q* and an answer.
2. *T.cardinality*: An integer defining the maximum cardinality of the set of answers.
3. *T.kind*: A string[6] containing additional information for combining the range condition and the cardinality condition, to report the answer set.

Let *T* denote the type (specified by the above three components) of the similarity query and let sim_T:*Objects*×*Objects* ? *Boolean* be a predicate defining the similarity of pairs of objects with reference to the type *T*. A similarity query, denoted as *DB.similarity_query(Q,T)*, returns the following database objects:

$$DB.similarity\_query(Q,T) = \{O \in DB \mid sim_T(O,Q)\}.$$

We are especially interested in approximate matching in which the retrieved sequence(s) are within a user-defined threshold distance $\in$. $\in$ can also be calculated automatically, for example, ten percent of the energy of the query signal[7]. We assume, without the loss of generality, that $\in$ is user specified. Such queries are called range queries. More formally,

Definition 2.2: Range Query

A range query with respect to a database $DB \subseteq Objects$ and a query object $Q \in Objects$ is a similarity query with $T.range = \in$, $T.cardinality = +\infty$, and *T.kind* = "NULL", which returns the following subset of database objects:

$$DB.similarity\_query(Q,T) = \{O \in DB \mid distance(O,Q) \leq \in\}.$$

[6] *T.kind* is a descriptive text string that can contain additional information, for example, on
1. How to filter the answer set such that certain data types are not included?
2. How to select objects when *T.cardinality* is less than the number of objects discovered by the range condition?

[7] Signals and sequences, in this dissertation, have been used interchangeably. They both refer to data streams that have distinct values at discrete intervals of time.

We employ range query for the whole matching problem for unequal sequences[8], which is direct analogous to the above definition. This is the focus of first part of the dissertation.

Definition 2.3: Whole Matching Problem for Unequal Sequences

Given a query sequence Q of length L_Q and a database DB of N data sequences. Each sequence in database is denoted by R_i $(1 \le i \le N)$ and is of length L_i $(1 \le i \le N)$. Let $T_i (1 \le i \le N)$ and T_Q be transformations that transform sequences R_i $(1 \le i \le N)$ and Q, such that a distance metric function can be computed between them[9]. Find all data sequence(s) in DB that qualify within range query of range $\in$ with respect to transformed database $DB_T \subseteq T_i(R_i)$ $(1 \le i \le N)$ and $T_Q(Q)$.

Our second problem of interest is the subsequence matching problem for two sequences. We desire to find pairs of similar sub-sequences within the given two sequences. The similar subsequence pairs in the answer set can be of unequal sizes. The problem is called *similar-pairs* subsequence matching query (or just *similar-pairs* query) defined as follows.

Definition 2.4: *Similar-pairs* Subsequence Matching Query

Given a reference sequence R of size L_R and a query sequence Q of size L_Q, find all subsequences $[R_i \ldots R_j]$, $1 \le i \le j \le L_R$ in R and subsequences $[Q_m \ldots Q_n]$, $1 \le m \le n \le L_Q$ in Q that match each other approximately (within certain threshold *distance*).

It might seem straightforward that problem of subsequence matching for detecting all similar pairs of subsequences within reference and query sequence can

[8] 'Whole matching' in the remaining part of the dissertation will refer to the generalized problem of similarity search in a database of sequences, when no assumption on the size of sequences in the database or query sequence is made.

[9] For example, we later (in Section 3.2.1) propose to use Euclidean distance as the measure to compute similarity between sequences. Euclidean distance is defined for sequences of same length. Hence a transformation employed on query and database sequences, in this case, should ensure that these sequences are converged to a common size.

be represented as a combination of M whole-matching problems where M is the number of all possible subsequences in Q. Figure 2.2 gives an algorithm for linear scanning based on this approach, to retrieve all (equal and unequal) similar subsequence pairs within R and Q.

Algorithm *scan_all*($R, Q, \in$)
Begin
/* Build database */
For a = 2 to size(R) do
    For each (i^{th}) subsequence sub_i^a in R of size a
        Add sub_i^a to $DB`$,
    End For.
End For.
/* Run Whole match query for all possible subsequences in a query sequence */
For b = 2 to size(Q) do
    For each (j^{th}) subsequence sub_j^b in Q of size b
        $result$ = whole_match($DB`$, sub_j^b, $\in$),
        If $result \neq NULL$
            Output $result$,
        End If.
    End For.
End For.

Figure 2.2: Algorithm employing whole matching linearly to obtain solution set for unequal-subsequence matching problem.

The approach presented in *scan_all* is simple. First, a database of all possible subsequences in a reference sequence is built. Then for each subsequence in the query sequence, a whole-matching query is posed on the database. In other words, all possible subsequences in Q are visited and compared with all possible subsequences within reference sequence to calculate similitude and answer the *similar-pairs* query. If the length of the reference sequence is L_R and the query sequence is of length L_Q, then the number of possible candidate pairs for similarity evaluation is ((L_R -1)!*(L_Q-1)!)/2. Obviously it is not a very efficient way to do *similar-pairs* subsequence

matching because these sequences can be very large, scaling the number of comparisons required for match. Secondly, not all subsequences may be required to be compared for calculating similitude. For example, in musical scores subsequences of certain size may not represent an identifiable unit of score. We obviously desire an improved solution.

In the second part of the dissertation we propose a computational framework to handle this *similar-pairs* query problem in subsequence matching.

2.5 Organization of the Dissertation

The rest of the dissertation is organized as follows.

In Chapter 3 we present our solution to the whole matching problem for unequal sequences. The chapter presents an algorithm for time-scale reduction and discusses how this algorithm can be employed for whole matched queries.

Chapter 4 introduces subsequence matching for an unequal sized answer set. We present the notion of similarity employed and outline of proposed approach. A grouping algorithm based on distance metric is also presented. The chapter concludes with a discussion on proposed *FG*-tree (Feature Group tree).

In Chapter 5 we present ideas on pose extraction of objects in time series. Chapter 6 presents a paradigm to build an *FG*-index based on geometric descriptors of objects in time series. This index is then employed to answer whole matched subsequence matching and *similar-pairs* query.

Chapter 7 presents some experimental analysis and results. We conclude our discussion in Chapter 8 with the summary of contributions and results, and possible applications. We also present some directions for future research.

Chapter 3

Whole Matching

In this chapter we present an approach called Dynamic Overlay and Summation (*DOLS*) technique. This is subsequently employed to handle the whole-matching problem for unequal sequences defined in Section 2.4. Given a query sequence Q of length L_Q and a database DB of N reference sequences R_i, each of length L_i ($1 \le i \le N$), we are interested in finding all data sequences in DB that match Q approximately.

In the next section we propose a technique that reduces the size of a data sequence by reducing redundancies by iteratively scanning a window over a sequence, aligning and summing overlapping areas of maximum correlation. A detailed discussion follows.

3.1 Dynamic Overlay and Summation Technique (*DOLS*)

Time scale modification (TSM) is a class of algorithms to modify the time scale of a data stream (see [44,47,32,28]). Time scale modification can involve either expansion or compression of a data stream. Correspondingly, there is an associated parameter α, called TSM factor. When α is one, no time-scale modification is performed[1], when $\alpha > 1$ the signal is expanded and $\alpha < 1$ indicates time compression.

We are particularly interested in time scale compression. We want to represent our sequences with fewer data points to facilitate faster similarity search. We present a new time scale compression technique, that allows us discover and reduce redundancies in the data. The algorithm operates by segmenting the input data sequence with different interframe shifts to construct a time-scale modified stream. The input stream segments are aligned to maximize the correlation of overlapping segments before they are *weighed* and summed.

[1] Using $\alpha = 1$ in our technique does modify the original signal without changing the scale of sequence. The resultant is a smoothed version of the original signal with the same time-scale.

Before we formally present our algorithm a few definitions of the algorithm's control parameters are in order.

Definition 3.1: Window Length (*W*)

The duration of the windowed segments of the input data stream. This parameter is identical for both the input and output buffers, and represents the smallest unit of time-segment manipulated by the algorithm.

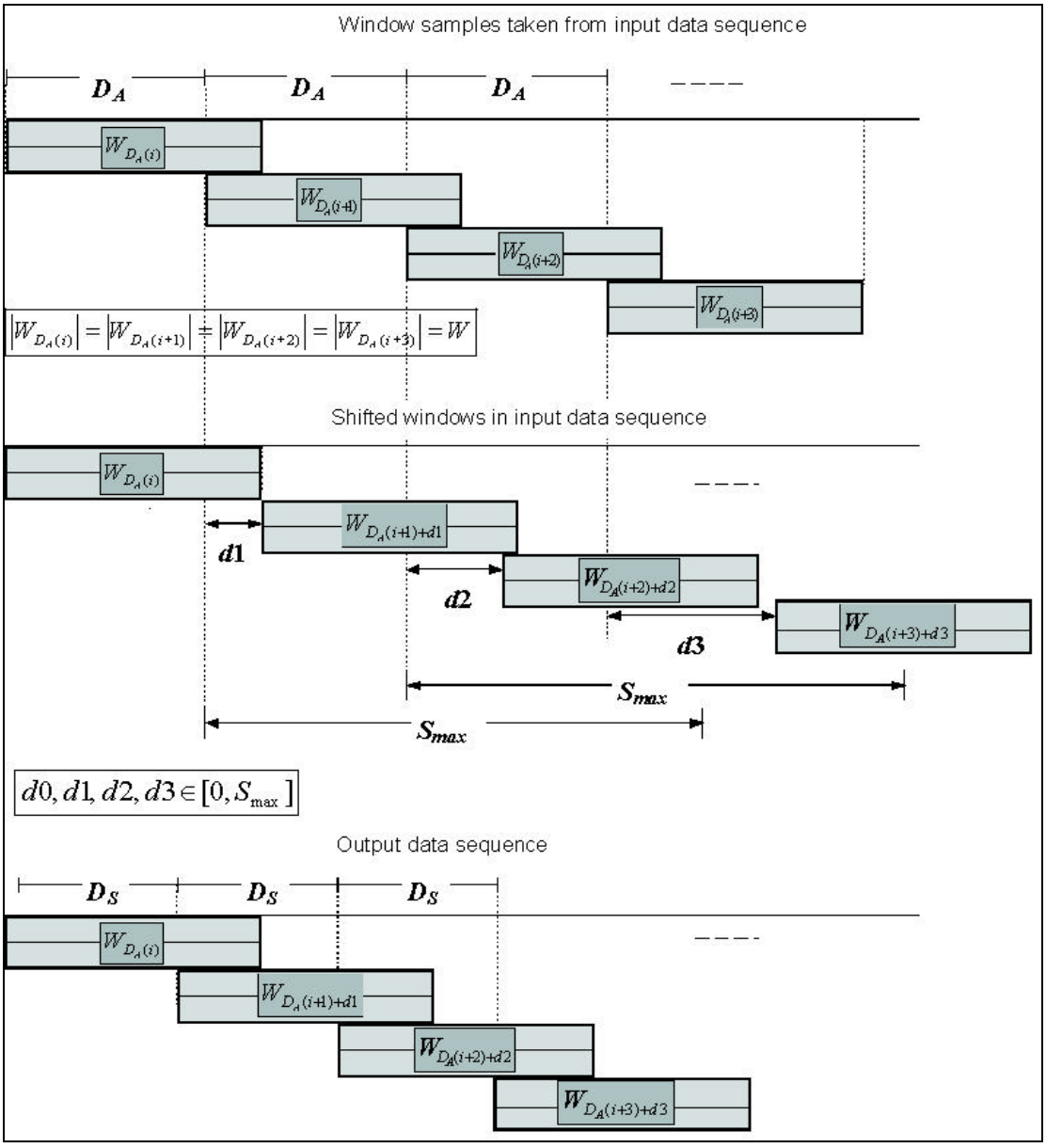


Figure 3.1: Control parameters for *DOLS* algorithm.

Definition 3.2: Analysis Displacement (D_A)

The interframe interval between successive windows along the unshifted (*raw*) input data stream.

Definition 3.3: Synthesis Displacement (D_S)

The interframe interval between successive windows placed along the output (*synthesized*) data stream.

Definition 3.4: Displacement Shift in Search Interval (S_{max})

The duration of the interval over which a window from the input sequence may be shifted for alignment with previous windows.

Definition 3.5: Accuracy Overlap (T)

Specifies the degree of overlap between the current and previous window. In the algorithmic implementation, the first T samples of each window overlap with last T samples of the previous window.

Figure 3.1 diagrammatically depicts these parameters.

We also assign a weight w ($0<w\leq 1$) called *overlay weight* to the elements of the current input data window[2] that overlay the output data sequence. Similarly, a weight of ($1-w$) is assigned to the overlapping portion of the output data sequence (last T elements) before they are summed with the weighed overlapping portion of the input data window.

The formal algorithm is shown in Figure 3.2. Procedurally, the *DOLS* algorithm modifies the time-scale of the data stream in basically three phases: initialization, analysis and synthesis.

Initialization phase: The algorithm begins by copying the first input sequence window of size W to the output sequence.

[2] This window is appropriately shifted before its first T elements are allowed to overlap the last T elements of the output data sequence. See discussion following the algorithm definition.

Algorithm *DOLS*

Input:

Mandatory input parameters

1. Input sequence *X*.
2. Window size *W*.
3. TSM Factor a .

Optional input parameters

4. Analysis displacement D_A (Default value = $\frac{W}{2}$).
5. Overlay weight *w* (Default value = $\frac{1}{2}$).
6. Displacement shift in search interval $S_{\max}$ (Default value = 2*W).

Output:

Scaled modified output sequence *Y*.

Begin

Initialization Phase

1. $D_S = D_A * a; T = W - D_S$.
2. k = 1. /*Set variable for number of input data frame */
3. $d_0 = 0$; /* Initialize shift value */
4. For i = 1 to W /* Initialize input data stream */
   $Y(i) = X(i)$,
   End For.
5. While($X(k D_A)) \neq$ NULL do /*Repeat until end of sequence*/

Analysis Phase

Let, $X^{d_k,k} = < X(kD_A + d_k + 1), X(kD_A + d_k + 2),....., X(kD_A + d_k + T) >$,

$Y^k =< Y(kD_S + 1), Y(kD_S + 2),....., Y(kD_S + T) >$.

Find $(d_k \in [0, S_{\max}]) \& (d_k \neq d_{k-1} + D_S - D_A)$ such that crosscorrelation coefficient (simplified, see eq. 3.3) $C'_{X^{d_k,k},Y^k}$ is maximum,

Synthesis Phase

/* Weigh and summate overlapping frames */

For j = 1 to T

$Y(k D_S + j) = (w * X(k D_A + d_k + j) + (1\text{-}w) * Y(k D_S + j))$,

End For.

Figure 3.2: Dynamic Overlay and Summation Algorithm.

```
        /* Iterate over remaining part of the frame */
        For j = T +1 to W
                Y(k D_S +j) = X(k D_A + d_k +j),
        End For.
        k = k+1.                        /* Increment frame number */
    End While.
End.
```

Figure 3.2 (Cont.)

Analysis phase: During the analysis phase windows are selected at an interval of D_A (analysis displacement) elements. In this implementation, the first T samples[3] in each of the analysis window overlay the last T samples in the output stream and the crosscorrelation calculated for these T elements The starting position of each analysis window is then allowed to shift (right, in the direction of the end of input data stream) by one element and crosscorrelation between the first T elements of this new (shifted) window and last T elements of the same window in the output data stream is calculated. This procedure (of shifting and calculating crosscorrelation) is repeated until, either number of shifts exceeds S_{max} or we reach the end of the data stream. The shift value (d_k) equal to $d_{k-1} + D_S - D_A$, where d_{k-1} is the shift determined for k-1th window, is ignored and not considered for determining maximum crosscorrelation since it would result in a *bias* in the construction of the output data stream (see Section 3.1.4). The shift value at which the crosscorrelation takes maximum value is chosen and the window of W samples occurring at this shift value of d_k from the current analysis position is selected for synthesis.

Synthesis phase: The window selected in the analysis phase is placed at the distance of D_S (synthesis displacement) from the start of the previous window in the output data stream. The portion of the input data stream that contributes to the overlap is weighed by a factor of w and correspondingly the elements of the output data stream

[3] Defined as accuracy overlap (see Definition 3.5).

contributing to the overlapping portion are weighed by a factor of (1-w) before they are summed. Overlapping and adding the windows at a fixed interval of D_S in the output data stream has two advantages. It can reduce discontinuities arising from different interframe intervals used during analysis and controls the growth rate of the output sequence to achieve constant data reduction. Note that since only last T samples of the output sequence are allowed to overlap with the input sequence, $D_S = W - T$.

We call the time-scaled sequence as the time emblem of the input data sequence. Correspondingly, we define an emblem operator $\sqsubseteq$ that converts the input data sequence to its time emblem for a given parametric set.

The subsequent discussion in this section deals with certain fine attributes and assumptions that are important to understand the operation of the algorithm.

3.1.1 Normalized Crosscorrelation

Crosscorrelation (see [10]) is a standard method of estimating the degree to which two series are correlated and is used to evaluate similarity between overlapping segments in the input and output sequences in our algorithm. The window to be added is shifted along the shift interval and a normalized crosscorrelation between overlapping elements evaluated at each position.

Consider two sequences p and q each of size T given as

$$p = < p_1, p_2, \ldots\ldots, p_T >,$$

$$q = < q_1, q_2, \ldots\ldots, q_T >.$$

It is shown, for example in [10], that an estimate $C_{p,q}$ of the unnormalized cross-correlation coefficient for p and q is provided by

$$C_{p,q} = \frac{1}{T} \sum_{t=1}^{T} (p_t - m_p)(q_t - m_q) \quad \ldots\ldots(3.1)$$

where p_t and q_t are t^{th}, $t \in [1, T]$, element and m_p and m_q are mean values of sequences p and q respectively. However, this unnormalized correlation product varies with the data points. Consequently, the highest correlation product may not

correspond to the most correlated element; which can give misleading indication of alignment.

To circumvent this, usually a normalized crosscorrelation is computed (see [10]) to provide a measure of relative correlation between two data segments in lieu of unnormalized crosscorrelation. It corrects the effects of increasing and decreasing amplitudes in the segments to be correlated by dividing the square root of power in the two data segments. This removes the amplitude bias that can arise from amplitude variations over several periods and restricts the function to the interval [-1, +1].

Consider the normalized crosscorrelation function given by

$$C_{p,q(normalized)} = \frac{\frac{1}{T}\sum_{t=1}^{T}(p_t - m_p)(q_t - m_q)}{\sqrt{\frac{1}{T}\sum_{t=1}^{T}(p_t - m_p)^2} * \sqrt{\frac{1}{T}\sum_{t=1}^{T}(q_t - m_q)^2}}$$

$$= \frac{\sum_{t=1}^{T}(p_t - m_p)(q_t - m_q)}{\sqrt{\sum_{t=1}^{T}(p_t - m_p)^2 * \sum_{t=1}^{T}(q_t - m_q)^2}} \quad \text{.....(3.2)}$$

By the Cauchy-Schwartz[4] inequality,

$$\left|C_{p,q(normalized)}\right| \le 1$$

with equality iff

$(p_t - m_p) = k(q_t - m_q) \;\; \forall t \in [1,T]$, for some scalar $k \neq 0$.

For our application, as depicted in the algorithm, we compute crosscorrelation between first T elements of the current input window under consideration and last T elements of output data sequence to determine the shift (d_k , k being the current window of interest). If sequence p and q in eq. (3.2) denote the contributing subsequences (of size T) from input and output data sequence

[4] If **x** and **y** are vectors in an inner product space, then $\left|\vec{x} \bullet \vec{y}\right| \le \left\|\vec{x}\right\|\left\|\vec{y}\right\|$.

respectively, then note that during the shift determination in the algorithm, the same elements of the output data sequence (given by Y^k in algorithm) are used for each crosscorrelation computation during shifts. In other words, quantity $\sum_{t=1}^{T}(q_t - m_q)^2$ remains constant during the shift determination (changing value of d_k) of the current (k^{th}) window. Since, only the maximum of the normalized crosscorrelation is desired for each shift determination, this constant can be removed with no effect on the location of the maximum, only its value and leading to a simplified expression given as follows.

$$C'_{p,q} = \frac{\sum_{t=1}^{T}(p_t - m_p)(q_t - m_q)}{\sqrt{\sum_{t=1}^{T}(p_t - m_p)^2}} \quad \text{.....(3.3)}$$

This expression for normalized crosscorrelation is used for our definition of the algorithm.

3.1.2 Window Length

The *DOLS* algorithm operates by manipulating windowed segments of the input stream to modify the time scale. The window length parameter sets the size of the smallest modifiable unit of data stream. An important observation is that these windows are treated as units of (independent) data streams and assumed to be relatively stationary in behavior. However, it should also represent a size such that overlapping segments realistically illustrate the redundancy in the data.

The window length must thus be chosen carefully to avoid violating this assumption, that is, short enough to contain the fact that signals do not change *drastically* over the duration of the window and large enough so that sufficient overlapping segments are available for correlation realization. Ideally, no window should contain transitory behavior.

Since the window length must remain fixed throughout the algorithm, it is difficult to eliminate transitory portions from all windows. Such a scheme is not

easily implemented nor is it proposed. However, window size can be well dictated by the application domain of interest.

For example, in stock prices and exchange rate databases, analysts are interested in monthly or longer patterns because shorter patterns are susceptible to noise [15]. In music databases a minimum query length can specify a recognizable segment of musical score, for example, designating window size such that it contains a pitch period[5].

To realize time scale reduction, the synthesis displacement D_S should be less than window size W. This would also ensure that that there are no gaps in the output signal. Also note that W is an a independent factor.

3.1.3 Recursive Addition Updating

If overlay weight w is not specified, a type of time-order dependent weighting is adopted, which adds the overlapping portions with the existing signal, and immediately divides by two ($w = \frac{1}{2}$, here). Subsequent overlapping windows are then added to the previously constructed stream and the sum divided by two. Averaging the windows at each frame incorporates a time order dependent weighting when multiple windows (≥ 2) overlap. The first window is weighted by $\frac{1}{2^n}$, where n is the total number of overlapping windows contributing to the given section of the window stream.

3.1.4 Bias due to Re-occurrence of Input Data

In *DOLS* implementation, during the determination of shift $d_k \in [0, S_{\max}]$ (in analysis phase), the value $(d_k = d_{k-1} + D_S - D_A)$ is ignored. This is done because it is observed that certain shift value of current window allows same trailing elements of previous window to reappear in as leading elements of a new (current) window selected, thus giving a correlation match during analysis phase. This behavior is

[5] Pitch is an attribute of every musical tone; the fundamental, or first harmonic, of any tone in the musical scale is perceived as its pitch.

undesirable since redundancy recognized here is due to same elements (not repeating behavior). A discussion on this follows.

Note that overlaying points in the construction of the output sequence depend on values of D_S and *T*. Three cases arise as follows depending on the relative values of D_S and *T*.

Case 1: $D_S < T$.

Figure 3.3 depicts this case.

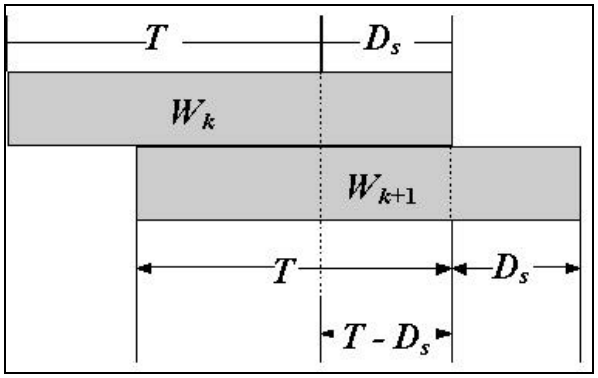


Figure 3.3: Overlay when $D_S < T$.

The starting position, in the input sequence, of the portion of the output sequence contributing *raw* (unchanged) elements to the overlay is

$(k)D_A + d_k + T + 1$, where d_k is the shift of the k^{th} window.

The portion of the input sequence overlaying *raw* elements of output sequence start from

$(k+1)D_A + d_{k+1} + (T - D_S) + 1$, where d_{k+1} is the shift of the $k+1^{\text{th}}$ window.

If same elements reappear in the selected input data window then,

$$(k)D_A + d_k + T + 1 = (k+1)D_A + d_{k+1} + (T - D_S) + 1 \Rightarrow$$

$$kD_A + d_k + T + 1 = kD_A + D_A + d_{k+1} + T - D_S + 1 \Rightarrow$$

$$d_k + (D_S - D_A) = d_{k+1}.$$

Case 2: $D_S > T$

Figure 3.4 depicts the case. The starting position, in the input sequence, of the portion of the output sequence contributing *raw* elements to the overlay is

$(k)D_A + d_k + T + (D_S - T) + 1$.

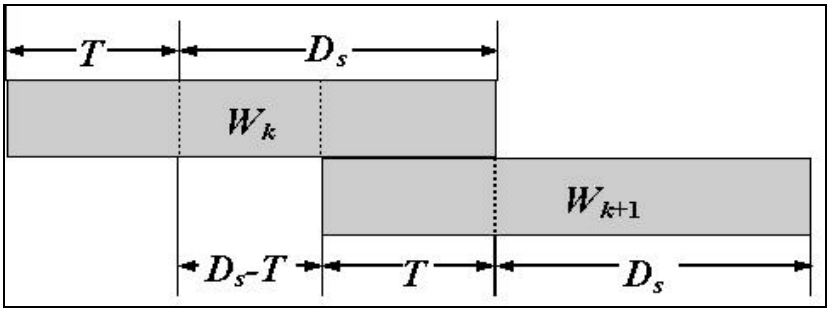


Figure 3.4: Overlay when $D_S > T$.

Portion of the input sequence contributing *raw* elements to the overlay start from

$(k+1)D_A + d_{k+1} + 1$.

If same elements reappear in the selected input data window then,

$(k)D_A + d_k + T + (D_S - T) + 1 = (k+1)D_A + d_{k+1} + 1 \Rightarrow$

$kD_A + d_k + D_S + 1 = kD_A + D_A + d_{k+1} + 1 \Rightarrow$

$d_k + (D_S - D_A) = d_{k+1}$.

Case 3: $D_S = T$

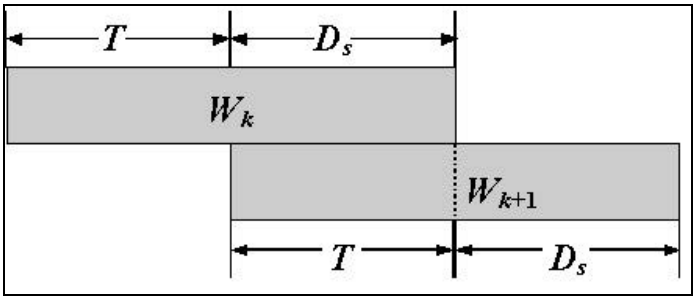


Figure 3.5: Overlay when $D_S = T$.

Figure 3.5 depicts the case. The starting position, in the input sequence, of the portion of the output sequence contributing *raw* elements to the overlay is

$(k)D_A + d_k + T + 1$.

Portion of the input sequence contributing *raw* elements to the overlay start from

$(k+1)D_A + d_{k+1} + 1$.

If same elements reappear in the selected input data window then,

$$(k)D_A + d_k + T + 1 = (k+1)D_A + d_{k+1} + 1 \Rightarrow$$

$$kD_A + d_k + T + 1 = kD_A + D_A + d_{k+1} + 1 \Rightarrow$$

$$d_k + (T - D_A) = d_{k+1} \Rightarrow$$

$$d_k + (D_S - D_A) = d_{k+1} \quad \text{.....(3.4)}$$

Thus, not allowing the shift of the current (k+1th) window to take a value of $(d_k + (D_S - D_A))$ will ensure no bias occurs due to re-occurrence of a portion of input data stream in the synthesized output data stream.

3.1.5 Global versus Local Maximization of Crosscorrelation

In *DOLS*'s analysis phase, the shift of a current window is determined without regard for its effect on subsequent window alignment. Reducing the crosscorrelation of a current window, i.e. $\max C'_{X^{d_k,k},Y^k}$, may increase the crosscorrelation of several subsequent windows, $\max C'_{X^{d_{k+1},k+1},Y^{k+1}}$, $\max C'_{X^{d_{k+2},k+2},Y^{k+2}}$ and $\max C'_{X^{d_{k+3},k+3},Y^{k+3}}$, increasing the global correlation. Here, subscript pairs are sets of input and output subsequence involved in crosscorrelation computation where d_k, d_{k+1}, d_{k+2} and d_{k+3} are corresponding shift values for input window numbered k, k+1, k+2 and k+3 respectively. Obviously, maximizing the global correlation would require non-causal processing of the signal, which is not desirable if we want to operate the algorithm in online[6] mode.

Let us consider the region $[\, kD_A + j,\ kD_A + j + S_{\max}\,]$ for $j \in [1, T]$, where k is the current input window of interest in the analysis phase. If this is perceived as the *feasible region* (possible positions of the analysis window) and the normalized crosscorrelation function is our objective function[7] of interest, then the shifting procedure in our analysis phase aligns the window to a local crosscorrelation

[6] Real time modification of data stream without any estimation of future windows.
[7] Function to evaluate the goodness of the solution (here, alignment of analysis window).

maximum position in the feasible region. These local alignments do not necessarily guarantee a globally optimal solution.

3.1.6 Actual Time Scale Modification Performed

The actual time scale modification performed by *DOLS* algorithm is slightly different from desired scale modification. Let l_i is the length of the input stream and l_o is the actual time scaled modified length of the output data stream. The number of frames extracted from the input sequence is given by

Number of frames = $1+\frac{(l_i - W)}{D_A}$, where the symbols have their usual meanings.

The algorithm begins by copying a first frame of size W to the output sequence (intitialization) and then placing each new frame at an interval of D_S in the synthesis phase. The synthesis phase is responsible for growing the signal by an amount of W-T (= D_S). Hence,

$$l_o = W + (\frac{l_i - W}{D_A})(D_S) \quad \text{.....(3.5)}$$

Since $\frac{D_S}{D_A}$ is defined as a , from above we get,

$$l_o = W + a(l_i - W) = a l_i + (1-a)W \quad \text{.....(3.6)}$$

Thus, for $a < 1$(compression) the output data stream has $(1-a)W$ extra data points than desired. These extra data points would never exceed window size W. To desire exact time-scale compression, these last $(1-a)W$ data points in the output sequence can, for example, plainly be ignored else averaged out with the previous same number of points in the output data stream.

3.1.7 Computational Complexity

The computational requirements of *DOLS* are determined by the choice of parameter values. The input signal is windowed every D_A points. The number of windowed segments, or frames, per unit time is therefore proportional to $\frac{1}{D_A}$. For each frame, a proper shift value must be determined such that the crosscorrelation

with the previous frames will be maximized. For each possible shift value in the interval [0, S_{max}] (except, when shift value = previous shift+ $D_S - D_A$), a normalized crosscorrelation must be evaluated to determine which shift gives proper correlation. If *CPC* (crosscorrelation product computation) refers to the crosscorrelation computation for every overlap, then the required computations can be expressed as proportional to:

$$\propto \left[\frac{1}{a\ D_A}\right]\left\{\ [S_{\max}\ ]\ [T]\ [CPC]\ \right\}$$

$$\propto \left[\frac{1}{a\ D_A}\right]\left\{\ [S_{\max}\ ]\ [W - D_S\ ]\ [CPC]\ \right\}, \text{……..……..…….(3.7)}$$

where the symbols have their usual meanings.

Worst case complexity occurs when we follow the choice of window size, analysis displacement and displacement shift in search interval as realizable factors of the sequence length, such that,

$$\text{W}=\frac{N}{c_1}, D_A = \frac{W}{c_2};\ S_{\max} = c_3 * W\ ; (c_1,\ c_2,\ c_3 \text{ are constants});$$

The computational complexity (worst case) for these choice of parameters is $O(N^2)$.

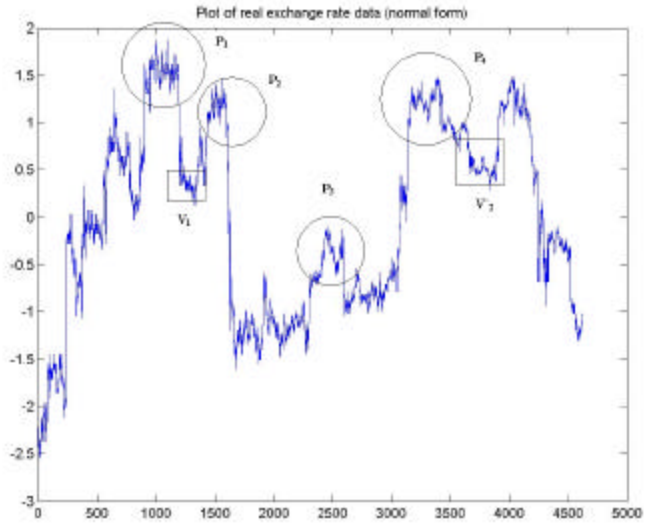
3.1.8 An Example

Consider a data sequence of closing price of real exchange rate data shown in Figure 3.6(a). We assume that sequences are normalized although it is not a mandatory requirement for our algorithm[8]. We time-scale this sequence by $a = 0.25$ and $\{W=48\ (=\sqrt{N}), D_A = 24, D_S = 6, T = 42\}$. The resultant plot is shown in Figure 3.6(b).

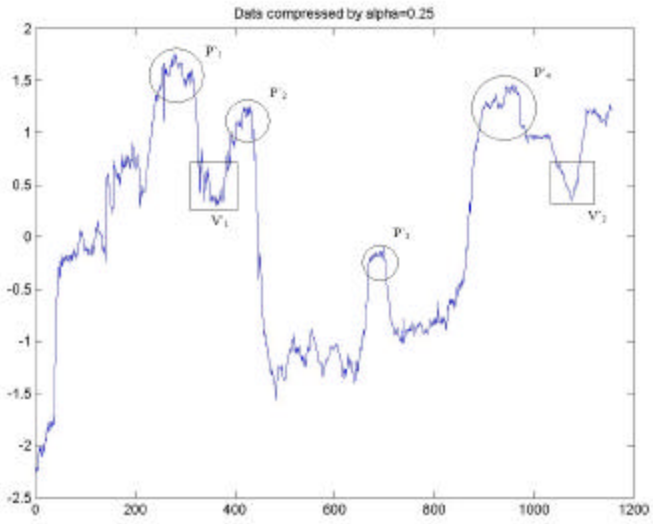
Note that in Figure 3.6(b) the short-term peaks (shown in circles denoted by P_1, P_2, P_3 and P_4) are approximately averaged and replaced by their fewer occurrences within an interval (denoted by $P`_1$ thru $P`_4$ in their corresponding *DOLS* scaled form in Figure 3.6(b)). Similar behavior is observed for short-term valleys

[8] Normal forms have mean of 0 and a standard deviation of 1. We postpone further discussion on the advantages of this to Section 3.2.2.

(shown in boxes denoted by V_1 and V_2, corresponding *DOLS* modified occurrence indicated by $V`_1$ and $V`_2$). The algorithm attempts to replace short intervals redundancies in the data by fewer than original occurrence without compromising their long-term behavior.



(a)



(b)

Figure 3.6: (a) Plot of normal form of exchange rate data between Swiss Franc-USD for 4624 days (Source: ftp://ftp.santafe.edu) (b) Plot for time scale modified stream (a = 0.25).

3.2 Whole Matching Using *DOLS*

Given a database of reference time sequences and a query sequence, an evaluation approach is to iterate over the sequences in the database and for each sequence compute the distance from the given query sequence.

The number of sequences in the database, length of each sequence and the notion of similarity being used determine the complexity of such an operation. Since individual sequences in the database can be large, approximate matching can be computationally very expensive.

Moreover, the sizes of these sequences might be different. We present a technique to reduce the comparison cost by using short representative time emblems to perform matching instead of real sequences. This procedure has two benefits:

1. This can allow us to compare sequences of unequal sizes (whole matching) by using different transformations for each sequence.
2. Due to their small size, the emblem matching can take place orders of magnitude faster than full scan and match on the entire database.

Before we proceed further in our discussion on whole matching, it becomes necessary to present a notion of similarity that would serve as a measure of similitude between sequences.

3.2.1 Notion of Similarity

The meaning of similarity depends on the specific application and also on the objective of the query. For example, some researchers (such as Goodman in [23]) believe that similarity is a three place predicate; that is, it is ill defined to say *a* is similar to *b*; but it is only meaningful if *a* is similar to *b* with respect to[9] *c*. Therefore it is important to define similarity for a specific query and query space of interest.

There are generally two general models to describe the notion of similarity. One model is called multidimensional scaling (see [52]). Multidimensional scaling (MDS) describes a family of techniques for the analysis of proximity data on a set of

[9] For example, it might be ill defined to just say that two given 3-dimensional objects are similar to each other. These objects can either have similar shape (two spherical objects) or can occupy the same volume (one sphere and other a cube). So to say that these two objects are similar, it is more meaningful to say that they are similar with respect to their shape or volume (a 3-place predicate).

stimuli to reveal the hidden structure underlying the data. The proximity data can come from similarity judgments, identification confusion matrices, grouping data, same-different errors or any other measure of pairwise similarity. The main assumption in MDS is that stimuli can be described by values along a set of dimensions that places these stimuli as points in a multidimensional space and that the similarity between stimuli is inversely related to the distances of the corresponding points in the multidimensional space. The Minkowski distance metric provides a general way to specify distance in a multidimensional space. An unweighted[10] form of Minkowski distance in a k-dimensional space is given by

$$D(O_a, O_b) = \left(\sum_{n=1}^{k} |y_{an} - y_{bn}|^r\right)^{1/r} \quad \text{(3.8)}$$

where y_{an} and y_{bn} is the value of objects O_a and O_b respectively, in n^{th} dimension $(1 \le n \le k)$ and r indicates the distance metric (r = 1 gives city-block distance, r = 2 gives Euclidean distance and $r = \infty$ gives Chebyshev distance). In this model, the notion of similarity is specified in terms of a distance function and some weights can be assigned to features. An alternative model defines similarity between two objects as a function of their common and distinctive features [54].

In our analysis of the whole matching problem we follow the first model for reasons outlined below. Specifically we use Euclidean distance (r = 2) given as follows.

$$D(O_a, O_b) = \left(\sum_{n=1}^{k} |y_{an} - y_{bn}|^2\right)^{1/2} \quad \text{(3.9)}$$

Euclidean distance is our choice of distance measure because:

1. Euclidean distance is the optimal distance measure estimation [54], if signals are corrupted by Gaussian, additive noise.

[10] Each dimension (n), here, is assumed to have the same attention weight. A corresponding weighted metric can be given as $D(O_a, O_b) = (\sum_{n=1}^{k} w_n |y_{an} - y_{bn}|^r)^{1/r}$ where w_n ($w_n \ge 0$, $\sum_{n=1}^{k} w_n = 1$) is the attention weight assigned to dimension n.

2. It can be used with any other type of similarity measure as long as this measure can be expressed as the Euclidean distance between feature vectors in some feature space.
3. In a recent survey of image[11] registration techniques, Brown [9] mentions that one of the typical similarity measures is crosscorrelation. This can be related to Euclidean distance (as presented in Lemma 3.1 below).
4. It readily satisfies the requirements for a distance to be a metric. Specifically, if the database objects are drawn from set *Objects* and let *distance* be a metric distance function for pairs of objects. That is, $distance: Objects \times Objects \to R_0^+$ then *distance* is a metric if it satisfies the following conditions $\forall O_1, O_2, O_3 \in Objects$.

   Identity Property: $distance(O_1, O_2) = 0 \Leftrightarrow O_1 = O_2$,

   Symmetry Property: $distance(O_1, O_2) = distance(O_2, O_1)$ and

   Triangle Inequality: $distance(O_1, O_3) \leq distance(O_1, O_2) + distance(O_2, O_3)$.

Euclidean distance obviously can only be computed for sequences of equal sizes. For two sequences it can be perceived as degree of dissimilarity between them. Having reasoned the choice of distance metric to measure similitude between sequences, we now develop reasoning behind using normal form of sequences for analysis rather their raw forms. This is the focus of our discussion in the following section.

3.2.2 Normal Form of Sequences

Given a time sequence *S* of real numbers of length *n*, the sequence data has a mean value of m_S and standard deviation[12] s_S given by,

[11] Images are considered as 2-dimensional signals.

[12] Note that the given formula of standard deviation is for a *population* (entire set) rather than for a *sample* set which is a subset drawn, for example, randomly from the population. The unbiased estimate of variance provided by the sample set normalizes by (*n*-1) rather than *n* in case of s_S here.

$$m_S = \frac{1}{n}\sum_{i=1}^{n} S_i \; ; \; s_S = \left(\frac{\sum_{i=1}^{n}(S_i - m_S)^2}{n}\right)^{1/2} \quad \text{..... (3.10)}$$

Definition 3.6: Normal Form:

A sequence *S* is in normal form if $m_S = 0$ and $s_S = 1$ for the sequence.

The following expression gives the data values for the normal form of a sequence.

$$NS_i = \frac{S_i - m_S}{s_S}, \; 1 \le i \le len(S) \quad \text{..... (3.11)}$$

where NS_i $(1 \le i \le length(S))$ is an element of normal form of *S*.

Lemma 3.1: If *R* and *Q* are two sequences of size *n* in their normal forms then $D^2(R,Q) = 2n(1 - C_{R,Q})$ where $C_{R,Q}$ is normalized crosscorrelation between *R* and *Q*.

Proof:

$$D(R,Q) = \left(\sum_{i=1}^{n}(R_i - Q_i)^2\right)^{1/2} = \left(\left(\sum_{i=1}^{n} R_i^2 + \sum_{i=1}^{n} Q_i^2 - 2\sum_{i=1}^{n} R_i Q_i\right)^{1/2}\right) \quad \text{.....(3.12)}$$

Since R and Q are in their normal forms,

$$s_R = 1, s_Q = 1; m_R = 0, m_Q = 0; \quad \text{.....(3.13)}$$

Since $s_R = \frac{1}{\sqrt{n}}\left(\sum_{i=1}^{n}(R_i - m_R)^2\right)^{1/2} = 1$, squaring and substituting from (3.13) we get

$$n = \sum_{i=1}^{n} R_i^2 \quad \text{.....(3.14)}$$

Similarly, $n = \sum_{i=1}^{n} Q_i^2$ .....(3.15)

From (3.12) we get,

$$D^2(R,Q) = n + n - 2\sum_{i=1}^{n} R_i Q_i \quad \text{.....(3.16)}$$

Normalized crosscorrelation for normal form of sequence *R* and *Q* takes the following form from eq. (3.2).

$$C_{R,Q} = \frac{\sum_{i=1}^{n}(R_i Q_i)}{\sqrt{\sum_{i=1}^{n}(R_i)^2 * \sum_{i=1}^{n}(Q_i)^2}}.$$

From (3.14) and (3.15)), $C_{R,Q} = \frac{\sum_{i=1}^{n}(R_i Q_i)}{\sqrt{n*n}} \Rightarrow nC_{R,Q} = \sum_{i=1}^{n}(R_i Q_i)$.

Substituting above in equation (3.16),

$D^2(R,Q) = n + n - 2nC_{R,Q} = 2n(1- C_{R,Q})$.

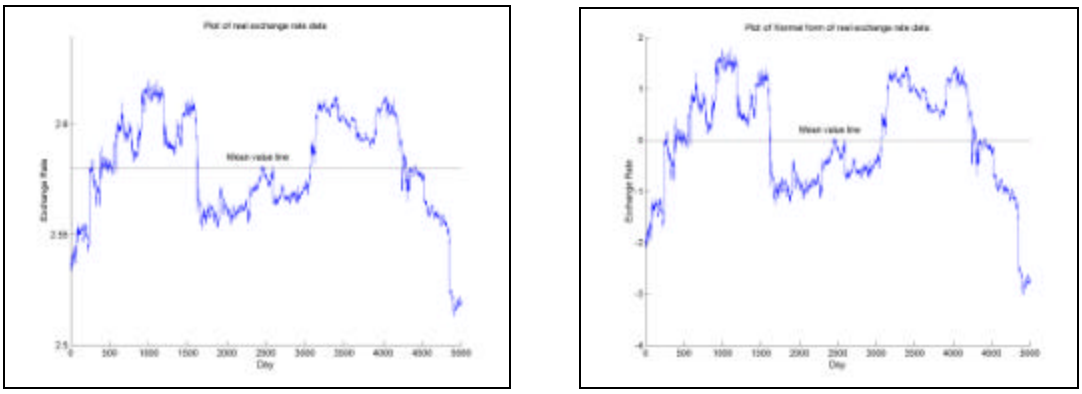
Although the algorithms presented in this dissertation do not require it, we assume time sequences are normalized for the following reasons.

1. It improves efficiency (as noted by Goldin *et al.* in [22]).
2. The Euclidean distance between normal sequences is related to their normalized crosscorrelation (as presented in Lemma 3.1).
3. Transformation to a normal form is a shape preserving affine (combination of scaling and shifting) transformation[13].

The property presented in point 2 above is particularly useful for defining similarity queries in terms of crosscorrelation, rather than Euclidean distance for a database of sequences of equal sizes and same sized query sequence. Since Euclidean distance between two sequences can range from zero to infinity, it is usually difficult to specify a threshold for this distance. Instead, we can specify a cross-correlation, which is between 0 and 1, and use the above equation to find corresponding threshold for the Euclidean distance.

We also define a normalization operator $\mathbb{N}$ that converts a given sequence to its corresponding normal form. Figure 3.7 illustrates a time sequence of a real stock market data and its corresponding normal form. Note that location of the mean (indicated by day location corresponding to the intersection of mean value line and the data plot) remains the same after transformation to normal form.

[13] Also known as *similarity transformations* in the well-established field of transformation geometry.



(a) (b)

Figure 3.7: (a) *R*: Exchange rate data between Swiss Franc and USD (Source: ftp://ftp.santafe.edu) (b) $\mathcal{N}(R)$: Normal form of Exchange rate data.

3.2.3 Whole Matching

In Section 3.1 we introduced a dynamic overlay and summation technique for reducing the time scale of a data sequence. In this section we employ it for the whole matching problem introduced in Section 2.4.

The idea is to converge each of the sequences in the database *DB* and query sequence *Q* to a common size so that Euclidean distance can be computed between them. We also assume that the users desire a maximum dimensionality reduction *K*. In other words, at least one of the data sequences in *DB* and *Q* must be time compressed by a factor *K*. This primitive is required for guided time-scale reduction when the user is interested to control the size of the output sequence. If *L* is the common size of the sequences obtained after time scale modification, then *L* is given by

$$L = \min[\min[\ len(DB.R_i)\forall i; 1 \le i \le size(DB), len(Q)],$$

$$\frac{\max[\ len(DB.R_i)\forall i; 1 \le i \le size(DB), len(Q)]}{K}].$$

An algorithm to answer range query of threshold $\in$ is shown in Figure 3.8. In *whole_match*, first the common length *L* of the sequence in the database is computed. Then the database is processed such that each sequence is *DOLS*-scaled to size *L* using a window size, which is approximately equal the square root of the length of sequence. Correspondingly, the query sequence is processed to achieve

Algorithm *whole_match*

Input: 1. Database *DB*
2. Query sequence *Q*
3. Maximum dimensionality reduction *K*
4. Range threshold $\in$

Output:

Sequences in *DB* that match *Q*

Begin

Initialization

L = *high_value*

For *i* = 1 to size(*DB*) do

$$L' = \min[\min[\ len(DB.R_i), len(Q)], \frac{\max[\ len(DB.R_i), len(Q)]}{K}],$$

$$L = \min[\ L', L],$$

End For.

Database Processing

For *i* = 1 to size(*DB*) do

$$a_i = \frac{len(DB.R_i)}{L},$$

$$W_i = \sqrt{len(DB.R_i)},$$

if $a_i \neq 1$

$$\Box(R_i) = DOLS(DB.R_i, W_i, a_i),$$

else

$$\Box(R_i) = R_i,$$

End For.

Query Sequence Processing

$$a_Q = \frac{len(Q)}{L}.$$

$$W_Q = \sqrt{len(Q)}.$$

$$\Box(R_Q) = DOLS(Q, W_Q, a_Q).$$

Main Loop

For *i* = 1 to size(*DB*) do

Dist = $D(\Box(R_i), \Box(R_Q))$,

If Dist $\leq \in$

Output $DB.R_i$,

End For.

End

Figure 3.8: Whole matching algorithm for range query of threshold $\in$.

the common size[14]. The resultant is a set of reference emblems (from *DB*) each of length *L* and a query emblem. The distances are then computed for each reference emblem in the database to report an answer set that has distance from query emblem, less than a threshold. The algorithm *whole_match* returns all sequences in database *DB* that match *Q* approximately.

3.3 Summary

In this chapter we presented an approach for time-scale modification called dynamic overlay and summation technique (*DOLS*). Some merits of using Euclidean distance as our choice of similarity metric for whole matching were discussed. We also presented the advantages of using normal form of sequences for similitude search. Subsequently we employed the proposed *DOLS* technique for the whole matching problem to search for similar sequences in the database of unequal sequence sizes that match a query sequence approximately.

[14] As indicated in Section 3.1.6 a small correction is applied to the *DOLS* extracted sequences to achieve the desired time scale compression.

Chapter 4

Subsequence Matching

The problem on which we will be focusing in this part of the dissertation is the design of an efficient searching method that will locate pairs of subsequences within a query sequence and a reference sequence that match approximately. We will also discuss several interesting approaches within this problem that can affect the quality of results obtained by the proposed framework.

The problem of *similar-pairs* query can be stated as follows. Given a reference sequence R of size L_R and a query sequence Q of size L_Q, find subsequences $[R_i \dots R_n],\ 1 \le i \le n \le L_R$ in R that match $[Q_j \dots Q_m],\ 1 \le j \le m \le L_Q$ in Q approximately. The *distance* between appropriately transformed subsequences measures the approximate match between them. The location and degree of closeness of similar objects in reference and query sequences are reported.

Given a reference sequence R of size L_R, where each data point R_i, $1 \le i \le L_R$ represents a data entry at an instant of time t_i, we consider subsequences $[R_i \dots R_n],\ 1 \le i \le n \le L_R$ as objects in time-space. These objects are parameterized by O_{in}^R where subscripts indicate the indices of start and end time of the object in the reference sequence. For *similar-pairs* query, objects in the query sequence (subsequences $[Q_j \dots Q_m],\ 1 \le j \le m \le L_R$) parameterized by O_{jm}^Q are *derived* from query sequence. These query objects are selectively compared with each object O_{in}^R for similarity and their degree of match reported. The reference objects are arranged in an index called *FG-index* to achieve faster retrieval of similar objects.

In the next section we present some preliminaries that will facilitate the understanding of our proposed approach.

4.1 Preliminaries

Searching for similar subsequences demands a notion of similarity, which indicates degree of closeness of these subsequences. This remains the topic of discussion in the next section.

4.1.1 Notion of Similarity

Consider objects A and B each of size M X N.

The measure of distance of object A from the object B is given by

$$D(A,B)=\left|\left[\sum_{j=1}^{N}\sum_{i=1}^{M}[B(i,j)-A(i,j)]^2\right]^{\frac{1}{2}}\right| \quad \text{.....(4.1)}$$

This is the discrete version of the distance formula

$D(A,B)=[\int\int_{Domain}[B(x,y)-A(x,y)]^2\,dxdy]^{\frac{1}{2}}$ where, the objects A and B are considered to vary continuously and *Domain* specifies the domain of definition of the considered objects.

$$D^2(A,B)=\sum_{j=1}^{N}\sum_{i=1}^{M}B^2(i,j)+\sum_{j=1}^{N}\sum_{i=1}^{M}A^2(i,j)-2\sum_{j=1}^{N}\sum_{i=1}^{M}B(i,j)A(i,j)$$

If object B is scalar shifted by (m,n) w.r.t object A in x-y Cartesian space, then the distance is given by

$$D^2(A,B_{m,n})=\sum_{j=1}^{N}\sum_{i=1}^{M}B^2(i+m,j+n)+\sum_{j=1}^{N}\sum_{i=1}^{M}A^2(i,j)$$

$$-2\sum_{j=1}^{N}\sum_{i=1}^{M}B(i+m,j+n)A(i,j) \quad \text{.....(4.2)}$$

The first and second term on the r.h.s. of above equation are the square of the energy object B and A respectively and the term $\sum_{j=1}^{N}\sum_{i=1}^{M}B(i+m,j+n)A(i,j)$ divided by product $(M*N)$ would give an unnormalized estimate of the correlation between objects B and A.

4.1.2 Discrete Fourier Transform

In this section we present a brief overview of the Discrete Fourier Transform (DFT). Detailed information on DFT and its properties can be found in any signal-processing textbook (for example, see [7]). The significance of DFT is that there exists a fast algorithm that can calculate Fourier coefficients in $O(N \log N)$ time.

For a continuous function of one variable $y(t)$, the Fourier Transform (see [7]) $Y(f)$ will be defined as:

$$Y(f) = \int_{-\infty}^{+\infty} y(t)\ e^{-j2pft} dt \quad \text{.....(4.3)}$$

and the inverse transform as:

$$y(t) = \int_{-\infty}^{+\infty} Y(f)\ e^{j2pft} df \quad \text{.....(4.4)}$$

where $j = \sqrt{-1}$, and e is the natural exponent.

$$e^{jf} = \cos(f) + j\sin(f)$$

Consider a complex series $x(k)$ with N samples of the form

$x_o, x_1, x_2, x_3, \ldots, x_{N-1}$, where x is a complex number ($x_i = x_{real} + j\, x_{imag}$).

Further, assume that that the series outside the range (0, N-1) is an extended N-periodic series, that is, $x_k = x_{k+N}\ \forall\, k \geq 0$. The Fourier Transform of this series will be denoted by $\mathcal{F}(x(k)) = X(k)$, and will also have N samples. The forward transform will be defined as:

$$X(n) = \frac{1}{N} \sum_{k=0}^{N-1} x(k)\ e^{-jk2pn/N} \text{ for } n = 0...N-1 \quad \text{.....(4.5)}$$

The Inverse transform[1] will be defined as

$$x(n) = \sum_{k=0}^{N-1} X(k)\ e^{jk2pn/N} \text{ for } n = 0...N-1 \quad \text{.....(4.6)}$$

[1] There is a minor discrepancy among authors on the use of $1/N$ in equation 4.5 . Some authors (for example in [6]) use $1/\sqrt{N}$ in both equations 4.5 and 4.6 place of $1/N$ in eqn. 4.5 and 1 in eqn. 4.6. We follow the definition in 4.5 and 4.6.

Although the functions here are described as complex series, setting the imaginary part to 0 can represent real valued series. In general, the transform into the frequency domain will be a complex valued function, with magnitude: $\|X(n)\| = (x_{real} * x_{real} + x_{imag} * x_{imag})^{0.5}$.

Figure 4.1 shows a time series of length T and its values in the frequency domain.

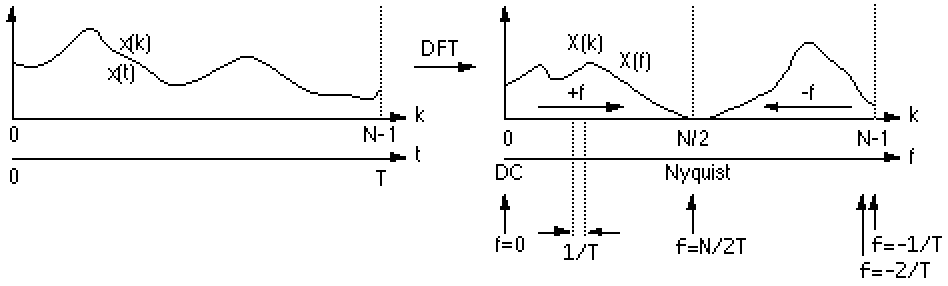


Figure 4.1: Time series of length T, N values and its corresponding discrete Fourier transform representation.

With respect to Figure 4.1, the following points are in order.

1. The first sample $X(0)$ of the transformed series is called the DC component, more commonly known as the average of the input series.

2. The DFT of a real series (that is, imaginary part of $x(k) = 0$) results in a symmetric series about the Nyquist frequency[2]. The negative frequency samples are also the inverse of the positive frequency samples.

•

There are important properties of Fourier transform such as linear, scaling, shifting and modulation relationships between time and frequency domains. The reader is referred to [6] for detailed discussion on these properties.

An important observation is the Rayleigh energy theorem also known as Parseval's theorem given as follows.

Theorem 4.1: Parseval's Theorem

[2] The highest positive (or negative) frequency sample is called the Nyquist frequency. This is the highest frequency component that should exist in the input series for the DFT to yield "uncorrupted" results. More specifically, if there are no frequencies above Nyquist frequency, the original signal can be *exactly* reconstructed from the samples.

Let $X(k)$ be a discrete Fourier transform of $x(n)$. Then the following holds true.

$$\sum_{n=0}^{N-1}\left|x(n)\right|^2 = \frac{1}{N}\sum_{n=0}^{N-1}\left|X(n)\right|^2 .$$

Proof: See [7].

Parseval's theorem is a special case of the Power Theorem, which essentially says that for a given signal $x(n)$, the energy of the signal remains the same after the Discrete Fourier Transformation. Using this and the linearity property of DFT, it is easy to show that the Euclidean distance between two signals in the time domain is the same as their distance in the frequency domain, that is,

$$D^2\left(\vec{x},\vec{y}\right) = E(\vec{x}-\vec{y}) = E(\vec{X}-\vec{Y}) = D^2(\vec{X},\vec{Y}) \quad \text{.....(4.7)}$$

This property is important because it essentially dictates that the Fourier transform is a Euclidean distance preserving transformation.

4.1.3 Normal Form of Sequence

Although the algorithm presented in this part of the thesis does not require it, we assume that windowed subsequences in the reference sequence are normalized for the reasons outlined in Section 3.2.2. Following lemma holds true for normal form of sequences.

Lemma 4.1: The DC component of the discrete Fourier Transform for the normal form of a sequence is zero.

Proof:

Let $x(t)$ be a given real valued time sequence of length N. Then the discrete Fourier transform of $x(t)$ is given by,

$$\Box(x(t)) = X(n) = \frac{1}{N}\sum_{k=0}^{N-1} x(k)\ e^{-jk2pn/N} \text{ for } n = 0,...,N-1 .$$

The DC component is the first coefficient given by $n = 0$. Hence, from the above equation we have

$$X(0) = \frac{1}{N}\sum_{k=0}^{N-1} x(k)\ \ e^{0} = \frac{1}{N}\sum_{k=0}^{N-1} x(k) = m_x$$ where m_x is the mean of elements of sequence $x(t)$.

If sequence $x(t)$ is in normal form then its mean is zero , that is, $m_x = 0 \Rightarrow X(0) = 0$.

4.2 Outline of Proposed Approach

In this section we discuss the outline of our approach to address *similar-pairs* query problem presented in Section 2.4. Without loss of generality, we assume that the minimum length of a query sequence[3] is w ($1 \leq w << \text{length}(R_i)$). Some values of w are presented in experimental results, but it certainly depends on the application of interest. For example, in stock prices and exchange rate databases, analysts are interested in longer or monthly patterns because shorter patterns are susceptible to noise [15]. In music databases a minimum query length can specify a recognizable segment of musical score (such as a pitch period).

Our approach relies on the idea of identifying subsequences within the reference and query sequence that are relatively stationary in harmonic behavior. This means that iterating within the identified subsequence does not result in substantial change in harmonic components of the subsequence. We start with segmenting the input sequence in windows of a specified size. These windows are represented as points in multidimensional feature space by their first few Fourier coefficients (for reasons outlined in Section 4.3). These points are grouped together to identify subsequences within the reference sequence. The index is built for these subsequences using the attribute vectors of their minimum enclosing rectangle (MER) approximations. The search on the index is performed as follows. All possible MERs are extracted from query sequence using techniques described above. The attribute vectors of these MERs are calculated and compared with those in the index to report possible match.

[3] We do not eliminate the ability to answer queries for size less than w. For query sequences for size less than w, we can always resort to sequential scanning.

The step-wise outline of our approach follows.

1. We use a sliding window of size w and place it at every possible offset in the reference sequence R. Each window is recognized as w_j where subscript j refers to the position of the window in the sequence.
2. For each w_j in step 1 we take the discrete Fourier transform $\digamma$ of the normalized subsequence enclosed in each window.
3. For each transform in step 2, we select first N_c (cut-off frequency) coefficients for reasons outlined in Section 4.3. Thus a window can be represented as a point in $\Re^{2N_c}$ feature space[4] where each N_c dimension represents a Fourier harmonic. These windows are time ordered and connected. We call this sequence in feature space as *feature trail*.
4. We group consecutive *points* in the feature trail where the *distance* between any two points (within the group) is not greater than a threshold. Note that each of these groups may not contain the same number of feature points. Suppose we formed M groups from all sequences.
5. Build a tree (called *FG*-tree) by recursively merging all 2 consecutive groups in the current level of the *FG*-tree. The height of the tree is M (detailed in Section 4.5) and each higher level contains one group less then the next lower level. The root group bounds all points in feature space. Suppose we have K groups in the *FG*-tree.
6. Each of the subsequences represented by each of K groups formed in step 5 is enclosed in a rectangle of minimum area.
7. These groups are then *indexed* based on their *attribute vectors* defining their corresponding MERs.
8. *Preprocess* the query sequence and search the index for similar subsequences.

Having introduced a brief resume of our approach for *similar-pairs* query we are now ready to detail the idea. We begin with our discussion on piecewise Fourier

[4] Remember, Fourier transform of real sequence is a complex number.

transform, subsequently employed for segregating windows based on their harmonic behavior, in the next section.

4.3 Piecewise Fourier Transform

Given a reference sequence we divide the sequence into every possible subsequence (pieces) of size *w* and compute their normal forms. For this, we slide a window of size *w* over the sequence at increasing offset. An algorithm to obtain all subsequences of size *w* from a sequence *R* is given in figure 4.2(*a*). Given a reference sequence *R* of length L_R and a window of size *w*, the number of possible subsequences of size *w* from *R* is (L_R -*w*+1). Clearly, the running time of the algorithm *extract_subsequence* is $O(L_R)$. Figure 4.2(*b*) demonstrates the process of extraction of subsequences. For each window w_i from *extract_subsequence* we compute its normal form and extract features using discrete Fourier transform.

The following lemma holds true for real-valued time-sequences.

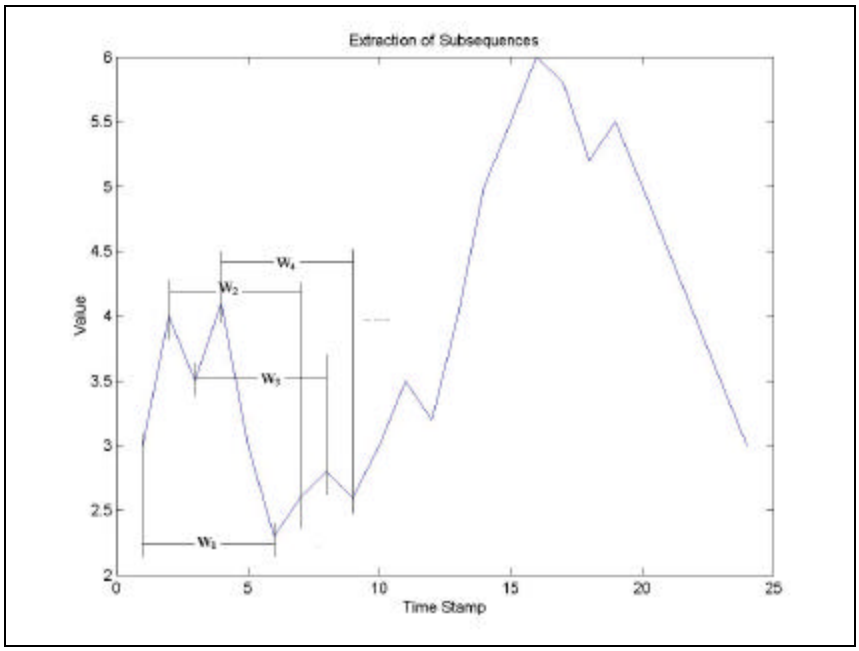
Lemma 4.2: The DFT coefficients of a real-valued sequence of duration n satisfy $X_{n-f} = X_f^*$ for f = 1,…,n-1 where the asterix denotes complex conjugation[5].

Proof: See [6].

```
Algorithm extract_subsequence(R,w)
Begin
        num = length(R) - w +1.
        For j = 1 to num do
                wj = R[j..j+w-1],
                Output wj,
        End For.
End
```

(a)

Figure 4.2: (a) Algorithm *Extract_Subsequence* to extract all subsequences of size *w* (=5) from a sequence (b) Extraction of subsequences from a reference sequence.



(b)

Figure 4.2 (Cont.)

Lemma 4.2 essentially implies that Fourier coefficients of a real-valued time sequence are symmetric around the center. In other words, the first $N/2$ coefficients contain all information of the signal and are sufficient to completely represent a signal.

We have our own reasons to believe (also in [2]) that most sequences of practical interest (such as stock market data, exchange rates, daily temperatures etc.) have a skewed energy spectrum. Figure 4.3 shows a log-plot of amplitude spectrum of real exchange rate data. Note the skew decay in amplitude with increasing frequency.

The majority of the *interesting* sequences[6] fall in the category of *colored noise* and have energy spectrum of the form $O(F^{-b})$, where F indicates the frequency of Fourier spectrum. Specifically,

[6] Skewed distribution of points in feature space should be true to almost any real set of sequences measuring a phenomenon, since sequence content representations usually include correlated elements.

- For b=1 in $O(F^{-b})$, the signal is classified as pink noise which according to Birkhoff (see [49]) represents signals like musical scores.
- For b=2 (with amplitude spectrum $O(F^{-1})$), the signals are classified as brown noise (random walk or Brownian walk) which has been shown to model stock prices and exchange rates (see [36]).

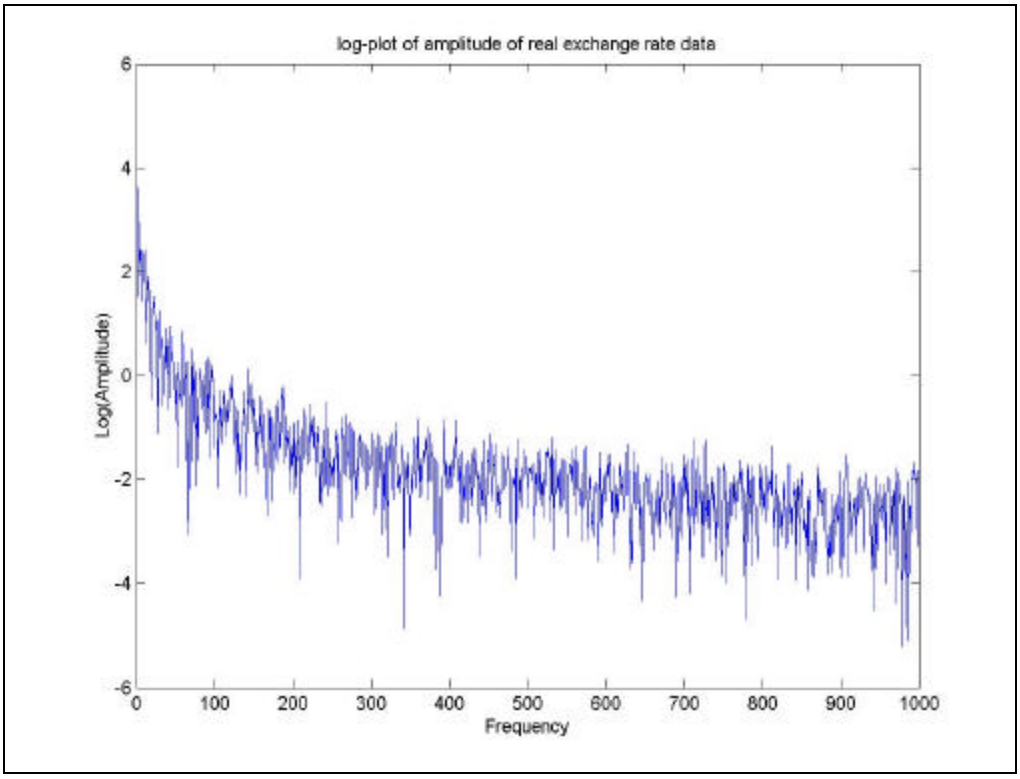


Figure 4.3: Amplitude log-plot of real exchange rate data versus increasing frequency.

A skewed energy spectrum implies that the energy of the sequence is concentrated in the first few Fourier coefficients. This implies that the *effective dimensionality* of these series is actually lower than the true spectral coefficients they contain. Thus, the first few coefficients can give us a good estimate of features of the original stream for the sequences of our interest. For these reasons, we select the first N_c (called cut-off frequency) coefficients of the normal form of subsequences

represented by w_j. Thus a window can be represented as a point in $\Re^{2N_c}$ feature space[7] where each N_c dimension represents a frequency in the Fourier spectrum.

Time ordered sequence of feature points for a reference sequence R is called *feature trail* and is denoted by RF.

4.4 Grouping of Feature Points

After taking a piecewise Fourier transform on a reference sequence and selecting the first N_c Fourier harmonics we now represent each window as a point in $2N_c$ dimensional space. These feature points are also time-ordered. This framework is sufficient for us to answer range queries of size *w*. The following section illustrates this.

4.4.1 Answering Whole Match Query of Size *w*

The procedure for answering range queries [16] for a query sequence of size *w* is straightforward. Given feature points of reference sequence in $\Re^{2N_c}$ feature space and a query sequence Q of length *w*, we take the Fourier transform of the normalized query sequence and select its first N_c Fourier coefficients for the reasons outlined in Section 4.3. A query sequence represents a query point in $\Re^{2N_c}$ space. A projection of a query point in real F_1-F_2 plane of $\Re^{2N_c}$ space is illustrated by point '*q*' in Figure 4.4.

To answer a range query of threshold $\in$, that is, to find all sequences of size *w* that have Euclidean distance less than or equal to $\in$ from the query sequence, draw a hypersphere of radius $\in$ around the query point in feature space as illustrated in Figure 4.3.

All points that lie inside and on the hypersphere give us a superset[8] of the answer set. Retrieve the actual reference sequence represented by these feature

[7] Remember, FFT of a real sequence is a complex number.

[8] We are approximating windows by their first few Fourier coefficients. Consequently, when computing Euclidean distance in feature space, we loose positive terms contributing to distance calculation. Parseval's theorem guarantees that when all Fourier coefficients are employed for distance computation, then the distance in feature space equals distance in time space. Underestimating the distance (by using only first few Fourier coefficients) and still using the same threshold for query recovers a superset (containing false alarms) of the answer set.

points and compute their Euclidean distance from the query. Select those sequences that are within $\in$ of the query sequence. These sequences form the answer set.

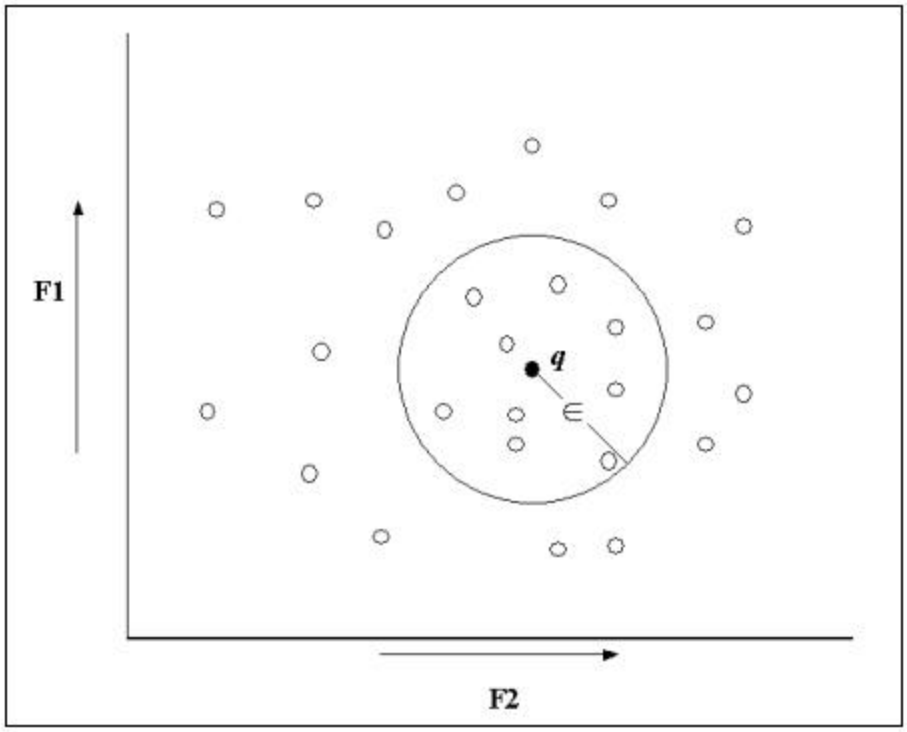


Figure 4.4: Answering whole matched range queries of size *w*.

We obviously desire to answer queries for sequences that have size greater than *w*. As discussed in Section 2.4, a large number (exactly, $((L_R\text{-}1)!*(L_Q\text{-}1)!)/2$) possible pairs of subsequences exist for similarity match, within reference and query sequences of sizes L_R and L_Q respectively. To minimize the search space for query processing, we employ an approach to aggregate these feature points in groups (segments of feature points of a windowed reference sequence) that in time domain represents a subsequence of a larger reference sequence. This is the focus of discussion in next section.

4.4.2 Answering *Longer* Queries

In order for us to answer longer queries, we need to define a reference search-space extracted from a reference sequence. We segment the feature trail in groups,

each of which represents a subsequence in the time domain. A straightforward approach would be to divide the feature trail in feature sub-trails with predetermined width of X number of feature points (for example $X = \sqrt{length(R)}$). The fundamental problem with this approach is that there is not a justifiable approach to determine the size of a feature group. We designate this method, of aggregating windows and subsequent treatment, as method FG_{naive}. Our experiments[9] have shown that this heuristic yields poor results. Figure 4.5 demonstrates the approach of having fixed window size.

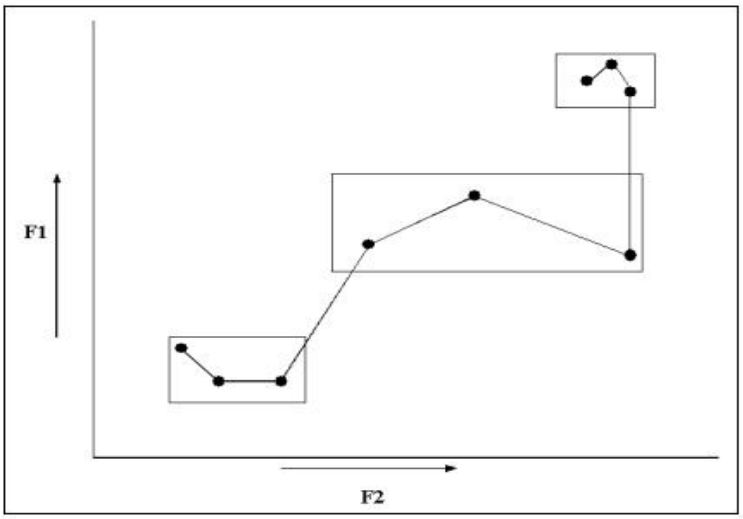


Figure 4.5: Feature groups of fixed size (X = 3) each.

We need a more *data-adaptive* approach to group feature points to form segments. One such approach is to group the points such that Euclidean distance between any two points in a group is less than or equal to a pre-determined error threshold. This technique works well when feature points of *similar* windows tend to form (hyper)sphere in feature space. We call this method $FG_{Euclidean}$. Figure 4.6 illustrates the process of aggregating points using Euclidean distance as distance metric.

[9] We postpone the discussion on experimental setup and results to Chapter 7 to maintain the continuity of our discussion here.

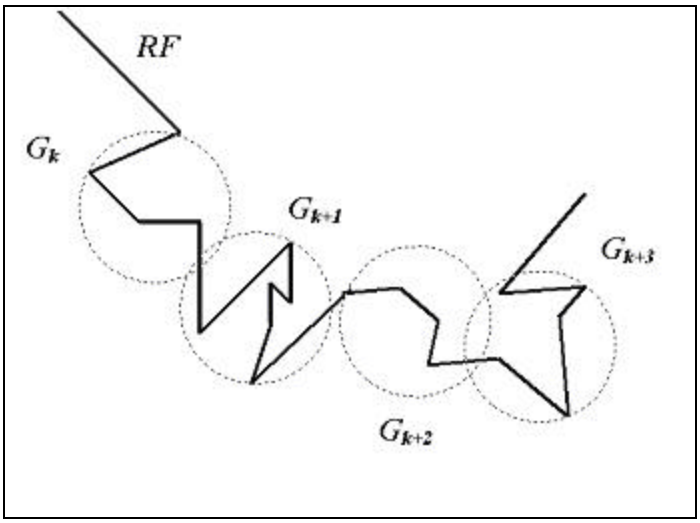


Figure 4.6: The grouping of segments using Euclidean distance as distance measure. Feature points are grouped together such that no two points have a distance greater than a threshold l_T.

However, grouping feature points using Euclidean distance suffers from a limitation. It does not take the statistical distribution of the feature points (representing windows) already assembled into a group into account before admitting a new feature point in the group. In other words, it does not give any information on how well an unknown value matches an already aggregated set of feature points. Secondly, Euclidean distance only measures a relative distance from the mean point in the group. The following example illustrates the idea.

Example 4.1: Consider the distribution of points (in 2-dimensions) in Figure 4.7. The circle indicates the Euclidean boundary with respect to the center (black point). Two hypothetical samples 'A' and 'B' are added to the distribution. By the Euclidean distance method, sample 'B' is equally likely to be classified as belonging to the group as sample 'A'. However, sample 'A' clearly lies in the elongated axis of the group points, indicating that selected frequencies in window 'A' are behaving much more like the aggregated group that those same frequencies in the spectrum of window represented by 'B'.

Generally, method based on Euclidean distance does not take into account the variability of the values in all dimensions, therefore not the *best* discriminant analysis algorithm here.

There is another distance measure, Mahalanobis distance [35], which however, does take the variability into account. Instead of treating all values equally when calculating the distance from the mean point, it weighs the differences by the range of variability in the direction of the sample point. The Mahalanobis boundary is superimposed in Figure 4.7. Mahalanobis distance constructs a space that weights the variation in the sample along the axis of elongation less than in the shorter axis of group ellipse. In terms of Mahalanobis distance measurements, sample 'A' will have a substantially smaller distance to the mean than sample 'B' since it lies along the axis of the group that has the largest variability. Therefore window sample 'A' is far more likely to be classified in the group than sample 'B'.

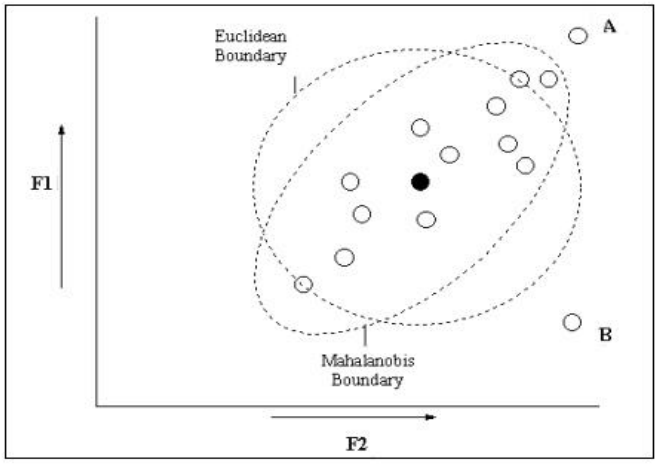


Figure 4.7:Illustration of advantage of Mahalanobis over Euclidean distance.

In the special case where the feature points are uncorrelated and the variances in all directions are the same, the boundaries represented by Mahalanobis distance is a circle, and the Mahalanobis distance becomes equivalent to the Euclidean distance. We refer the framework that employs Mahalanobis distance metric grouping as FG_{mahal}. Next section gives a brief overview of the theory behind computing Mahalanobis distance.

4.4.3 Mahalanobis Distance

The covariance of two features measures their tendency to vary together (to co-vary). Covariance is the average of the products of the deviations of feature values from their means. Let x be the variable of interest and there be n such variables, each extending itself in more than one dimension. Let $x(k,i)$ and $x(k,j)$ be the value of the k^{th} variable in i^{th} and j^{th} dimension respectively. The covariance of variables represented in feature i and feature j is defined by

$$C(i,j) = \frac{\sum_{k=1}^{n}[(x(k,i) - m(i))(x(k,j) - m(j))]}{n-1} \quad \text{.....(4.8)}$$

where $m(i)$ and $m(j)$ is the mean in the direction of feature i and j respectively. If $s(i)$ and $s(j)$ are standard deviations of variables in domain of features i and j respectively then covariance $C(i,j)$ is a number between $-s(i)s(j)$ and $+s(i)s(j)$ that measures the dependence between feature i and feature j. With $C(i,j) = 0$ if there is no dependence. Covariance also holds several interesting properties such as:

- If feature i and feature j tend to increase together, then $C(i,j) > 0$.
- If feature i tends to decrease when feature j increases, then $C(i,j) < 0$.
- If feature i and feature j are independent, then $C(i,j) = 0$.
- $C(i,j) <= s(i)\, s(j)$.
- $C(i,i) = s(i)^2 = v(i)$.

All the covariances $C(i,j)$ in N_c dimensions can be collected together to form a covariance matrix C:

$$C = \begin{bmatrix} C(1,1) & C(1,2) & \dots & C(1,N_c) \\ C(2,1) & C(2,2) & \dots & C(2,N_c) \\ \vdots & \vdots & & \vdots \\ C(N_c,1) & C(N_c,2) & \cdots & C(N_c,N_c) \end{bmatrix} \quad \text{.....(4.9)}$$

The covariance matrix is always symmetric and positive semi-definite.

Definition 4.1: Mahalanobis Distance:

Mahalanobis distance from the feature vector $\boldsymbol{x}$ to the mean vector $\boldsymbol{m}_x$ is given by $D_m = \left[(x - m_x)` C_x^{-1} (x - m_x)\right]^{1/2}$, where C_x is the covariance matrix of elements in the feature space.

It can also be shown that the surfaces on which D_m is constant are ellipsoids that are centered about the mean m_x(see[35]).

As indicated earlier Euclidean distance is a special case to Mahalanobis distance metric. Additionally, Mahalanobis distance addressed several limitations imposed by Euclidean distance. Computationally, Mahalanobis distance D_m from the point $\boldsymbol{y}$ to a set $\boldsymbol{X}$ of points is the (squared) Euclidean distance from **y** to the centroid (mean) of $\boldsymbol{X}$ weighted with respect to the variance matrix of $\boldsymbol{X}$.

Mahalanobis metric addresses a limitation imposed by Euclidean metric. While grouping feature points, there is a possibility that the features may be highly correlated. It will happen that two features that were meant to measure a characteristic are influenced by some mechanism to vary together. This correlation is identified by the Mahalanobis distance metric, which Euclidean distance might fail to detect (Figure 4.6). Mahalanobis distance can also provide curved as well as linear decision boundaries.

4.4.4 Grouping Algorithm

Given feature points of reference sequence R, represented in 1..N_c space, the strategy is to group feature points such that every point in a group does not have distance from its mean greater than predefined threshold distance of l_T. The group contains feature points, which are time ordered, and belongs to a same reference sequence. The algorithm to obtain groups is presented in Figure 4.8. The notations followed the algorithm are shown in Table 4.1

Definitions: For a group G_x of n points represented by $[((x_1^1, x_2^1, \ldots, x_{N_c}^1), (y_1^1, y_2^1, \ldots, y_{N_c}^1)), ((x_1^2, x_2^2, \ldots, x_{N_c}^2), ((y_1^2, y_2^2, \ldots, y_{N_c}^2)) \ldots, ((x_1^n, x_2^n, \ldots, x_{N_c}^n), (y_1^n, y_2^n, \ldots, y_{N_c}^n))]$, where *x* and *y* refer to real and imaginary part respectively.

- The *Center C* of the group is given by $C_{G_x} = ((x_1^m, x_2^m, \ldots, x_{N_c}^m), (y_1^m, y_2^m, \ldots, y_{N_c}^m)) = ((\frac{1}{n}\sum_{i=1}^{n} x_1^i, \frac{1}{n}\sum_{i=1}^{n} x_2^i, \ldots, \frac{1}{n}\sum_{i=1}^{n} x_{N_c}^i), (\frac{1}{n}\sum_{i=1}^{n} y_1^i, \frac{1}{n}\sum_{i=1}^{n} y_2^i, \ldots, \frac{1}{n}\sum_{i=1}^{n} y_{N_c}^i))$

- The *Distance* between a point and a group is the distance between the group center and the point. If point *p*, given by $((a_1, a_2, \ldots, a_{N_c}), (b_1, b_2, \ldots, b_{N_c}))$, is a candidate point to be inserted in group G_x then *Distance* takes following forms depending on the metric adopted.

For $FG_{Euclidean}$: $Distance(A, G_x) = [\sum_{i=1}^{N_C} (|a_i - x_i^m|^2 + |b_i - y_i^m|^2)]^{1/2}$

For FG_{mahal}: $Distance(A,G_x) = \left[\left| (a - m_x)` C_x^{-1} (a - m_x) \right| + \left| (b - m_y)` C_y^{-1} (b - m_y) \right| \right]^{1/2}$ where a, b, m_x and m_y are in their matrix notations and C_x, C_y are covariance matrix of the real and imaginary point groups respectively. $(a - m_x)`$ and $(b - m_y)`$ are transpose of the matrices $(a - m_x)$ and $(b - m_x)$ respectively.

Table 4.1: The table of notations for groups

Notation	Definition
RF	Feature trail for sequence *R*
l_T	Threshold distance
G_k	*kth* group of feature trail
$t_{sta}(G_k)$	Start time of sequence represented by G_k
$t_{end}(G_k)$	End time of sequence represented by G_k
C_k	Center of G_k
N_k	Number of elements in G_k

Suppose we have *M* groups formed by *group_features* algorithm. Each of these groups is non-overlapping sub-trail of feature trail. Each of these sub-trails corresponds to a subsequence of reference sequence in time domain. Note that these subsequences of reference sequence might be overlapping.

In the next section we propose a data structure to hierarchically represent these groups. This would subsequently form our search space for similarity queries originating from query sequence.

Algorithm *group_features*(*RF*, l_T):
Input: Feature trail *RF*, l_T
Output: Group G_k such that distance between C_k and any element $G_k(i)$ $(1 \leq i \leq N)$ is not greater than l_T
Begin
Initialization
1. "Push" feature points in a stack in non-decreasing order of time (top element corresponding to the newest time entry).
2. Set counter for number of group , $j = 1$.
3. Do an operation "pop up" of a point *A*.
4. Create group G_1, with center C_1 equal to *A*.
5. Set number of elements in G_1 , $N_1 = 1$.

Main Loop
6. While (stack is not empty)
{
6.1 Do an operation "pop up" of a point *A*,
6.2 Compute Distance between A and current group center C_j, *dist* = *Distance*(*A*,G_j),
6.3 If *dist* > l_T
then
/* Create New Group */
6.3.a1 Create New group, $j = j+1$,
6.3.a2 Assign center $C_j = A$,
6.3.a3 Set count for new group, $N_j = 1$,
6.3.a4 Goto step 6.
else
/* Insert element in group */
6.3.b1 Insert *A* into group j, $N_j = N_j + 1$,
6.3.b2 Re-compute center C_j of group j,
6.3.b3 Recompute Distance $distance_k$ $(1 \leq k \leq N_j)$ from center for each element in group,
6.3.b4 If a $distance_k > l_T$
6.3.b4.1 Delete last element from *jth* group, $N_j = N_j - 1$,
6.3.b4.2 Recompute Center C_j of *j*th group,
6.3.b4.3 Push *A* in stack,
6.3.b4.4 Goto Step 3.
6.3.b5 Goto Step 6.
}
End

Figure 4.8: Algorithm *group_feature* to form groups in feature space.

4.5 The *FG*-Tree

In this section we introduce Feature Group tree (*FG*-tree) and illustrate the process of building it bottom-up from groups derived in feature space. We start our discussion with defining operation *merge* for two groups to form a larger group.

Two groups G_a and G_b can be merged iff $b = a+1$ or $a = b+1$. This essentially means that we can only merge two groups if they occur consecutively within a same feature trail.

Let G_j and G_{j+1} be two groups with centers C_j and C_{j+1} respectively. The group obtained by merging G_j and G_{j+1} is called G_j^2 with superscript illustrates next higher level in the *FG*-tree. Correspondingly the center of the merged group is denoted by C_j^2. Group G_j^2 contains all elements of G_j and G_{j+1}. Generally, operation merge is represented as

$$G_j^{i+1} = G_j^i \oplus G_{j+1}^i.$$

Number of elements in G_j^2 are $N_j + N_{j+1}$ and the center of G_j^2 is given by

$$C_j^2 = \frac{C_j N_j + C_{j+1} N_{j+1}}{N_j + N_{j+1}}.$$

We bottom-up build the tree with consecutively performing operation *merge*. The *FG*-tree for a reference sequence is given in Figure 4.9.

Note the following in the *FG*-tree.

- The leaves (level 1) correspond to the groups obtained by g*roup_feature* algorithm.
- The number of levels equal to number of leaves, which is equal to number of groups in reference sequence.
- The number of nodes reduces by one as we move one level higher in the tree.
- Consecutive groups at level 2 to level *M* contains overlapping features, hence contain redundancy.

Note that *FG*-tree may not require any additional memory requirement to store. Once lowest level groups are computed, subsequent levels can be computed by

recursively merging the lower level groups. It is not a required to store all levels in a memory-critical environment. Moreover, during the similarity search between reference and query subsequences, the search (hence, the building of *FG*-tree) can be stop once all (or sufficient) similar pairs of subsequences and discovered.

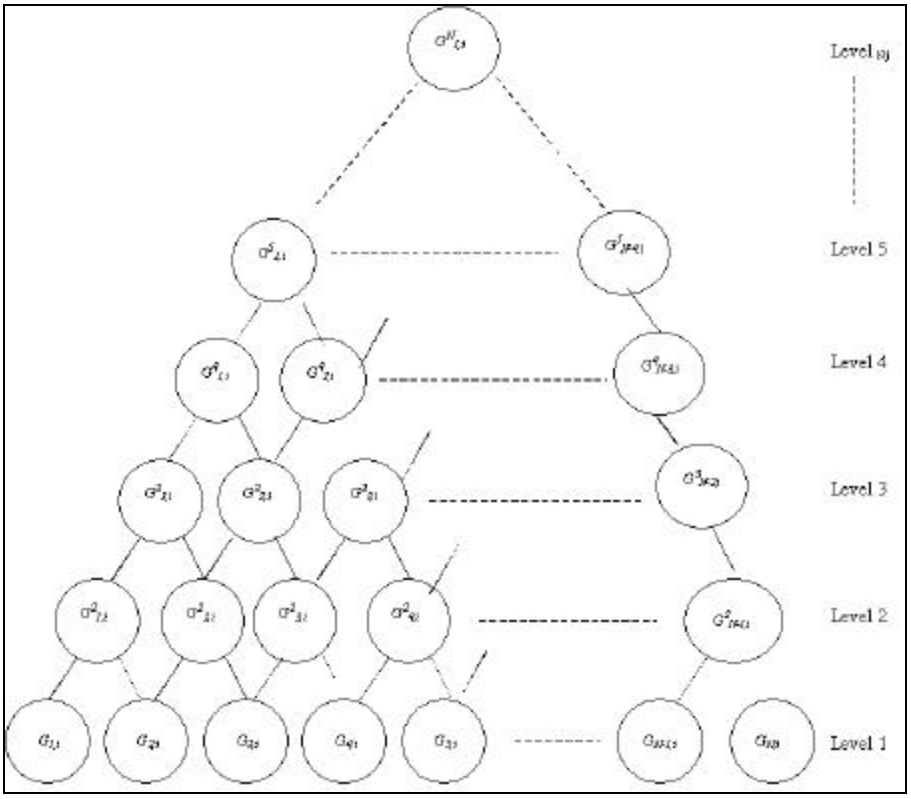


Figure 4.9: FG-tree for feature trail *RF*.

Once we have built the *FG*-tree we have recognized different (sub)subsequences of reference sequence in time domain which would be employed for further similarity analysis.

4.6 Summary

In this chapter we presented a framework for representation of time sequence in a $\Re^{2N_c}$ feature space where each N_c axis corresponds to a Fourier coefficient. We then introduced a grouping algorithm that groups features to segment trails in feature space employing fixed-sized windows, Euclidean distance metric or Mahalanobis distance metric. These groups derived out of the algorithm are employed to

recursively build a *FG*-tree whose height is equal to the number of groups in the bottom-most level. In subsequent discussion we present an indexing technique to index these groups to define a search space for similarity queries.

Chapter 5

Building the *FG*-index

In the previous chapter we described the process of identifying groups of feature points from a trail in feature space. Each of these groups represent time-ordered subsequences of reference sequence in time domain. These subsequences form a shape within the region identified by the group. We call these shapes as Objects O_k in time space, the subscript k corresponding to the group G_k they represent. The start and end time-points for these objects, represented by $t_{sta}(O_k)$ and $t_{end}(O_k)$ respectively are stored in a lookup table indexed by their respective object *id*s.

In this chapter we present a technique to obtain geometric properties and quantitative descriptors of these uniquely identified objects.

5.1 Shape Quantification from Time Subsequences

The strategy to discover shape from an Object and then computing and assigning unique attribute descriptors for these identified objects in reference data sequence is by achieved by enclosing each object in a minimum area rectangle. The minimum area rectangle is also called Minimum Enclosing Rectangle (MER). The object represented in the time subsequence is preprocessed to obtain its convex hull(see [53] for definition). This convex hull is then enclosed in its MER. Obviously, the geometric attributes associated with each object are

1. The orientation angle between the MER's length and the horizontal axis (X-axis),
2. The aspect ratio (breadth/length) of the MER,
3. The length of the MER and
4. The center, represented by x and y Cartesian coordinates, of the MER.

Each of these geometric descriptors has an advantage in shape representation and matching. The orientation angle gives the amount by which the query object must be rotated with respect to the represented reference object for similarity calculation. The aspect ratio approximately characterizes the scale-free shape of the time subsequence

enclosed. The aspect ratio of the MER gives the factor by which the query object should be enlarged or reduced to calculate match.

The center of the MER of the object provides the correct position where the Query object for proper alignment and subsequent processing.

5.2 The Minimum Enclosing Rectangle Algorithm

The MER of a two-dimensional object can be computed by using the following algorithm developed by Toussaint[1] [see 53]. Given a polygon P, it is possible to enclose it in a rectangle R_P of least possible area in $O(\log N)$ time, where N is the number of points in G, as demonstrated by the following algorithm (from [53]). Before we employ the Toussaint's algorithm, let us first discuss some preliminary but important ideas given as follows. A group of attribute points G can be enclosed in a rectangular bounding box (not necessarily MER), with its sides parallel to the axis, in $O(N)$ time, where N is the number of points in G. So if we construct the MER of Convex Hull C_G of G, it would be the MER for the group G itself, because the MER (called MER_G) of a group G contains the convex hull. Given a set of points G, its convex hull can be constructed in $O(N \log N)$ time[2], where N is the number of points in G. The convex hull C_G of a set of attribute points G is a convex polygon C enclosing the group. Freeman and Shapira in their theorem given in [18], have proven that the minimum-enclosing rectangle of a convex polygon P has a side parallel to edge of convex hull P.

This technique for obtaining the MER is popularly called *Rotating Calipers* method given as follows. Let $v_1,\ldots,v_N$ be a vertices of a convex polygon P, in clockwise order. Consider the bounding box B_i of P, one of whose sides contains the side $\overline{v_{i-1}v_i}$. Consider the lines L_1, L_2, L_3 and L_4 obtained by extending infinitely in both directions,

[1] The original algorithm is on edge pixels of the object. The running time of the algorithm is O(m log m), where m is the number of edge pixels of the object.

[2] The convex hull of a set of points is the boundary of the intersection of all convex sets containing S. Alternatively; it is the smallest enclosing convex polygon for the set S. The convex hull of a set of K points can be obtained by the Graham's scan algorithm (see [24]), which is optimal (due to the fact that sorting of K points itself takes $O(K \log K)$ time), and takes $O(K \log K)$ time.

the side of B_i. Let v_1, v_2, v_3 and v_4 be the vertices of P in contact with L_1, L_2, L_3 and L_4 respectively. If $v_{i-1}v_i \subset L_1(say)$ choose $v_{i_1} = v_j$.

The lines L_1, L_2, L_3 and L_4 are perceived as rotating calipers, that rotate clockwise preserving the relation $L_1 \perp L_2 \perp L_3 \perp L_4$. The positions of the calipers are restricted to those where one of the four lines contains a side of C (due to remark above).

The computation of the next position of the calipers is done by computing the angles q_i between L_i and $v_i v_{i+1}$; i=1,2,3,4. The smallest nonzero q_i is chosen to be the angle through which the calipers are rotated. The rotations will be carried out until calipers return to the configuration they started off with.

It is possible to find more than one rectangles with same (minimum) area for a time subsequence. These rectangles might have different geometric descriptor. However it is possible to obtain all the solutions using the described algorithm.

The MER algorithm can be used to obtain quantitative descriptors of subsequences identified in the reference sequence. The subsequences are classified according to certain geometric attributes of the objects they contain. For each subsequence *R*, the object O_R in R is associated with its MER. The feature points contained in an MER, and the aspect ratio $r_A(O_R)$ of the MER, given by the breadth/length is stored along with the length $l(O_R)$ of the MER in an attribute vector. The center $c(O_R)$ of the MER, given by $(c_x(O_R), c_y(O_R))$ is also stored for matching purposes. The angle that length of MER makes with positive horizontal axis, $q(O_R)$ is also recorded.

The attribute vector of a reference object is given by

$$A(O_R) = [q(O_R), r_A(O_R), l(O_R), \mathrm{c_x}(\mathrm{O_R}), \mathrm{c_y}(\mathrm{O_R})].$$

Figure 5.1 illustrates the process of derivation of MER from a reference sequence and extracting the MER's attribute vector from a time subsequence.

The query sequence presented to the feature database is preprocessed for similarity matching as follows: A query sequence is segmented into groups by the algorithm *group_features* (Figure 4.7). A *FG*-tree is built for these groups. For each of the corresponding group of points in the *FG*-tree, the MER is computed using the

algorithm presented in Section 5.1. The aspect ratio $r_A(O_Q)$ of each of these MER's along with the magnitude of its length $l(O_Q)$, and the angle $q(O_Q)$ of the length of the MER with the horizontal axis are stored in an attribute vector $A(O_Q)$ representing the geometric attributes of the object with which the MER is associated. As in the representation of attribute vector of reference object, the attribute vector of query object can be represented as $A(O_Q) = [q(O_Q), r_A(O_Q), l(O_Q), c_x(O_Q), c_y(O_Q)]$.

The first component of $A(O_Q)$, that is q, gives us the orientation of the rectangle with the horizontal axis.

The second component of $A(O_Q)$, that is r_A, gives a scale-free characterization of the 'shape' of the query object it is associated with. This is used to locate the right *class* of objects O_t in the reference object database.

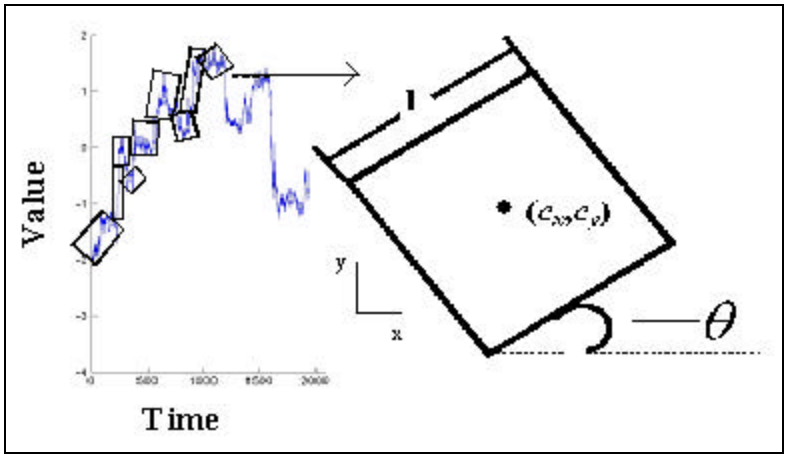


Figure 5.1: Process of derivation of MERs from a time sequence and extraction of geometric attributes from a MER.

The third component of $A(O_Q)$, that is l, is the length of the MER which divided by the reference object MER's length gives us a scale factor by which the query object must be expanded or contracted in order to match with the reference object (Chapter 6 describes procedure of building index using these geometric descriptors and consequently running queries on this index).

Once the candidate set is recognized the smaller object (belonging to either reference or query sequence) is scaled higher and aligned to the center of the object in candidate (sub)class. Several candidate objects with same aspect ratio may have the same orientation angle, in which case similarity matching is carried out with the corresponding query object, after proper scaling them. An time subsequence may have more than one MER associated with it, with possibly different aspect ratios and angles. In such a case all the possible MERs with their corresponding attribute vectors are recorded and employed.

5.3 Summary

In this chapter we discussed a technique for pose extraction from objects recognized in a sequence. Their corresponding attribute vectors are quantitative descriptors of these objects. These attribute vectors lay the necessary framework for our search index of reference subsequences presented in the next chapter.

Chapter 6

Similarity Search Using *FG*-Index

In this chapter we describe the process of building an index structure based on geometric descriptors of objects in reference sequence. This index is employed to answer *similar-pairs* query to recognize similar objects, and their corresponding similar subsequences.

6.1 Building the *FG*-Index

The objects in the reference database are stored in the index as follows. If the number of groups represented in *FG*-tree is *K*, then each of the objects in our index contains one of objects from *K* groups. The MER for each of the *K* groups is obtained and their orientation angles computed. The index is formed for all unique orientation angles. If the *K* objects have *r* distinct orientation angles in all ($r \leq K$), then the index is divided into r classes $C^1, C^2, ..., C^r$ parameterized by the orientation angles $q^1, q^2, ..., q^r$ respectively. Each class C^i, $i \leq r$ contains all the objects $O_k (k \leq K)$ for which $q(O_k) = q^i$. Within each class C^i, the objects are (sub)classified by their aspect ratios. If each of the objects in a class C^i has s_i distinct aspect ratios then each class C^i is further divided into s_i sub classes $C^{i1}, C^{i2}, ..., C^{is_i}$ objects having s_i distinct aspect ratios $r_a^{i1}, r_a^{i2}, ..., r_a^{is_i}$. Within each subclass C^{ij}, the objects may be ordered arbitrarily or according to some probability threshold that gives priority to (sub)sequences that are most likely to occur as the query (sub)sequence. This choice of ordering is dictated by and may be suited to the application in hand. For example, in comparing two stock data sequences, we might desire to find similar pairs of subsequences that occur immediately after the declaration of quarterly results by companies. Figure 6.1 illustrates the *FG*-index organization.

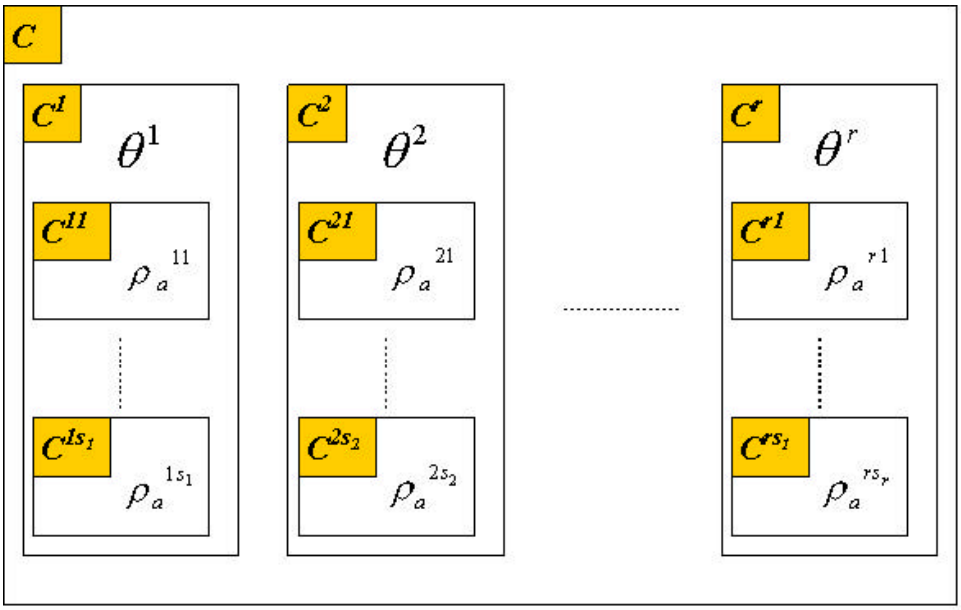


Figure 6.1: *FG*-Index organization- the class identifiers are in shaded boxes.

This method of searching a similar class is good enough if the number of candidate MERs in each sub-class is small. However, in real applications, each of the classes may contain a large number of subsequences, in which case, the search reduces to sequential scanning within a sub-class, which is still not sufficiently efficient. To enable an efficient search, a *distance table* is built for all objects with same size within a sub-class where the distances between them are stored.

Denote the distance $D(O_a, O_b)$ between (one-dimensional) objects O_a and O_b in the same sub-class and having same size by D_{ab}, given by,

$$D_{ab} = D(O_a, O_b) = \left| \left[\sum_{i=1}^{L} [O_b(i) - O_a(i)]^2 \right]^{\frac{1}{2}} \right|$$, where L is the equal length of objects

under consideration. The following holds true,

$D_{ab} = D_{ba}$? R_0^+, and $D_{aa} = 0 \; \forall \; a$.

Table 6.1 illustrates the *distance*-table for 4 objects.

The search using *distance table* is done as follows. Within all subsequences (with same length) in a sub-class, an object O_a is arbitrarily selected. The distance with the corresponding query object is calculated. The distance obtained from this match is used to look up the *distance table* in the row corresponding to O_a. If the computed distance

with O_b is less than that of O_a from *b*, then the *recognized* subsequence is redirected to object O_b and the match is performed with this new object. The redirection sequence thus formed does not allow the recurrence of an object. The same procedure is repeated for each same-length (sub)subsequence until a match is found.

Table 6.1: *Distance table* for 5 objects.

O_{ab}	O_1	O_2	O_3	O_4	O_5
O_1	0	D_{12}	D_{13}	D_{14}	D_{15}
O_2	D_{21}	0	D_{23}	D_{24}	D_{25}
O_3	D_{31}	D_{32}	0	D_{34}	D_{35}
O_4	D_{41}	D_{42}	D_{43}	0	D_{45}
O_5	D_{51}	D_{52}	D_{53}	D_{54}	0

The search can be possibly simplified using *distance table*. Within all subsequences (with same length) in a sub-class, an object O_a is arbitrarily selected. The distance with the corresponding query object is calculated. The distance obtained from this match is used to look up the *distance table* in the row corresponding to O_a. If the computed distance with O_b is less than that of O_a from *b*, then the *recognized* subsequence is redirected to object O_b and the match is performed with this new object. The redirection sequence thus formed does not allow the recurrence of an object. The same procedure is repeated for each same-length (sub)subsequence until a match is found.

If the orientation angles and the aspect ratios are closely spaced then the errors in computing the aspect ratios and angles of reference objects may lead to a incorrect choice of classes, in which case the search for a match within the incorrectly chosen class may lead to unsatisfactory matches. To provide for this eventuality, one may classify a range of values rather than discrete values to parameterize a class or a subclass.

6.2 Whole Match in Subsequences Query

Given a reference sequence R of size L_R and *FG*-index of R and a query sequence Q of size L_Q, we are interested in finding all subsequences in R that match Q. The search can be performed using *FG*-index as follows.

1. Find the MER(s) of the query sequence using techniques outlined in Chapter 5.
2. For each MER found in step 1:
    2.1. Find the attribute vector $A_p(O_Q)$.
    2.2. Find a class C^i in index with $A_p q = q^i, i \le r$.
    2.3 Within C^i, find a sub-class C^{ij}, $j \le s_i$, such that $A_p.r_a = r_a^{ij}$.
    2.4 For each object in C^{ij} in step 2.3 apply the center of the query object to center of the reference object.
    2.5 Appropriately *DOLS* scale the larger object (reference subsequence or query sequence) and find distance.
    2.6 Lookup the table for start and end points of reference object and report location and distance.

6.3 *Similar Pairs* Subsequence Matching

Given a reference sequence R of size L_R and *FG*-index of R and a query sequence Q of size L_Q, we are interested in finding all pairs of subsequences in R and Q that match each other. The search can be performed using *FG*-index as follows.

In the preprocessing stage, construct a *FG*-tree for the query sequence using techniques outlined in Section 4.4 and 4.5. We call the *FG*-tree of query sequence as *Q*-tree to differentiate it from *FG*-tree of the reference sequence.

For each i^{th} object in *Q*-tree find its corresponding attribute vector A_{Qi}. The *FG* index is searched for the class that have same (or within a threshold range in which cases one or more classes are selected) orientation angle q_Q as that of the query object. The retrieved classes are searched for sub-class having aspect ratios that are same as that of query sequence. The center of the query subsequence is applied to the center of the candidate object and the longer subsequence (query or reference) is appropriately scaled to meet the size of the shorter (reference or query) subsequence. The distance function is calculated for each of the candidate objects within the reference sub-class and all the objects that have distances that are within an error threshold are reported as matched. Repeat the above procedure for remaining objects in *Q*-tree.

6.4 Summary

In this chapter we have presented an indexing method for indexing objects discovered from reference sequence based on their attribute vectors. The query sequence is then *preprocessed* to query this index. We then presented a framework to answer whole matched subsequence query using the proposed index. We then addressed the problem of *similar-pairs* query using *FG*-index.

Chapter 7

Experimental Results

In this chapter we present various implementation experiments we performed, using the framework presented in previous chapters, and their significance. The experiments were performed using the following two varieties of data sets.

1. **Random Walk:** We generated a synthetic data sequence $\vec{x}=[x_t]$, such that, $x_{t+1} = x_t + z_t$ where z_t is a uniformly distributed random variable within the range [-500, +500].
2. **Real Exchange Rate Data:** We obtained real data for exchange rate between Swiss-Franc and US-Dollar from ftp://ftp.santafe.edu/. There are total of 65536 records. Each data point is a real number. Sequences were selected at random from these records.

7.1 Effect of *DOLS* Scaling on Harmonic Behavior

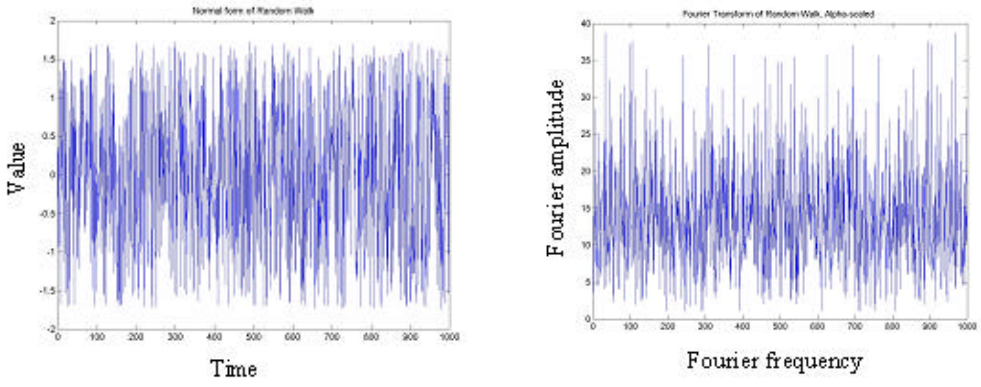
An important desired ability of similarity search in time sequences is the capability to index sequences such that a unified framework for query based retrieval can be employed. In [2] authors propose indexing sequences based on their first few Fourier coefficients. The authors did not deal with sequences of unequal sizes, which was the focus of our research. Based on our experimental results presented in this section, we propose a technique to integrate sequences of unequal sizes in the existing index.

In the following experimental setup, we investigated the degree of preservation of scaled Fourier coefficients of the original data stream with Fourier coefficients of time-scaled sequence obtained using dynamic overlay and summation technique. Based on the observations from our experiments, we propose an indexing schema for unequal size sequences.

In following experiment we took a *random walk* sequence and time compressed it using *DOLS* technique using different window sizes of approximate length $2\sqrt{L}$, $\sqrt{L}$, $0.5\sqrt{L}$ and $0.25\sqrt{L}$, where L is the length of random walk sequence. We then compared

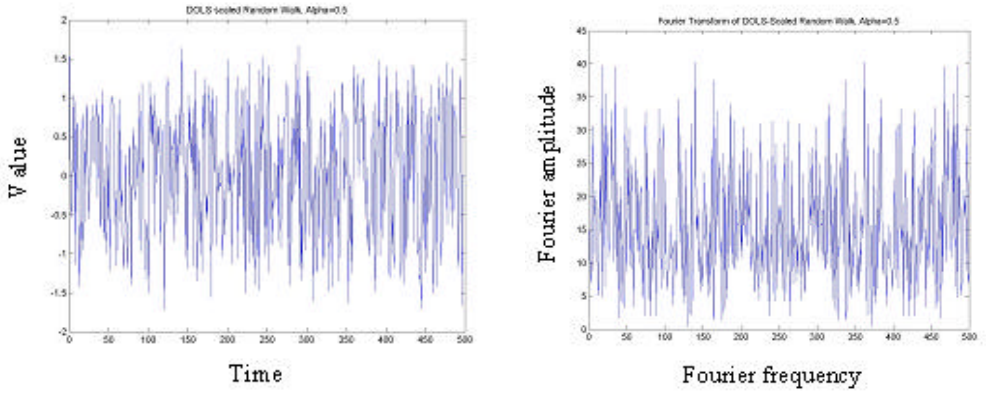
their first K Fourier coefficients (using different values of K) with the a -scaled first K Fourier coefficients of the original sequence.

Figure 7.1(a) depicts the normal form of random data (L = 1000) and Figure 7.1(b) shows its corresponding Fourier transform. We time scaled this sequence using *DOLS* using a window size of $\sqrt{L}$ ($\approx$ 31). The TSM factor a =0.5 was used. The plot of time-scaled sequence is shown in Figure 7.2(a) and its corresponding normalized amplitude Fourier transform is depicted in Figure 7.2(b).



(a) (b)

Figure 7.1: (a) Normal form of random walk (b) Its a -scaled Fourier transform amplitudes.



(a) (b)

Figure 7.2: (a) *DOLS* scaled random walk (w = 31, a =0.5) (b) Its corresponding Fourier transform amplitudes.

We repeated the exercise and *DOLS*-compressed (a =0.5) random walk (L = 1000) using different values of window size ($2\sqrt{L}$,$0.5\sqrt{L}$ and $0.25\sqrt{L}$). We then selected the first K (K = 10, 20, 30, 40, 50) Fourier coefficients for both original sequence and its corresponding *DOLS*-compressed sequence.

Error between a -scaled first K Fourier coefficients of original sequence and corresponding first K Fourier coefficients of the a -scaled sequence was computed to estimate the loss in strong Fourier harmonics. If $F_a^i(S)$ indicates the a -scaled i^{th} coefficient of the original sequence (S) and $F^i(S_a)$ indicates the i^{th} coefficient of a -scaled sequence S, then the error (see [6]) to estimate the loss in Fourier coefficients is defined as $\frac{\sum_{i=1}^{K}\left|\frac{F_a^i(S)-F^i(S_a)}{F_a^K(S)}\right|}{K}$. The results are plotted in Figure 7.3.

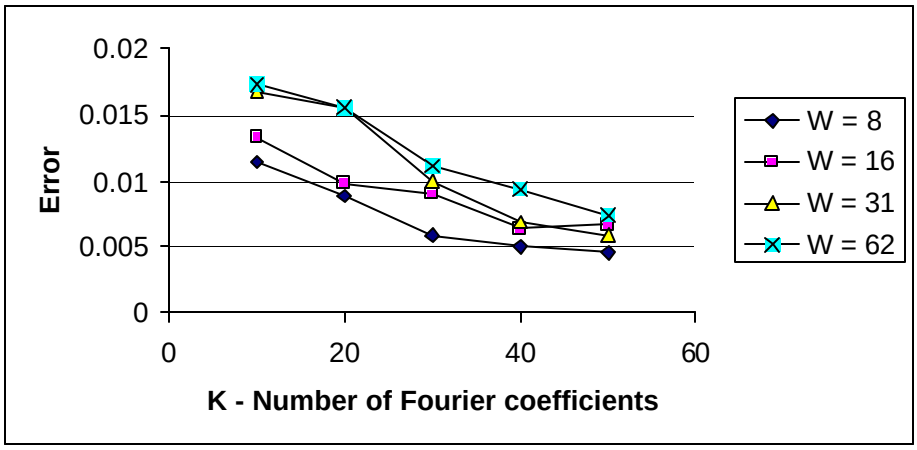


Figure 7.3: Plot of error between first K Fourier amplitudes for different values of w for random walk.

We now repeated the experiment using different values of TSM factor a (alpha) and constant value of window size ($\approx \sqrt{L}$). The results are plotted in Figure 7.4.

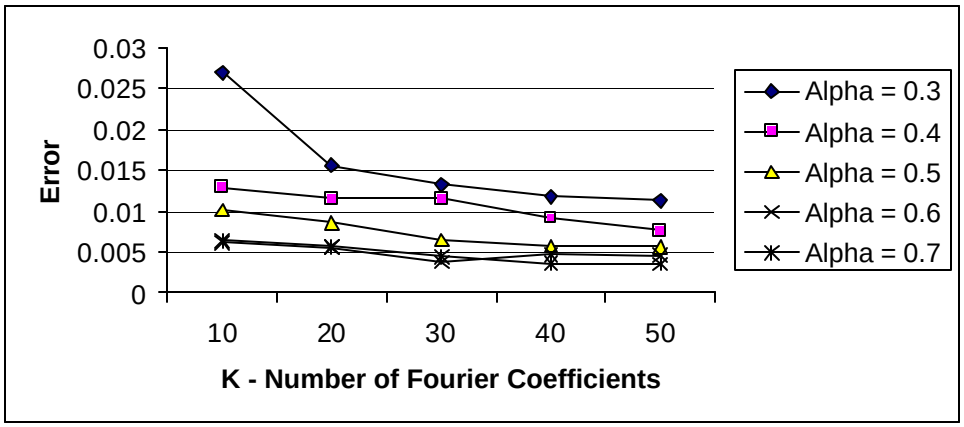
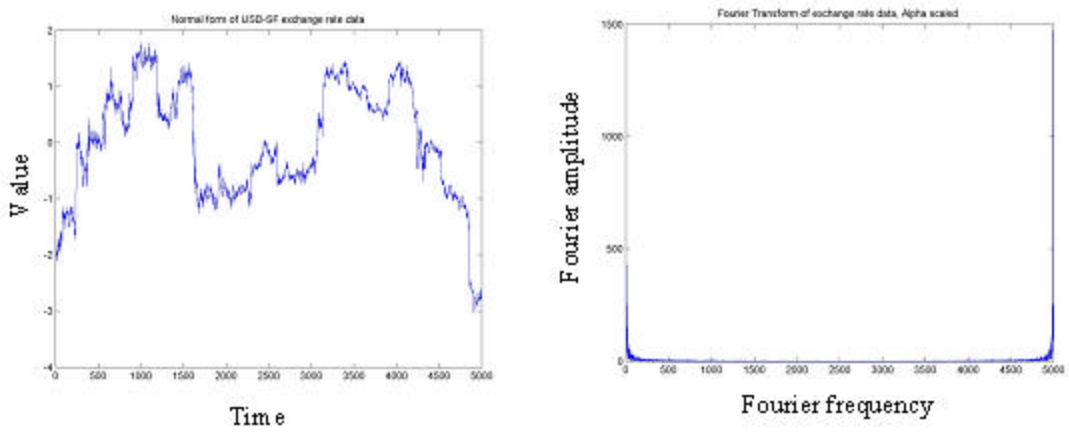


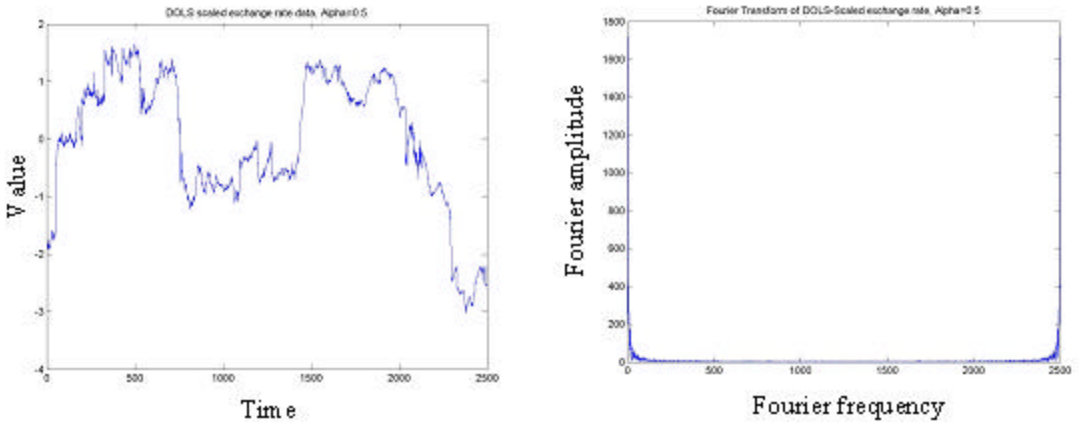
Figure 7.4: Plot of error between first *K* Fourier amplitudes for different values of a for random walk.

We repeated the experiments using real exchange rate data. The plots of normal form of exchange rate data and its corresponding Fourier transform are depicted in Figure 7.5 (a) and (b) respectively. Similarly, *DOLS* scaled exchange rate and its corresponding Fourier transform are plotted in Figure 7.6 (a) and (b) respectively. Note the skewed energy spectrum of the Fourier transform for real stock data as compared to spectrums depicted in Figures 7.1(b) and 7.2(b) for random walk. We again computed the error for different values *w* and *K*. The results are plotted in Figure 7.7.



(a) (b)

Figure 7.5: (a) Normal form of exchange rate data (b) Its corresponding a -scaled Fourier transform.



(a) (b)

Figure 7.6: (a) *DOLS*-scaled exchange rate (w = 31, a =0.5) (b) Its corresponding Fourier transform.

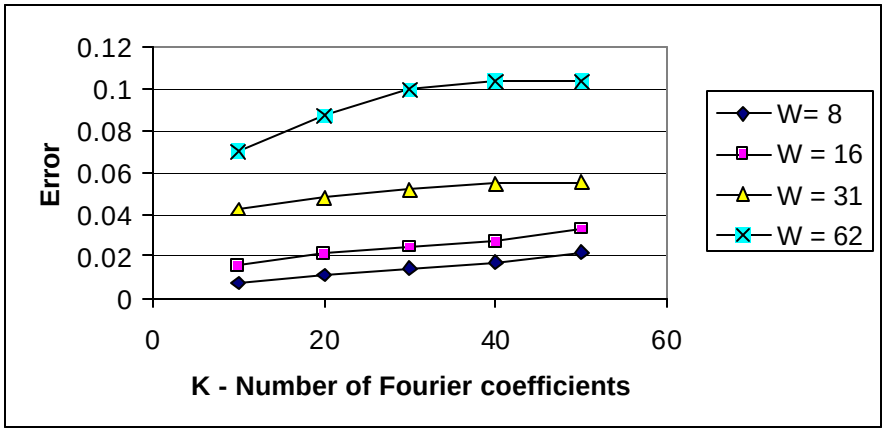


Figure 7.7: Plot of error between strong Fourier amplitudes for different values of *w* for real exchange rate data.

In next setup we kept window size constant ($w = 31 \approx \sqrt{L}$) and computed error for different values of TSM factor a (alpha). The results are shown in Figure 7.8.

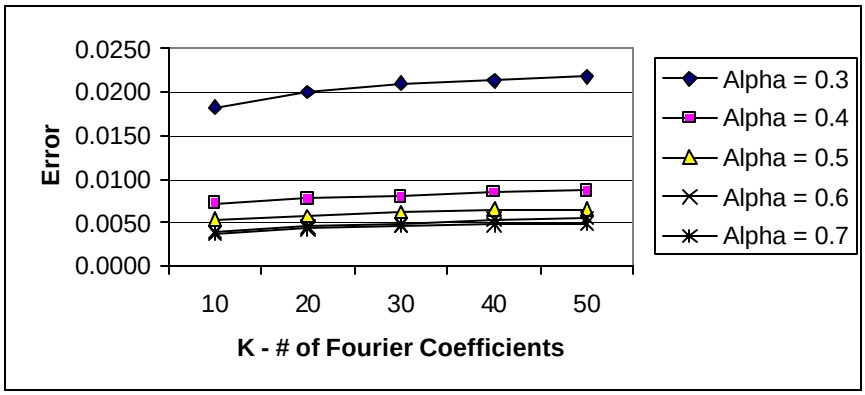


Figure 7.8: Plot of error between first *K* Fourier amplitudes for different values of *a* for real exchange rate data.

For a constant value of *K*, the error was observed to increase with the increase in window size both for random walk and real exchange rate data. The error was observed to decrease with the increase in TSM factor. This was expected as greater TSM factor corresponds to lesser time scaling.

In random walk, the error decreased with an increase in value of *K* (for constant *W*), signifying that scaled higher frequencies and corresponding higher frequencies of the scaled signal tended to be similar. However, for real-exchange rate data the error increased signifying disparity among the high frequencies.

Generally, a very low value of error can be obtained for a certain value of window size, both for random walk and real exchange data. For example, for *W*=16 in random walk (in Figure 7.3) the error remained below 0.015 (1.5 %). Similarly, for *W*=16 in real exchange rate data (Figure 7.7) remained below 0.04 (4.0%). Specifically, a lowest value of 0.005 (0.5%) for *W*=8, *K*=50 was observed for random walk (Figure 7.3). Similarly, a lowest and a value of 0.01 (1.0%) for *W*=8, *K*=10 was observed for real exchange rate data (Figure 7.7).

A low value of mean square error indicates that relative amplitudes of first few frequencies (which can be employed for indexing) tend to behave similarly after time-scale compression using *DOLS*. For real exchange rate data, these first few frequencies are also most important (strongest) frequencies (following the discussion presented in section 4.3). This observation is especially important for indexing techniques that can

employ Fourier coefficients as indexing attributes (as in [2,15] and our index as proposed next).

Our experiments, as presented above, have shown that significantly low mean square error between first few a -scaled Fourier coefficients of the original signal and first few coefficients of *DOLS* time-scaled signal can be obtained both for random walk and real stock data.

We therefore recommend using the existing Fourier index (of selecting first few Fourier coefficients) to store the sequences of unequal sizes by selecting their first K a -scaled coefficients, and then retrieve them as possible candidates using the index (like the authors in [2] do for equal sequences). Once the candidate sequences are recognized, time-scale only those sequences appropriately to a common size (following recommendations presented in Section 3.2.3), which are present in the retrieved set. In the next step, dismiss false alarms by computing actual Euclidean distance between query sequence with time emblems of unequal-sized sequences and same-sized retrieved sequences. The resultant set forms the answer set to the posed range query.

7.2 Running Similarity Queries Using *DOLS*

To study the effects of *DOLS*-scale modification for similarity queries, we formed a database *DB* with 20 sequences distributed as shown in Table 7.1. $f(x)$ was chosen as function $e^{-x}\sin(x)+19x^2$.

A function $u(x)$ was formed by adding a noise (with high frequency and very low amplitude) function $n(x)$ to $f(x)$ such that $u(x) = (e^{-x}\sin(x)+19x^2)+ 0.1\cos(10x)$. Samples were taken randomly within a interval. The query sequences of size 300 were chosen according to table 7.2.

Table 7.1: Database *DB* of 20 sequences.

SEQUENCE ID	SIZE	DESCRIPTION
*R*1	500	Sampled function $f(x)$
*R*2	400	Sampled function $f(x)$
*R*3	300	Sampled function $f(x)$
*R*4	200	Sampled function $f(x)$
*R*5	100	Sampled function $f(x)$
*R*6	500	2.3 * R1

Table 7.1 (Cont.)

*R*7	400	3.1 * R2
*R*8	300	2.4 * R3
*R*9	200	3.2 * R4
*R*10	100	1.2 * R5
*R*11	500	Sampled function $u(x)$
*R*12	400	Sampled function $u(x)$
*R*13	300	Sampled function $u(x)$
*R*14	200	Sampled function $u(x)$
*R*15	100	Sampled function $u(x)$
*R*16	500	Random walk
*R*17	400	Random walk
*R*18	300	Random walk
*R*19	200	Random walk
*R*20	100	Random walk

Table 7.2: Query sequences for range query.

SEQUENCE ID	DESCRIPTION
*Q*1	Sampled function $f(x)$
*Q*2	Sampled function $u(x)$
*Q*3	2.8 * R3
*Q*4	Random walk

The sequences in the database are converted to their normal form and then appropriately *DOLS*-scaled to a common size of 100. The query sequences are also converted to their normal forms and *DOLS*-scaled to a size of 100. The Euclidean distances of appropriately scaled elements of database from sequences scaled *Q*1, *Q*2, *Q*3 and *Q*4 are shown in Figures 7.9, 7.10, 7.11 and 7.12 respectively. The distances are measured from the center (query sequence being referred as center).

Note the following in Figure 7.9 through Figure 7.12.

- *R*1 through *R*5 (same $f(x)$ sampled at different rates), *R*6 through *R*10 (amplitude modified $f(x)$), *R*11 through *R*15 (sampled noise added $f(x)$) cluster around the query sequences *Q*1(sampled $f(x)$) , *Q*2(noise added $f(x)$) and *Q*3(amplitude modified $f(x)$). Random sequences (*R*15 through *R*20) are well isolated from the cluster.

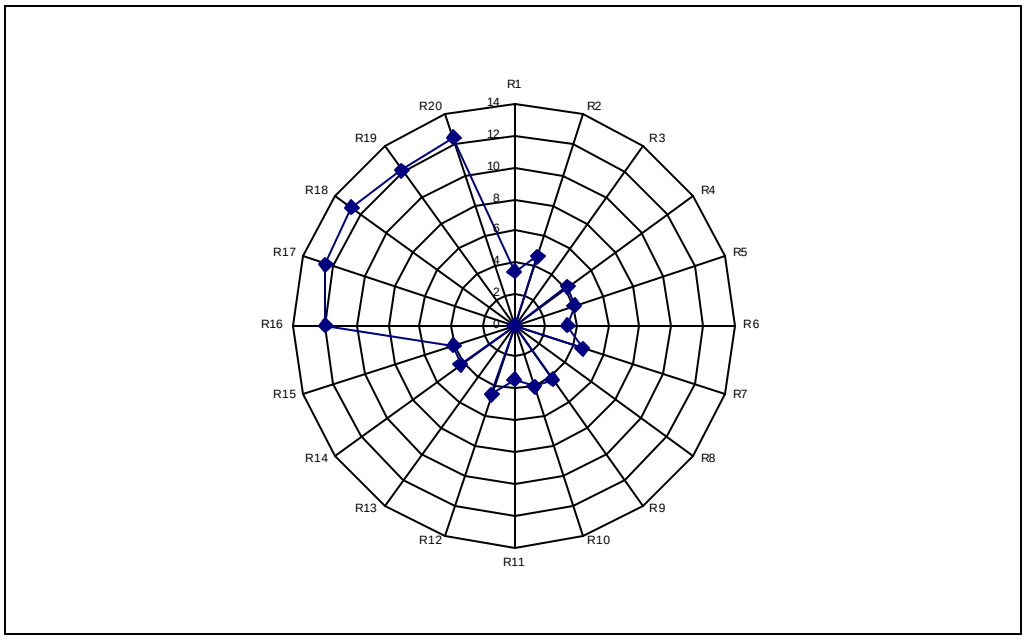


Figure 7.9: Euclidean distances of *DB* elements from *Q*1.

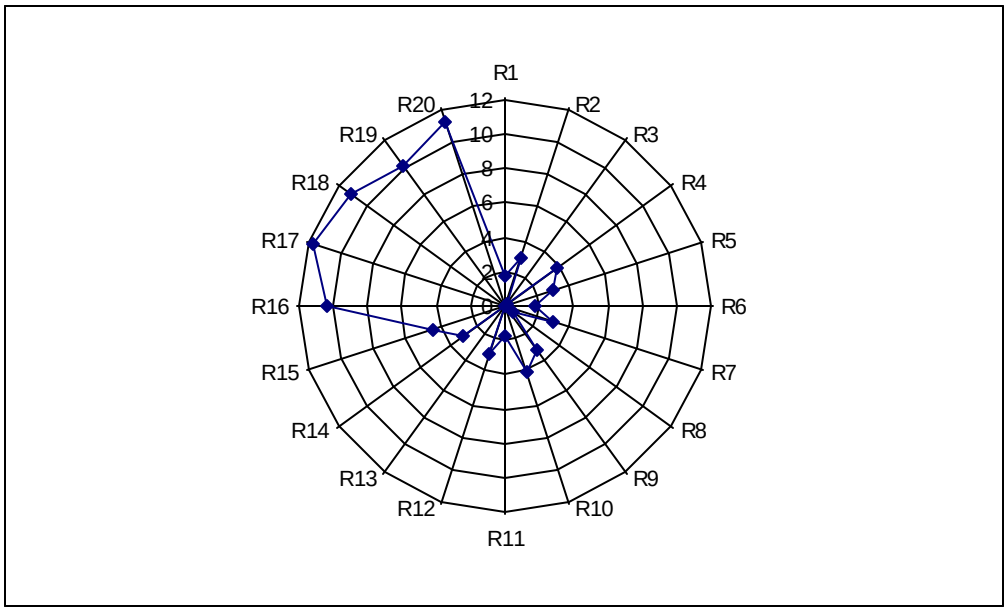


Figure 7.10: Euclidean distances of *DB* elements from *Q*2.

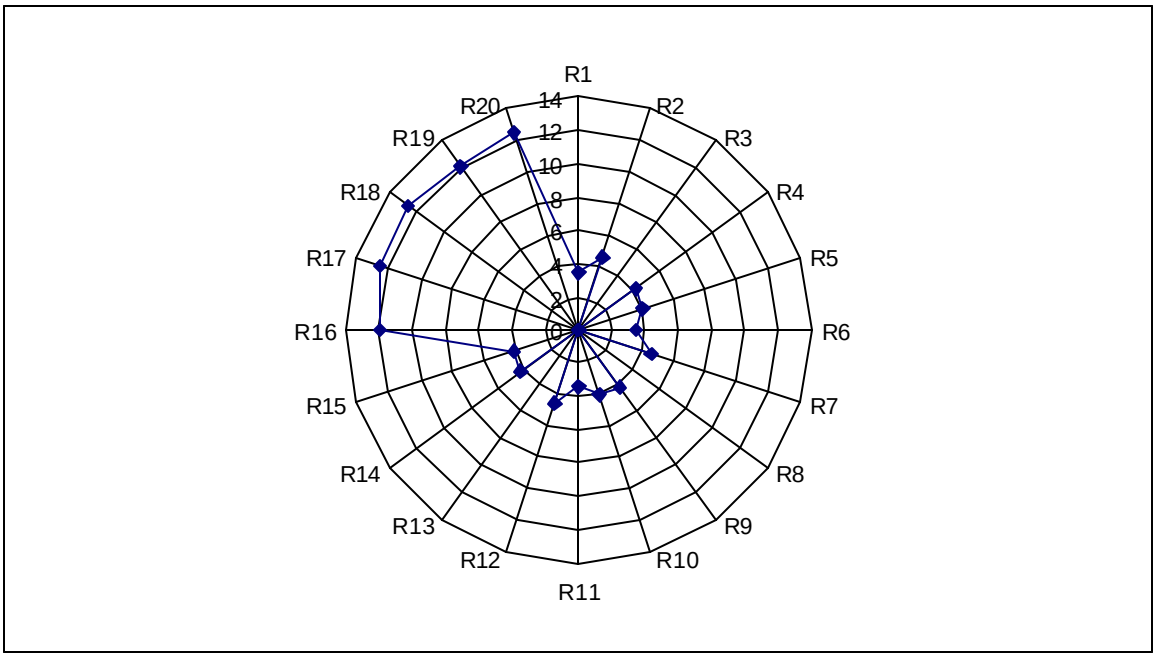


Figure 7.11: Euclidean distances of *DB* elements from *Q*3.

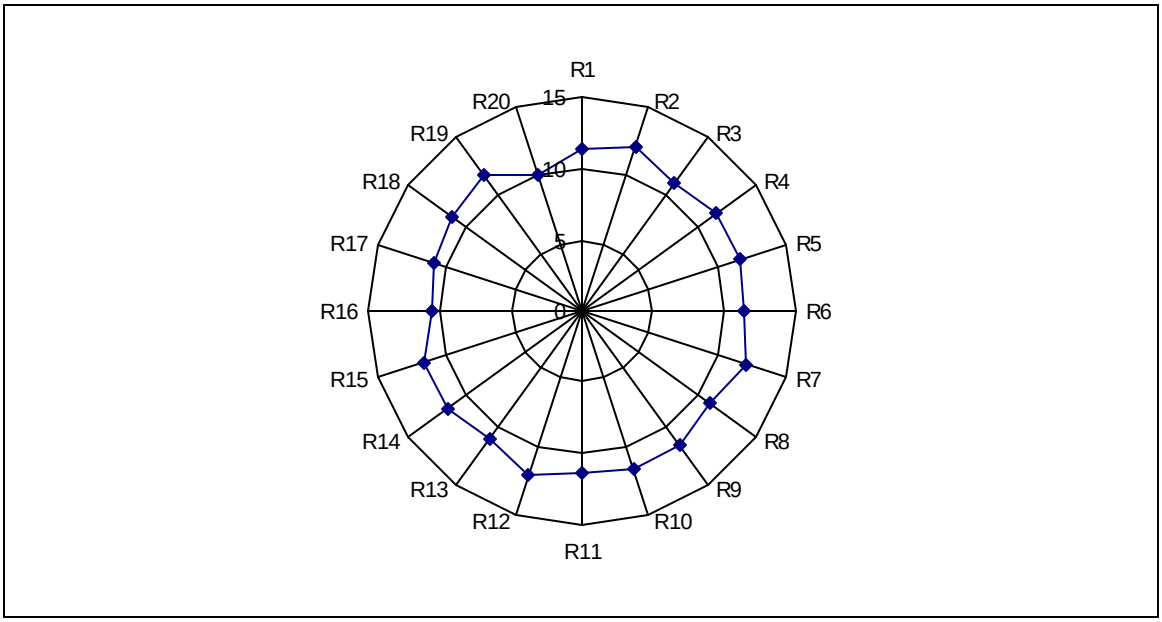


Figure 7.12: Euclidean distances of *DB* elements from *Q*4.

- The answer set of range queries of *DB* and *Q*1 with $5.0 \leq \epsilon \leq 12$ would include all scale and amplitude modified query sequences sufficiently ignoring other random sequences *R*16 through *R*20.

- The answer set of range queries of DB and $Q2$ with $4.7 \leq \epsilon \leq 10$ would include all scale and amplitude modified query sequences, safely ignoring random sequences $R16$ through $R20$.
- The answer set of range queries of DB and $Q3$ with $5.0 \leq \epsilon \leq 12$ would include all scale and amplitude modified query sequence sufficiently ignoring random sequences $R16$ through $R20$.
- Also note that the lower bound of $\epsilon \approx 4.7$ includes all time-scaled and amplitude modified sequences and this remains constant for different query sequences.
- In Figure 7.12, when query sequence is itself random walk, there is no sufficient value of ϵ that well separates scaled and amplitude-modified sequences.

The above observations indicate that there exists a range of values of threshold ϵ, which sufficiently separates a scaled,modified and noise added similar sequences from the other random sequences in the database. A range query of this threshold would provide all sequences that are similar but are either scaled or amplitude modified.

7.3 Our Proposal versus Previous work: A Case Study

In [2] authors propose using first few Fourier coefficients as index for the sequences (of equal sizes) within a database. Based on our framework, we propose the following new approach to index a special case database of equal-sized sequences.

Given a database of N sequences each of size n and a query sequence, also of size n, the proposed *DOLS*-index (*D*-index) works as follows. *DOLS*-scale all the sequences in database and the query sequence by a constant TSM factor a. For the compressed set, which we denote by $\sqsubseteq(DB)$, we take the discrete Fourier transform[1] and select first few (K) Fourier coefficients. We indicate the resultant set as $\sqsubset_K(\sqsubseteq(DB))$.The sequences are then in indexed based on these first K Fourier coefficients.

[1] We recommend scaling to a size, which is a power of 2. Remember, for sequences of size power of 2, there exists a high-speed radix-2 algorithm to compute DFT. This is the fastest available algorithm for computing DFT and is shown to have 40% increases in speed as compared to a normal case (see [23] for an elaborate discussion).

For a range query with a threshold size of ϵ, the search is performed as follows. *DOLS*-scale the query sequence and threshold ϵ by the same TSM factor a. Compute DFT of the sequences and select first *K* Fourier coefficients of the query sequence. Find all sequences in $F_K(F(DB))$ that are within the (Euclidean) distance of $(a * \epsilon)$ from $F_K(F(Q))$. For all candidate sequences reported by above, compute their actual sequence distances and discard false alarms. We call this indexing approach *D*-index.

We compared the performance of above index with '*F*-index' proposed by authors in [2]. The experiments were performed on a dedicated Intel Celeron-466 MHz machine with 128MB memory. The experiments were performed on sequences extracted from random walk. Execution time includes both index search time and post-processing (discarding false alarms) time. Specifically, we investigated the following.

1. How does performance relate with the increase in Fourier Coefficients (*K*)?
2. How does performance relate with the change in length of sequences (*n*)?

Table 7.3 summarizes the parameters employed for the experiment. All experiments were performed using 200 sequences in the database.

Table 7.3: Summary of experimental parameters.

PARAMETER	SYMBOL	VALUES	DEFAULT VALUE
# of Fourier coefficients	*K*	2, 4, 7, 10, 15	7
Length of each sequence	*n*	100,200,600,900,1200	600

Figure 7.13 shows the execution time of *D*-index versus *F*-index for increasing values of *K*. Number of false dismissals[2] are also reported. Figure 7.14 shows the execution time with increasing sequence lengths for constant number of Fourier coefficients. Percentages of false dismissals versus increasing sequence length are plotted in Figure 7.15.

[2] False dismissals refer to records that were supposed to be a part of the answer set (determined by sequential search) but are not reported.

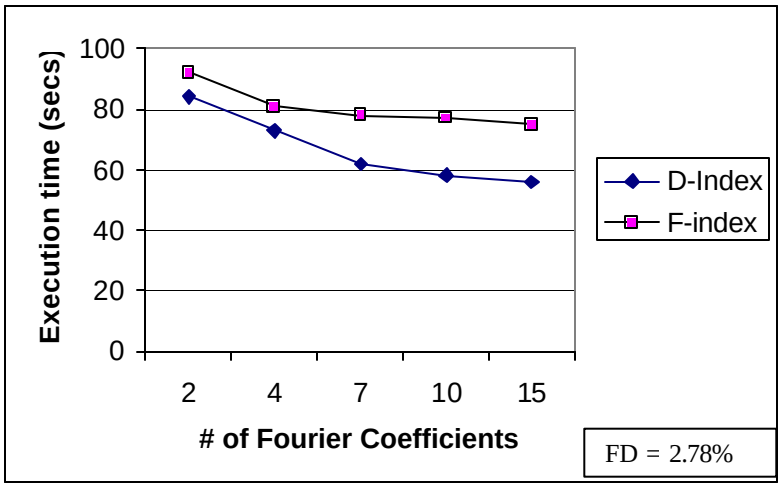


Figure 7.13: Time per query versus number of Fourier coefficients.

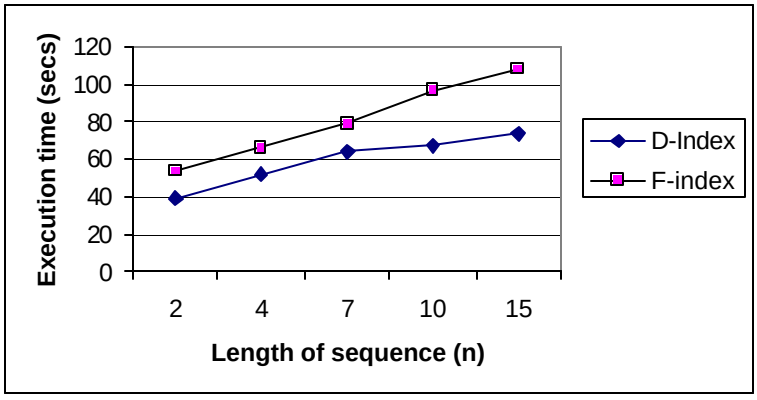


Figure 7.14: Time per query versus length of each sequence.

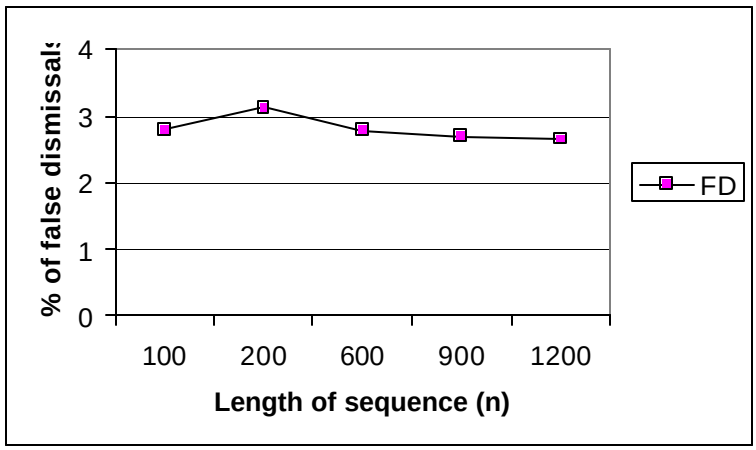


Figure 7.15: Percentage of false dismissals versus length of sequence.

Our approach depicted about 27% increase (K= 7, n = 600) in speed of query execution over F-index. The gain of our method increased with the increasing sequence size (for constant K=7). The gain also tends to increase with the increase in number of Fourier coefficients (for constant n=600). The numbers of false dismissals tend to remain flat with the increase in sequence length.

7.4 Subsequence Matching Using *FG*-Index

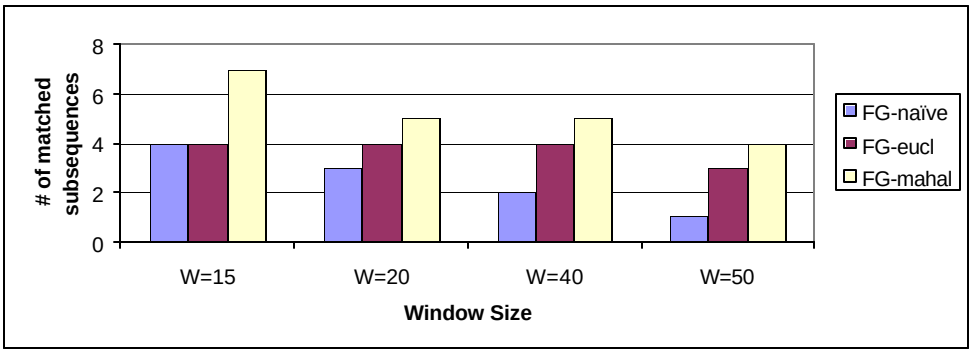
In this experimental setup we compared the performance of proposed methods for feature aggregation in Fourier space subsequently employed for recognizing similar sub-sequences within a pair of reference and query sequence.

In the experimental setup we used random walk sequence and randomly segmented it in 20 subsequences of different sizes between 10 and 90. We then extracted 2 sequences (reference and query) from different locations within random walk sequence and inserted the above extracted 20 subsequences at arbitrary locations within reference and query sequence. The locations at which subsequences are inserted in query and reference sequence are recorded.

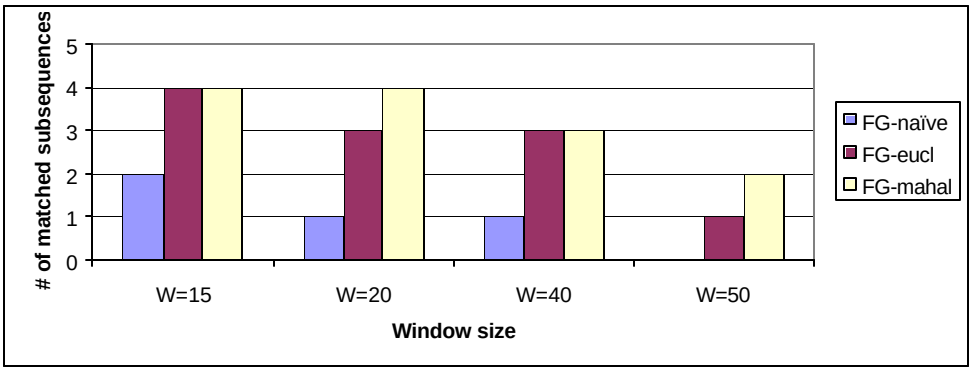
In computing results, an error threshold of 4.5 was used to define similarity (the subsequences were declared similar if their distance was less than or equal to 4.5) and discovered subsequences were also declared detected if they lied within $\pm 2\%$ (of size of subsequence) from their original embedded location.

We used three proposed methods: FG_{naive}, $FG_{Euclidean}$ and FG_{mahal}, to identify groups within these sequences. These groups are compared between reference and query sequence and similar subsequences described by these groups reported. No scaling was allowed.

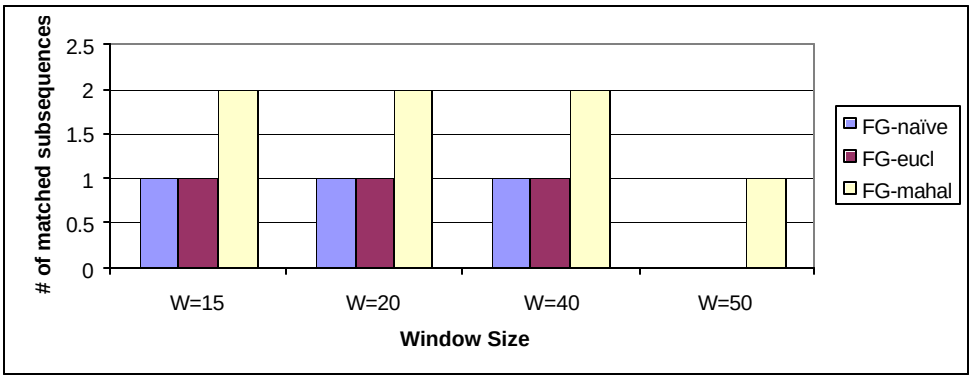
We repeated the exercise at next higher level of reference *FG*-tree until there are no new similar groups to report or *FG*-tree reaches its root. The exercise was repeated for different values of window size. The numbers of matched subsequences with the change in window size for different levels of *FG*-tree are shown in Figure 7.16(a) – (e). Figure 7.16(f) shows the total # of discovered similar subsequences. Figure 7.17 shows the total number of false alarms for three methods at the designated window sizes.



(a) # of matched subsequences at level-1 of *FG*-tree.

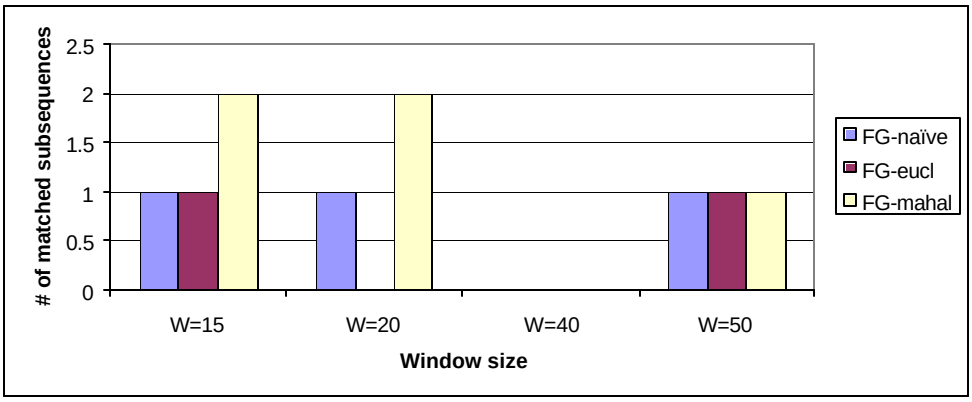


(b) # of matched subsequences at level-2 of *FG*-tree

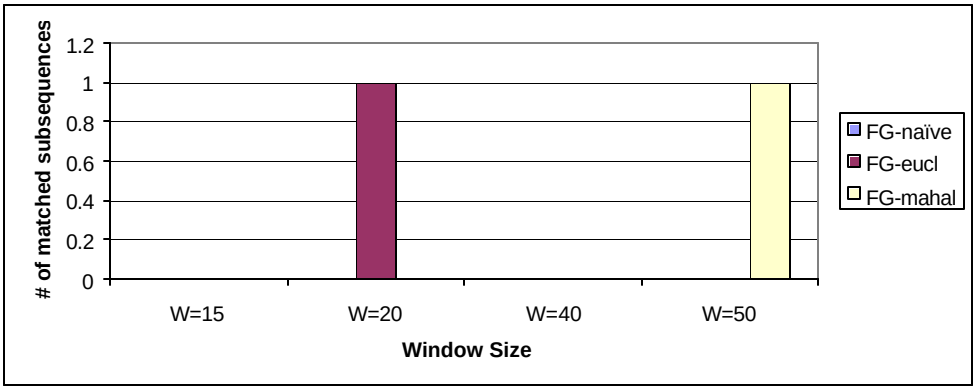


(c) # of matched subsequences at level-3 of *FG*-tree

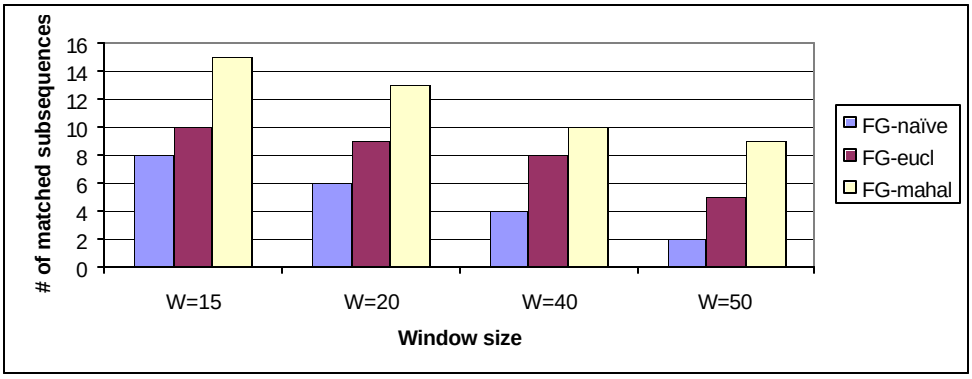
Figure 7.16: (a) – (e): # of matched subsequences at different levels of FG-tree versus window size for FG_{naive}, $FG_{Euclidean}$ and FG_{mahal} techniques employed for random walk; (f): Total # of matched subsequences versus window size for different methods.



(d) # of matched subsequences at level-4 of *FG*-tree.



(e) # of matched subsequences at level-5 of *FG*-tree.



(f)Total # of matched subsequences.

Figure 7.16(cont.)

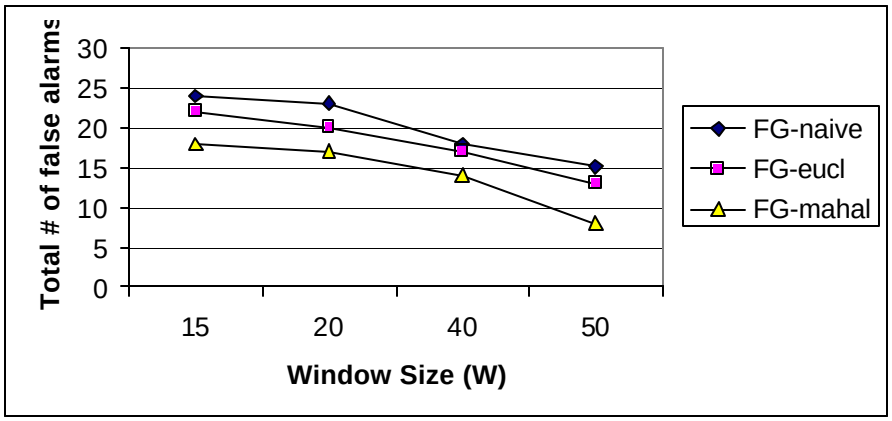
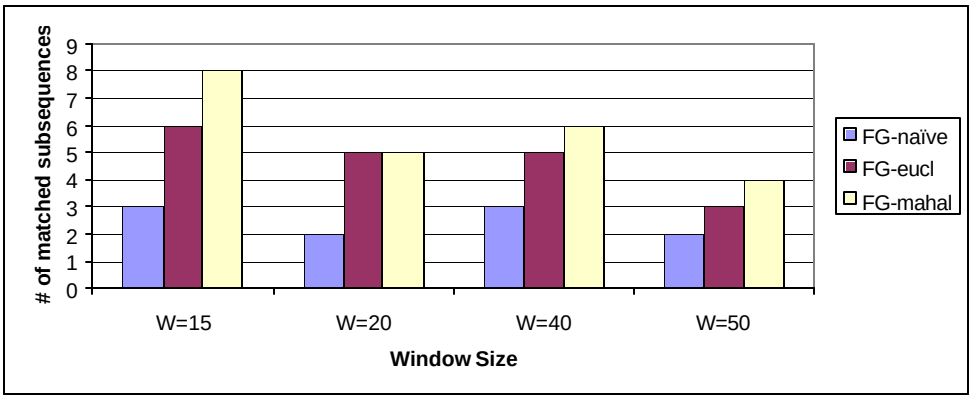


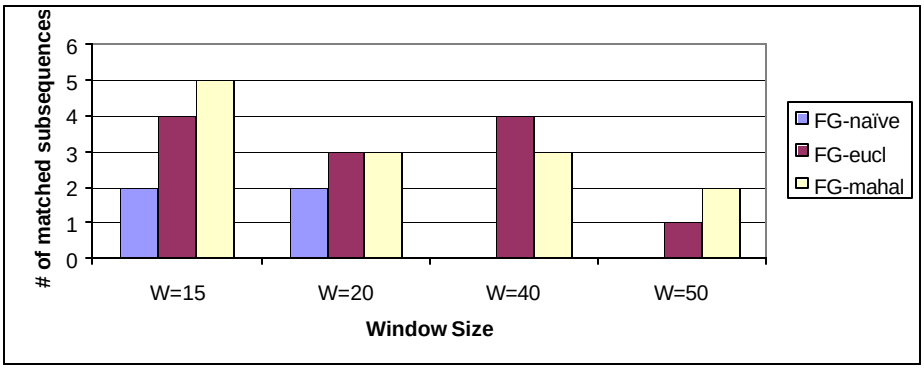
Figure 7.17: Total # of false alarms versus window size for FG_{naive}, $FG_{Euclidean}$ and FG_{mahal} techniques employed for random walk.

We repeated the exercise, employing the same experimental setup for real exchange rate data. Number of matched subsequences versus window size for methods FG_{naive}, $FG_{Euclidean}$ and FG_{mahal} are plotted in Figure 7.18. Figure 7.17 shows the total number of false alarms for three methods at the designated window sizes.

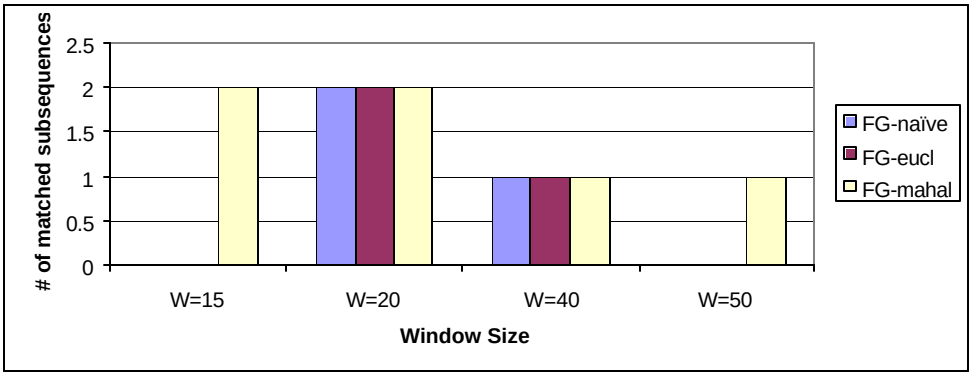


(a) # of matched subsequences at level 1 of reference *FG*-tree.

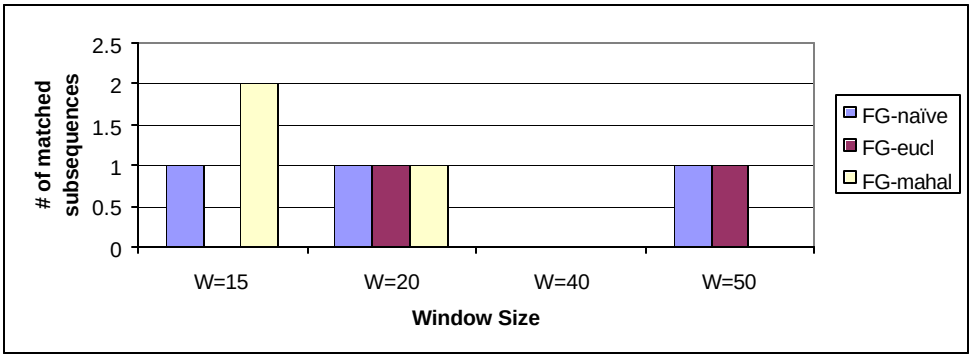
Figure 7.18: (a) – (d): # of matched subsequences at different levels of FG-tree versus window size for FG_{naive}, $FG_{Euclidean}$ and FG_{mahal} techniques; (e): Total # of matched subsequences versus window size for different methods.



(b) # of matched subsequences at level 2 of reference *FG*-tree.

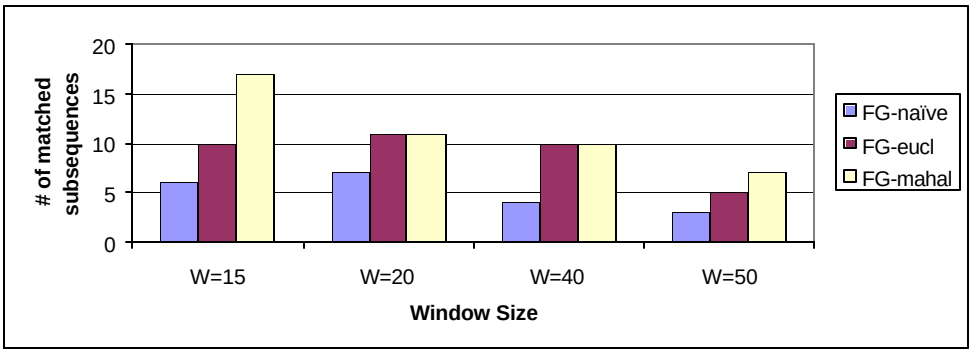


(c) # of matched subsequences at level 3 of reference *FG*-tree.



(d) # of matched subsequences at level 4 of reference *FG*-tree.

Figure 7.18 (Cont.)



(e) Total # of matched subsequences in reference and query sequence

Figure 7.18 (Cont.)

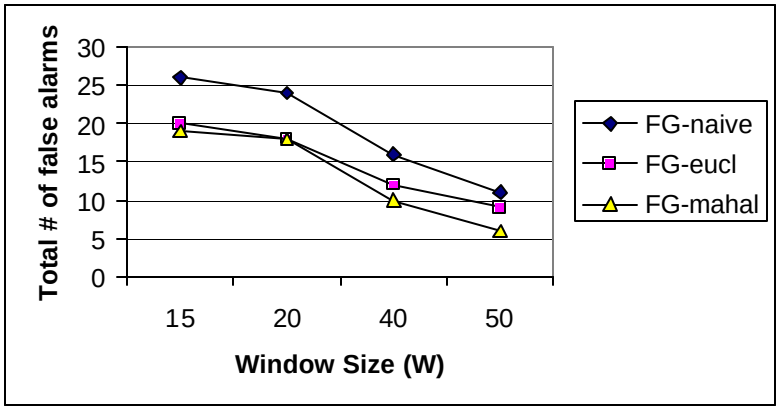


Figure 7.19: Total # of false alarms versus window size for FG_{naive}, $FG_{Euclidean}$ and FG_{mahal} techniques employed for real exchange rate data.

Feature aggregation based on Mahalanobis distance outperformed other methods in detecting total number of similar objects (15/20 for random walk and 17/20 for exchange rate data). 5 subsequences in random walk and 3 subsequences in exchange rate data remained undetected. For both random walk sequences and real exchange rate data, aggregation based on Mahalanobis distance exhibited better performance in terms of number of false alarms, compared to other techniques. Generally, total number of false alarms decreased with increase in window size.

Aggregation based on fixed size segmentation of feature trail exhibited poor detection, as expected. In majority of the cases, increase in window size resulted in decrease in number of objects discovered (the subsequences of sizes less than window size would remain undiscovered).

Chapter 8

Conclusions and Future Directions

This chapter presents the conclusions, which have been drawn from the preceding research. Both problems, which have been approached, are of practical interest with many immediate applications, and are both computationally challenging. We begin our discussion with summarizing our contributions and then proceed to present potential applications for the same. We conclude with some directions for future research.

8.1 Merits and Contributions

8.1.1 Approach to Answer Similarity Queries in Unequal-Sized Sequence Databases

We have presented a unique time scale reduction algorithm called dynamic overlay and summation technique that reduces time-scale by compressing redundancy in sequential data. We also discussed the advantages of using normal forms of sequences for similarity search. We subsequently employed the framework to answer range query for a database of unequally sized sequences, which can better be understood for many applications.

Our experimental results have illustrated that relative error between the scaled harmonic behavior of the original sequence and that of the corresponding contracted sequence is very small. We consequently proposed an indexing scheme to index a database of sequences of unequal sizes and presented a framework to pose range query on this database. This index integrates our approach with existing method of indexing and can add and retrieve records (sequences) to the index without re-building it. This enhances the capability of the existing index to facilitate queries of unequally sized sequences without modifying the existing retrieval mechanism.

We also proposed a novel indexing technique for equally sized sequences (a special case scenario), which was experimentally shown to achieve better performance in terms of query processing time on both synthetic and real data.

8.1.2: Approach to Answer *Similar-pairs* Query in Sequences

We presented an approach to find pairs of similar unequally sized subsequences in reference and query sequence. This object matching paradigm put forward has several advantages, of which, most important are enumerated below.

- This method can detect embedded subsequences when their positions are previously unknown.
- The number of locations in which matching is done depends on the number of objects discovered in a sequence and not on its size.
- This method identifies the objects in the reference sequences that are likely to find a match, thus making the selection deterministic and efficient. This is particularly important because the sequences are very large and an efficient method for narrowing down the search space significantly reduces the computational demand of the search query.
- In a likely event, as in many applications, data is added to a reference sequence after the index is built; only last group in the existing sequence is used to integrate new objects without having to re-build the whole index.
- It is interesting to note that it is possible to extend our method to make it rotational invariant to the pose of object. Recall, *FG*-index indexes objects first on their orientation angles. If we skip this step and immediately index by their aspect ratios (that is, each class C^i contains object that have distinct aspect ratios), all objects retrieved by querying this object index would be invariant to rotation with respect to each other. This is specifically of interest in the problem of object recognition in images (see Section 8.2).
- By finding *harmonically stationary* objects in feature space and using the theory of employing first few Fourier coefficients, we essentially reduce the problem of similarity search within windows of size w to searching in $2N_c$ space ($N_c << w$). This can positively affect the speed of building and searching the index.

8.2 Applications

The approaches outlined here are shown to be specifically developed for similarity search in the area of data mining. It is interesting to note that potential applications of these can be extended to other practical areas, as discussed below.

- The time-scale reduction algorithm presented here can be employed for any data compression task in hand, in which it is important to preserve the relative harmonic behavior of the data.
    - For example, in the area of multi-sensor fusion for speed-critical environments, sensor readings from remote sensor stations are regularly transmitted to ground receivers for analysis and command. The delay in transmission contributes maximum to total processing time of data, which is further proportional to the size of data transmitted. An important attribute is the harmonic variation of the data stream, which is studied on ground stations for anomaly detection (also see [7]). The data can be time-scaled on sensor station to an appropriate size for transmission and TSM factor integrated with the header information. This can reduce the transmission time significantly, thus increasing the turnaround time of the ground station for time-critical tasks.
    - Another potential application in this category is to utilize the proposed technique to store large sequences in memory-critical environment. For example, miniature mobile agents distributed over war field record information, which might be redundant but occupy space in their limited memory resource. Enhancing the mobile agent's capability to time-scale these data (preferably in an online mode) and then record data can enhance the storage capacity of these agents.
- Another potential application of dynamic overlay and summation technique is signal smoothening. If the application demands no time-scale modification then the resultant version of the signal would be a smoothened form of the original signal with suppressed short-term fluctuations (which are presumed as noise for many signals).
- The paradigm presented to handle *similar-pairs* query in sequences can be employed for object recognition in images. Given a query image, the problem of image

recognition demands to check the presence or absence of the query image in a set of images. A skeleton solution based on our approach is straightforward: For all images the set of images, compute their convex hulls and consequently the attribute vectors describing their MERs. Index them using the proposed *FG*-index. Given a query image object, extract its convex hull and corresponding attribute vector (of its MER). Search the *FG*-index for similar pose following procedure outlined in Chapter 6. Once the pose-similar objects are identified, scale the recovered objects appropriately and compute normalized crosscorrelation between each of them and the query object. Report all matches that have crosscorrelation values less than a specified threshold. Following the recommendation presented in Section 8.1 above, it is possible to make this procedure invariant to rotation of the image object.

8.3 Future Directions

The approaches presented in this dissertation open exciting avenues of further investigation including but not limited to the following.

We have restricted our algorithms to real-numbered time series data. The ideas could be further extended to discrete domains like alphabets, as in representation of protein sequences. The problem of searching similar motif[1] pairs can then be reduced to problem of searching similar pairs in these sequences.

The results of similarity search can be further used for mining other rules, such as association, classification and prediction. For example, we may find that a steep jump in the number of graduate school applications always associates with a steep decline in stock market index. This is an association rule. Similarly other rules can also be discovered and open a good research direction.

[1] For example, similar motifs in protein sequences tend to have similar secondary structures, hence similar functions. This is specifically important in drug design applications. Also see [29].

References

[1] Agrawal R., T. Imielinski and A. Swami. Database Mining: A Performance Perspective. *IEEE Trans. on Knowledge and Data Engineering*, 5(6): 914-925, 1993.

[2] Agrawal Rakesh, Christos Faloutsos and Arun Swami. Efficient similarity search in sequence databases. *Proc. of FODO Conference*, Evanston, Illinois, Oct 13-15, 1993.

[3] Agrawal Rakesh, Tomasz Imielinski and Arun Swami. Mining association rules between sets of items in large databases. *Proc. of ACM SIGMOD*, pages 207-216, May 1993.

[4] Altschul S.F., Gish W., Miller W., Myers E.W. and Lipman D.J., Basic local alignment search tool. *Journal of Molecular Biology*, 215(3): 403-410, Oct. 1990.

[5] Baeza-Yates Ricardo and Gonnet Gaston H., A New Approach to Text Searching., *Communications of ACM*, 35 10, 74-82, Oct. 1992.

[6] Bansal Kanthi, Vadhavkar S. and Gupta A., Neural network based forecasting techniques for inventory control applications, *Data mining and knowledge discovery* 2, 97-102 (1998).

[7] Bracewell Ronald N., *The Fourier Transform and its applications*. Mcgraw Hill, Third edition.

[8] Brooks Richards R., Iyengar S.S., *Multi-Sensor Fusion: Fundamentals and Applications with Software*. Simon & Schuster Trade, 1997.

[9] Brown L.G., A Survey of Image Registration Techniques. *ACM Computing Surveys*, 24(4), pages 325-376, December 1992.

[10] Chatfield C., *The Analysis of Time Series: an Introduction*. Chapman and Hall, London & New York, Third Edition.

[11] Debregeas A. and Hebrail, G. (1998). Interactive interpretation of Kohonen maps applied to curves. *Proceedings of the 4th International Conference of Knowledge Discovery and Data Mining*. pp 179-183, August 27-31, 1998.

[12] Dua Sumeet, Dynamic and implicit profiling of consumer patterns for efficient customer branding in e-commerce; *Systems Science Masters Thesis, Louisiana State University*, May 2000[*].

[*] References [12],[13],[14] and [39] are not cited.

[13] Dua Sumeet, Iyengar S.S., Bharatheesh T.L., A new method for information discovery in plant databases. *International journal of information science*. August 2000*.

[14] Dua Sumeet, Iyengar S.S., Cho Eungchun, Discovery of web frequent patterns and user characteristics from web access logs: A framework for dynamic web personalization. *Proceedings of 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology* (*ASSET*), 24-25 March 2000*.

[15] Edwards Robert D. and Magee John, *Technical analysis of stock trends*. John Magee, Springfield, Massachusetts, 1966. 5th edition, second printing.

[16] Faloutsos C., Ranganathan M., and Manolopoulos Y., Fast subsequence matching in time-series databases. *Proc. of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (PODS '94), pages 4-13, Minneapolis, May 1994.

[17] Fayyad Usama M., Piatesky-Shapiro Gregory, Smyth Padhraic, and Uthurusamy Ramasamy, *Advances in Knowledge Discovery and Data Mining*. The MIT Press, 1996.

[18] Freeman H. and Shapira R., Determining the minimum-area enclosing rectangle for an arbitrary closed curve. *Communications of the ACM*, 18(7):409--413, July 1975.

[19] Gaede V.,Gunther O., Multidimensional access methods, In *ACM Computing Surveys*, Volume 30, Number 2, June 1998.

[20] Galil Z. and Park K. An improved algorithm for string matching. *SIAM Journal on Computing*, 19:989-999, 1990.

[21] Gelb A., *Applied Optimal Estimation*. MIT Press, 1986.

[22] Goldin D. Q. and Kanellakis P.C., On similarity queries for time-series data: constraint specification and implementation. *Proc. of 1st Intl. Conference on the Principles and Practice of Constraint Programming*, pages 137-153. LNCS 976, Sept. 1995.

[23] Goodman Nelson, Seven structures on similarity. In Nelson Goodman, editor, *Problems and Projects*, pages 437-447. Bobbs-Merril, New York, 1972.

[24] Graham, R.L., An efficient algorithm for determining the convex hull of a finite planar set, *Information Processing Letters*, 1, 132-133, 1972.

[25] Grossi R. and Luccio F. Simple and efficient string matching with k mismatches. *Information Processing Letters*, 33:113-120, 1989.

[26] Hales Lisa and Hallgren Sean, An Improved Quantum Fourier Transform Algorithm and Applications. In proceedings of *41st annual symposium on foundations of Computer science (FOCS)*, November 2000, Redondo Beach, California.

[27] Hayakawa M. and Y. Fujinawa (Eds.), Electromagnetic Phenomena Related to Earthquake Prediction, Tokyo: Terra Scientific Publishing Co., 1994. 677 pp. in *Proceedings of International Workshop*, 6-8 September 1993, The University of Electro-Communications, Chofu, Tokyo, Japan.

[28] He Liwei & Gupta Anoop, User benefits of Non-Linear time compression. *Microsoft Technical report MSR-TR-2000-96*, Sept. 2000.

[29] Jokinen P., J. Tarhio and E. Ukkonen. A comparison of approximate string matching algorithms, *Software-Practice and Experience*, 26:1439-1458, December 1996.

[30] Keogh E., and Pazzani M. (1998). An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. *Proceedings of the 4hInternational Conference of Knowledge Discovery and Data Mining*. pp 239-241, AAAI Press.

[31] Landau G.M. and Vishkin U., Fast and parallel and serial approximate string matching. *Journal of Algorithms*, 10:157--169, 1989.

[32] Lee, S. & Kim, H. Variable time-scale modification of speech using transient information. *IEEE international conference on acoustics, speech, and signal processing*, Vol. 2, pp 1319-1322, Munich, 1997.

[33] Li Zhi, Dua Sumeet, Iyengar S.S. *et al.*, A grouping algorithm to group similar motifs in protein data banks. *Submitted for publication*.

[34] Lipman D.J. and Pearson W.R., Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435-1441, March 1985.

[35] Mahalanobis P. C., On the generalized distance in statistics. *Proceedings of the Indian National Institute of Science*, 2:49 – 55.

[36] Mandelbrot B., *Fractal geometry of Nature*. W.H. Freeman, New York, 1977.

[37] Motro A., VAGUE: A user interface to relational databases that permits vague queries. *ACM Trans. on Information Systems* (*TOIS*), 6(3), pages 187{214, July 1988.

[38] Ng, M. K., Huang, Z., & Hegland, M. (1998). Data-mining massive time series astronomical data sets - a case study. *Proc. of the 2nd Pacific-Asia Conference on Knowledge Discovery and Data Mining*. pp 401-402

[39] Niblack Wayne, Ron Barber, Will Equitz, Myron Flickner, Eduardo Glasman, Dragutin Petkovic, Peter Yanker, Christos Faloutos, and Gabriel Taubin. The QBIC Project: Querying images by content using color, texture and shape. In *SPIE 1993 Intl. Symposium on Electronic Imaging: Science and Technology, Conf. 1908, Storage and Retrieval for Image and Video Databases*, Feb. 1993*.

[40] Ozaktas Haldun M., Zalevsky Zeev, Kutay M. Alper, The Fractional Fourier Transform: with Applications in Optics and Signal Processing, *John Wiley and Sons*, 2001.

[41] Pearson W.R. and Lipman D.J., Improved tools for biological sequence comparison. *Proceedings of the National Academy of Science*, 85(8):2444-2448, 1998.

[42] Pfeiffer S., Effelsberg W. et al., Automatic audio content analysis. *Technical Report TR-96-008*, University of Mannheim, April 1996.

[43] Povinelli, R. (2000). Identifying temporal patterns for characterization and prediction of financial time series events. In Proc. *International Workshop on Temporal, Spatial and Spatial-Temporal Data Mining, TSDM2000*, Lyon, France. Lecture Notes in Artificial Intelligence, 2007. Roddick, J. F. and Hornsby, K., Eds., Springer.

[44] Quereshi, S.U.H. Speech compression by computer. In S. *Duker (Ed.), Time-Compressed Speech*, 618-623. Scarecrow, 1974.

[45] Richards John A., *Remote sensing digital image analysis: An introduction*. Springer-Verlag.

[46] Roth William G., MIMSY: A system for analyzing time series data in the stock market domain. *Master Thesis. University of Wisconsin*, Madison, 1993.

[47] Roucos, S. & Wilgus, A. High quality time-scale modification of speech. *IEEE International conference on acoustics, speech and signal processing*, Vol 2,pp 93-496, Tampa, FL, 1985.

[48] Salton G. and M.J. McGill, *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[49] Schroeder M., Fractals, Chaos, *Power laws: minutes from an infinite paradise*. W.H. Freeman and Company, New York, 1991.

[50] Seshadri P., Livny M., and Ramakrishnan R., Sequence query processing. *Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD '94)*, pages 430-441, Minneapolis, May 1994.

[51] Shasha D. and Wang T.L., New techniques for best-match retrieval, *ACM TOIS*, 8(2):140 -158, April 1990.

[52] Shepard Roger N., Toward a universal law of generalization for psychological science. *Science*, 237:1317-1323, 1987.

[53] Toussaint G.T., Solving Geometric problems with the 'Rotating Calipers'. *In Proc. IEEE MELECON*, Athens, Greece, 1983.

[54] Tversky Amos, Features of similarity. *Psychological Review*, 84:327-352, 1977.

Vita

Sumeet Dua was born in New Delhi, India, in 1975 and is son of V. J. Dua and Kanta Dua. He obtained his schooling from Saint Francis De Sales School, New Delhi. In 1993 Sumeet began his university education at Thapar Institute of Engineering and Technology (Deemed University) and obtained a bachelor's degree of engineering in electronics and communication in 1997. He then attended Louisiana State University, Baton Rouge, Louisiana and obtained the degree of Master of Science in Systems Science from the Department of Computer Science in 2000. He then continued with his doctoral studies in the same department and will graduate in May 2002 with the degree of Doctor of Philosophy in computer science. He is currently a researcher with Biological Computing and Visualization Center, Department of Ophthalmology, LSU Medical Services Center, New Orleans. His areas of interest include knowledge discovery in large databases, bioinformatics, parallel algorithm design and e-commerce.