Similarity Search in Time Series Data Sets

by

Betty Bin Xia
M.S., Jilin University, 1993
B.S., Jilin University, 1990

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
December 1997

0-612-24275-7

APPROVAL

Name:	Betty Bin Xia
Degree:	Master of Science
Title of thesis:	Similarity Search in Time Series Data Sets

Examining Committee: Dr. Arvind Gupta
Chair

Dr. Jiawei Han
Senior Supervisor

Dr. Qiang Yang
Supervisor

Dr. Veronica Dahl
External Examiner

Date Approved:

Abstract

Similarity search on time-series data sets is of growing importance in data mining. With the increasing amount of data of time-series in many applications, from financial to scientific, it is important to study the methods of retrieving similarity patterns efficiently and user friendly for business decision making.

The thesis proposes methods of efficient retrieval of all objects in the time-series database with a shape similar to a search template. The search template can be either a shape or a sequence of data. Two search modules, subsequence search and whole sequence search, are designed and implemented.

We study a set of linear transformations that can be used as the basis for similarity queries on time-series data, and design an innovative representation technique which abstracts the shape notion so that the user can interactively query and answer the multi-level similarity patterns. The wavelet analysis technique and the OLAP technique used in knowledge discovery and data warehousing are applied in our system. The retrieval technique we propose is efficient and robust in the presence of noise, and can handle several different notions of similarity including changes in scale and shift.

Keywords: time-related database, data warehouse, data mining, and wavelet analysis.

Acknowledgments

I would like to particularly thank my supervisor, Dr. Jiawei Han for getting me interested in the area of data mining and data warehousing, and his consistent and sound critiques of my ideas and work. I am particularly grateful for his enthusiasm, constant support, and encouragement. His enthusiasm for work will always be a source of inspiration in my future career.

I am grateful to Dr. Qiang Yang for his great help throughout my study and serving on my supervisory committee. I also want to thank Dr. Veronica Dahl for graciously consenting to be my external examiner and Dr. Arvind Gupta for serving as the Chairman of my defense.

I must thank my DBMiner friends, those we worked together day and night and gave me a lot of technical help. Special thanks go to Sonny Chee, Qing Chen, Shan Cheng, Jenny Chiang, Wan Gong, Micheline Kamber, Kris Koperski, Yijun Lu, Nebojsa Stefanovic, Osmar Zaiane, Shuhua Zhang, Hua Zhu, Jie Wei, Hongsheng Qin, for their valuable suggestions and help in these two years of study as well as their friendship. Thanks to all the other friends I have made at Simon Fraser University for making my stay at SFU an enjoyable period of time.

I will always indebted to my family, especially my parents Lixian Xia and Yulu Liu, my husband Wei Tian, and my brother Quan Xia. I would like to thank them for their love. This thesis would not have been possible without all their support and confidence in me.

Dedication

To my dear parents and my husband, for their love.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The research in this thesis grows out from the development of data mining techniques. With the rapid development of automated data collection tools, database systems, and data warehouse technologies, the scientific community has shown a rapidly growing interest in the discovery of hidden information in databases and data warehouses, also known as `Data Mining`. Many enterprises need to store and analyze sequences of time-stamped data, also called time-series data. It is very important to develop techniques to mine the useful patterns or knowledge from it. A time series can be defined as " a sequence of real numbers, each number representing a value at a time point" [3, 2] . Time-series data sets are of growing importance in many new database applications, such as data mining or data warehouse. We are especially interested in mining similarity patterns in large sets of time-series data. In this chapter, we give a brief introduction of the notions of time-series data sets and the data mining and data warehouse technique which our system is based on, and then address the motivation of similarity search.

1.1 Data Mining and Data Warehousing Technique

Data warehousing is a collection of *decision support* technologies, which is to help people make better and faster decisions. We have seen rapid growth of data warehouse systems in the past few years. Data warehousing has been successfully adopted by many industries: manufacturing, retail, financial services, transportation, telecommunications and medical.

A data warehouse is a "subject-oriented, integrated, time-varying, non-volatile collection of data that is used primarily in organizational decision making" [9]. The data warehouse supports on-line analytical processing (OLAP), and it is targeted for decision support. Data warehouse contains historical, summarized and consolidated data, which can be from several operational databases, over potentially long periods of time. Thus the size of the data warehouse tends to be hundreds of gigabytes to terabytes, and the workloads depend mostly on the ad hoc, complex queries that require accessing millions of records and performing a lot of scans, joins, and aggregates.

As its name suggests, a data warehouse acts as a central storage area (a "warehouse") for the data. It is also a data cleanser and a data organizer for easy and intelligible access of data. Whereas, data warehousing is a process of construction and utilization of data warehouse.

Data mining tool is one of the front-end tools of the data warehouse to do querying and data analysis.

Mining information and knowledge from large databases is not a new idea and has been recognized by many industrial companies as an important area with an opportunity of business success.

With the explosive growth of data in databases, and these databases contain treasures of information that allows the company to detect trends or patterns and react flexibly to them. However this information is hidden in the mountains of data, and cannot be discovered using conventional database management systems. The solution is **data mining**. It has become a research area with increasing importance [28, 33, 37]. Data mining is a technique to reveal the *strategic information* hidden in large databases. This information, e.g. trends and patterns, can be used to improve business decision making.

Data mining, which is also referred to as *knowledge discovery in databases*, means "a process of nontrivial extraction of implicit, previously unknown and potentially useful information (such as knowledge rules, constraints, regularities) from data in databases" [33].

Data mining is very useful for the company managers, such as helping the company to find and reach better customer, gaining critical business insight to help raising profits, etc.

1.2 Motivations for Similarity Mining in Time-Series Data Sets

In the last few years, storing and analyzing sequences of time-stamped data are becoming more and more principal in many enterprises. For example, financial firms associate a stock's high, low, closing price, and volume with a given day, and intelligence agencies associate complex satellite data with the time the data was collected. Time-related data is used by many other kinds of enterprises as well, including manufacturing (assembly line events), journalism (news dispatches), earth science (seismic events), and engineering (code changes).

In the Information Age, knowledge is power, and information is the key to profits, so the ability to manage *time-series* data accurately, efficiently, and flexibly is vital to business success. To mine similarity patterns from large time-series data sets is especially important.

From the following examples, we study the important applications of the similarity search in time-series data sets.

Example 1.2.1 Suppose there are about 10 years of data about the daily closing price of IBM company and Microsoft company. A user may want to find if they behaved in approximately the same way in all the times, or during the years of 1985 to 1987 period, or even during the months of July 1988 to November 1988. A user may dynamically drill-down or roll-up along the time dimension to explore the desired information.

Example 1.2.2 Given two time sequences of the temperature in two regions of the world, one may want to find years when the temperature patterns in two regions of the world were similar.

Example 1.2.3 Given a bunch of time sequences of the stocks for the companies, we may want to find other companies whose stock price fluctuations resemble Microsoft's during a year or at all the times.

These are the cases that people want to compare two or more time-related data sequences, but there are also the cases that the users want to analysis the trend for one time sequence data, such as the following example.

Example 1.2.4 For one company, a user may want to find that its stock price increased sharply up to September 1980, and then crashed sharply. One may likely to perform data analysis on different resolutions of shapes, such as the overall shape or the detail shape.

In these examples, the approximate matching is usually more useful than exact matching, since it will be very rare to have exact match for two data sets.

The above examples indicate some interesting applications of the similarity search in time-series data sets. To implement them successfully, there are some challenges as follows:

- **What is similarity.** The meaning of similarity may vary depending on the application domain and even the purpose of the query. Work in this area is usually specific to one particular domain and uses one specific notion of similarity. Such as in the 2-dimension space polygons searching, or in the multimedia databases image searching, or in the text-processing system the text string searching. The research domain in this thesis is in databases with time-series of real numbers. The syntax and semantics for similarity queries, which account for approximate matching, scaling and shifting, are given in later chapters.

- **Two kinds of similarity problems.** For our 1-dimension time-series domain, two basic problems in this area are *First(All)-Occurrence Subsequence Matching* and *First(All)-Occurrence(s) Whole Sequence Matching.* For the All-Occurrences Matching problems (either subsequence or whole-sequence), there also exist two additional categories: *approximate matching* and *exact matching* [16]. The definitions and implementations of the above problems are addressed in the following chapters.

- **Efficient retrieving.** This system is designed towards on-line interactive mining, thus efficiency is very important. Although there are many methods developed [1, 2, 14], new models and techniques should be developed to support on-line analysis of voluminous time-series data.

- **Validity and accuracy.** To minimize the occurrences of the *false dismissals*, the validity and accuracy of the matching result should be taken into consideration.

- **Friendly user interface.** To represent the querying and the result of the similarity search, a friendly user interface should be designed. That means, it should be taken into consideration of how to design a good user interface or a representational language which can directly capture the notions of sequence shapes and is intuitive for human interactively querying and answering.

- **Noisy data.** There are two kinds of noisy data: one is due to the unavoidable imprecision of measuring devices and clocking strategies; the other is the short interval gap, which means some very small regions of data have sharp jumps or valleys, but the time intervals are so small that can be ignored without influence the whole sequence trend. If we do not do preprocessing to the noisy data, they will influence the searching result. A well-known solution for comparing two sequences a and b is Euclidean distance, which is very sensitive to the short-term jumps or valleys. Other kinds of distance metric are also very sensitive to the noisy data. To make these noisy data uninfluenced the similarity search, they must be removed or some preprocessing must be done to smooth the sequence data before doing the similarity search.

- **Regular vs. irregular time-series.** Time series data varies in how predictably the data arrives. In some cases, the data is associated with a regular time interval (daily, monthly, or quarterly). In other cases, the data may be irregular, such as network management events, and assembly line trouble reports. Our similarity miner module currently supports regular time series, and will soon be extended to support irregular time series as well. In our module, *time-series* data are sequences of real numbers representing measures at uniformly-spaced temporal instances [16, 25, 35, 24].

1.3 The Goals of Similarity Miner System

The similarity miner system is one of the knowledge discovery modules of our relational data mining system, DBMiner, researched and developed in our laboratory [21]. Besides the similarity miner module, currently, the discovery modules of DBMiner include characterizer, comparator, classifier, associator, and predictor.

Similarity search is a fairly important issue in the time-related data mining area. The goal of this thesis is to implement it on the bases of the data mining cube which is developed in our lab. Cube is a data structure for efficient retrieval of the data needed.

Two similarity miner modules have been designed in this thesis. One is subsequence search module; the other is whole sequence search module. In the subsequence search module, the problem to solve is given a query shape template Q, finding the similar shape subsequences in the time-series sequence. In the whole sequence search module, the problem to solve is given a query time-series data sequence Q, and a set of time sequences with equal length as Q, finding all of the data sequences that match Q approximately.

The goals of our system are to (1) provide fast response time, (2) support interactive analysis, (3) make it flexible for user to use, allowing multi-level similarity search, e.g. from exact matching to blurry matching, (4) make the implementation independent of the kinds of the data sources, and (5) make the system robust in the presence of noise.

1.4 Thesis Organization

The organization of the rest of the thesis is stated in the following. First, a review of the previous related work on the time-related data mining and similarity search is presented in Chapter 2. Chapter 3 describes the general properties of our similarity search system in time-series domain including the discussion of the system architecture, and some notations. Chapter 4 addresses the background knowledge of the wavelet analysis technique. Chapter 5 presents the subsequence search module, and the algorithms and performance study are included. The whole sequence search module, including algorithms and performance study, is discussed in Chapter 6. Chapter 7 summarizes the thesis and discusses the future research issues.

Chapter 2

Related Work

There has been a great deal of research in similarity search area, especially in recent years. In this chapter, we will discuss the different approaches in this area. It includes techniques on the similarity search and similarity search in the data mining area.

There are many different similarity search algorithms that are determined by the kinds of searching space to deal with. 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 with time-series of real numbers is considered to be searching in continuous space.

There has been a lot of work on similarity search in discrete and continuous spaces.

2.1 One Dimension String Similarity Searching

There has been a lot of work on finding text subsequences that approximately match a given string [31, 29, 40, 44, 15, 45, 20, 43]. Text sequences normally consist of a few discrete symbols as opposed to continuous numbers that makes the similarity measures and the search methods quite different.

A classical *string searching* problem is, given two strings, *text* $T[1 \cdots i]$ and *pattern* $P[1 \cdots j]$, to determine whether the *text* T contains an occurrence of the *pattern* P as a substring, that is, whether T can be written as $T = yPy'$. An *approximate string matching problem* means a substring P' of T such that at most k editing operations (insertions,deletions, changes) are needed to convert P' to P [42, 26].

- Edit distance-based approach

  Generally they use the concept of *edit distance* [47, 41] to measure the goodness of approximate occurrences of a pattern. The *edit distance* between two strings, A and B in alphabet $\sum$, 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 $a \rightarrow \epsilon$ (a deletion), $\epsilon \rightarrow b$ (an insertion), or $a \rightarrow b$ (a change) where a, b are in $\sum$ and ϵ is the empty string. Assuming the cost of each of these operations is 1, the edit distance is the minimum number of operations needed to obtain a pattern from a text. A dynamic programming solution for this problem is given in [36, 15, 42, 26, 44]. Let D be an $m+1$ by $n+1$ table such that $D(i,j)$ is the minimum edit distance between $p_1p_2\cdots p_i$ and any substring of T ending at t_j. Then

$$D(0,j) = 0, \quad 0 \leq j \leq n$$

$$D(i,j) = min \begin{cases} D(i-1,j)+1 \\ D(i-1,j-1)+ & \text{if } p_i = t_i \text{ then } 0 \text{ else } 1 \\ D(i,j-1)+1 \end{cases}$$

The three parts of this formula corresponds to deletions, changes, and insertions respectively. Table D can be evaluated column-by-column. Whenever $D(m,j)$ is found to be at most k for some j, there is an approximate occurrence of P ending at t_j with edit distance $D(m,j) \leq k$. Hence j is a solution to the k differences problem.

Example 2.1.1 In Table 2.1 there is an example of table D for $T = bcbacbbb$ and $P = cacd$. The pattern occurs at positions 5 and 6 of the text with at most two differences.

		0	1	2	3	4	5	6	7	8
			b	c	b	a	c	b	b	b
0		0	0	0	0	0	0	0	0	0
1	c	1	1	0	1	1	0	1	1	1
2	a	2	2	1	1	1	1	1	2	2
3	c	3	3	2	2	2	1	2	2	3
4	d	4	4	3	3	3	2	2	3	3

Table 2.1: Table D: an edit distance-based approach

All the algorithms presented in [47, 41, 15, 45, 20, 43] work within this model, but they use different approaches in restricting the number of entries that are necessary to evaluate in table D. The trivial solution is to compute all entries of table D in time $O(mn)$. Considering computation along diagonals gives a simple way to limit unnecessary computation. It is easy to show that entries on every diagonal h are monotonically increasing [41]. Therefore the computation along a diagonal can be stopped, when the threshold value of $k + 1$ is reached, because the rest of the entries on that diagonal will be greater than k. This idea leads to Algorithm EDP(Enhanced Dynamic Programming) working in average time $O(kn)$ [42].

There are other algorithms [15, 20, 43] similar to EDP with running time $O(kn)$. Whereas, there is no single method always the fastest. The speed of these algorithms varies according to the alphabet size, the series length and the value of k.

- Suffix tree-based approach

  Another popular approach is to support updates for dynamic strings using suffix tree indexing technique in [31, 29, 40, 44]. There are several appealing properties for the suffix tree: the construction of the suffix tree takes linear time and linear space; the frequency and position information of substrings are readily available in the suffix tree; the suffix tree serves as a natural and compact representation of sequential patterns. The major benefit of using suffix tree is that the suffix tree can be easily updated to solve the incremental similarity search problem.

A string S can be mapped to a tree T in which root-to-leaf paths are suffixes of S and terminal nodes represent uniquely starting positions of suffixes. Formally, the *suffix tree* T for S satisfies the following properties:

1. Each arc of T represents a non-empty substring of S,
2. Each non-terminal node of T, except the root, must have at least two offspring arcs,
3. Substrings represented by offspring arcs of the same node must begin with different characters.

Example 2.1.2 Consider the string $S = abcebcdbc\$$, we can build the suffix tree T of S by inserting suffixes into T one at a time, starting from the longest suffix *abcebcdbc*. In Figure 2.1 is the suffix tree after all suffixes of S are inserted. For any substring α of S, by following the path from the root that spells out α we can find the subtree containing all starting positions of α in terminal nodes. For instance, by following the path that spells out bc, i.e., arc(A, B) in this case, we find the root B of the subtree containing starting positions $2, 5, 8$ of bc in its terminal nodes.

2.2 Two Dimensions Polygons Similarity Searching

Shape matching is an important image processing operation. Considerable work has been done on this problem, with different techniques being used to identify shapes, usually in terms of boundary information or other local features [4].

There are quite a few streams of work in this area. One technique is to index an image after having analyzed it and recognized its semantic components [8]. Such techniques are not applicable with an image with no semantic information.

The other technique is to compute properties of local boundary features of objects, and then to index these [19]. Their idea relies on small features and hence is not robust.

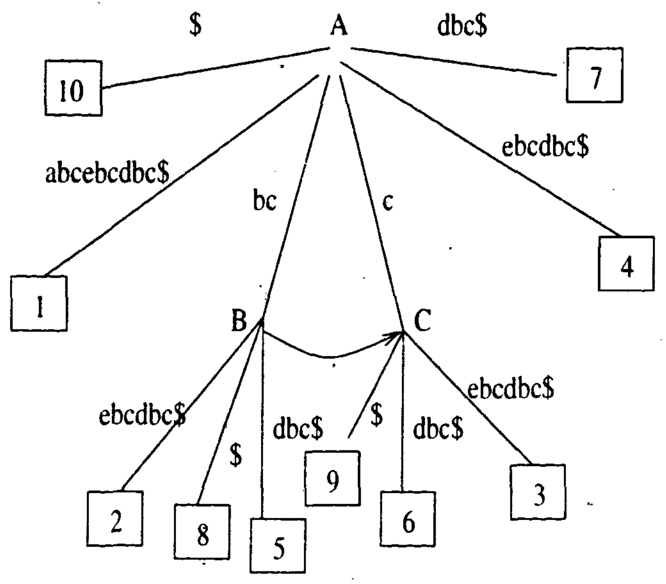


Figure 2.1: Construction of suffix trees

Jagadish in [24] introduces an indexing technique to retrieve shapes, which are similar to a given query shape, from a database. In this paper, he shows that the technique can be used for an area-based similarity measure, even in the presence of scaling and/or shifting in one or all dimensions.

2.3 The Similarity Search in Data Mining

The problem of discovering similarity patterns in massive time series data sets is an important and non-trivial one, on which a lot of work has been done [1, 14, 2, 34].

To our knowledge, [1] is the first work which proposes a solution for similarity matching sequences. In [1], it is assumed that all sequences are of the same length, 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 ϵ. Since each sequence is mapped to a point, they use R^*-tree [5] as the index structure. Sequences are represented as K-dimensional points using K features for each sequence. Discrete Fourier Transform (DFT) is used for feature extraction since it preserves the Euclidean distance. First K terms after the transformation are used to represent a sequence.

Faloutsos et al. extend the method proposed in [1] to locate subsequences that match a query sequence or a subsequence of it [14]. DFT is used for feature extraction; however, in order to transform subsequences, a sliding window of size W is used and put at every possible position on every data sequence. Therefore, instead of one point for a feature vector, there is a *trail* of points in the feature space. To index these points, the trails are divided into sub-trails and each of these subtrails is represented with its minimal bounding rectangle (MBR). Similarity queries are answered by applying the same transformation to the query sequence, and the MBR that enclose the trails for the query object is used as the query window. Sequences whose MBR's intersect with the MBR of the query sequence is further checked for matching.

Agrawal et al. [2] give a method to retrieve similar sequences in the presence of noise, scaling and translation in time series. In their method, two sequences are considered similar if they have enough non-overlapping time-ordered pairs of subsequences that are similar. Some portions of sequences that are considered as outliers are left out in the matching process, and the matching subsequences need not be aligned along the time axis. Testing the similarity of two subsequences is done by checking if one lies within an envelope of a specified width around the other, ignoring the outliers. Small, atomic subsequences that represent all the sequences are indexed using R-trees, based on some features of these atomic sequences. This method also depends on the sequence elements to be mostly correlated (i.e., not very distant from each other), since outliers are discarded in the matching process.

Davood et al. [34] use moving average method for stock data to smooth out short term fluctuations, and add time warping and reversing in their transformation language. It is implemented using the Fourier transform to transform the data to the frequency domain, and using R-tree index method.

One common feature of these methods [1, 14, 2, 34] is that they use the Discrete Fourier Transform (DFT) to map time sequences into the frequency domain and keep the first few coefficients in the index. Two sequences are considered similar if their Euclidean distance is less than a user-defined threshold. DFT preserves the Euclidean distance between sequences, and the first few coefficients after the transformation characterize the sequence in general, provided that consecutive sequence elements are correlated most of the time. The fact that it is a distance preserving transformation makes DFT attractive for indexing. However, it can be used only for sequences of the same length. Also, it is not very effective for sequences with mostly uncorrelated elements. In other words, DFT is well suited to sequences, which are locally stationary in time. However, many sequences contain transient behavior, e.g. short interval jumps or valleys, like the stock market data.

Another search direction is the periodic pattern directed search in time-series databases. Problems related to periodicity search is stated as *problems of finding patterns occurring at regular intervals*, which means given a sequence of events, we would like to find the patterns which repeat over time and their recurring intervals (period). Wan [48] has proposed some possible solution to the problem of finding periodic behaviors in large data sets.

2.4 The Similarity-Based Queries

There is a vast class of database applications where it is important to be able to pose queries in terms of *similarity* of objects, rather than equality or inequality [25].

The meaning of similarity may vary depending on different application domains. In other words, the notions of similarity will be different in different domains, such as sequence similarity searching in a time sequence database, approximate string searching in text, images, and genome/protein matching [1, 17, 24].

Jagadish et al. [25] develop a domain-independent framework to pose similarity queries on a database. It provides a full-fledged query language to support queries, such as "find all objects that are similar to some objects in class A and are not similar to any object in class B." The framework has three components: a *pattern language* P, a *transformation rule language* T, and a *query language* L. An expression in P specifies a set of data objects. An object A is considered similar to an object B, if B can be reduced to it by a sequence of transformations defined in T. The query language proposed by the paper is an extension of relational calculus with predicates that test whether an object A can be transformed into a member of the set of objects described by the expression e using the transformation t, at a cost bounded by c. The framework can be "tuned" to the needs of a specific application domain by the choice of P, T, and L.

For time-series approximate similarity queries, Goldin et al. [16] propose a framework that allows the user to pose a wide variety of queries, and allows shifts and positive scales transformations. The main contribution of them is that they formalize the intuitive notions of exact and approximate similarity between time-series patterns and data. The resulting set of constraint queries support the indexing scheme proposed in [1, 14].

Agrawal et al. [3] present a shape definition language, called *SDL*, for retrieving objects based on shapes contained in the histories associated with these objects. It is a small, yet powerful, language that allows a rich variety of queries about the shapes found in historical time sequences. An interesting feature of *SDL* is its ability to perform blurry matching. A "blurry" match is one where the user cares about the overall shape but does not care about specific details. But there is no support for shifts and scales transformations in this language.

Chapter 3

The Similarity Miner System

In this chapter we describe the architecture of our similarity miner system and explain our definition of similarity queries and the notions that will be used in our system.

3.1 General Definitions

Here we will clarify some notations and concepts that are generally discussed in the time-series similarity search area.

- **Time sequence or time-series data** are sequences of real numbers representing measures at uniformly-spaced temporal instances [16]. The ith element of a sequence S is $S[i]$, and a subsequence of S consisting of elements i through j is $S[i, j]$. The length of the sequence $S[i, j]$ is equal to $j - i + 1$.

- **Two similarity problems.** In the time-series similarity search area, the similarity problems can be classified into two categories:

  a. *All-Occurrences Subsequence Matching:* given a query sequence Q of length n and a much longer data sequence S of length N, find the first (all) occurrences of a contiguous subsequence within S that matches Q approximately;

b. *All-Occurrences Whole-Sequence Matching:* given a query sequence Q of length n and a set of N data sequences, all of the same length n, find the first (all) of the data sequences that match Q approximately.

The first case is under the condition that the query sequence is smaller, and we look for subsequences in the larger sequence that best match the query sequence. The second case is under the condition that the sequences to be compared have the same length n, and we look for the sequences that match the query sequence.

These two cases are all solved in this thesis in the *subsequence search module* and the *whole sequence search module* respectively .

- **Similarity queries**

We consider four different types of queries in our system. These are:

i **Full match.** A full match for a given query shape is a database shape that has the same shape as the query shape.

ii **Match with shift.** Usually, when we think of what a shape looks like, we do not care about the position of the shape in any coordinate system. Thus, we would like to retrieve similar shapes from the database irrespective of their positions in the coordinate system used to describe them.

iii **Match with scaling.** Besides not caring about the position of the shape, we may not care about the size either. For example, the size may depend on how far the shape is seen from the human eye, or what scale factor is used for the representation. In such a case, we can throw out the scale factor to retrieve the similar shapes. In the real implementation, we may wish to permit independent scaling along the X and Y axes or a uniform scaling along the X and Y axes.

iv **Match with the combination of scaling and shifting.** Often, the similarity search criterion is enlarged to allow the combination of scaling and shifting, that means, to allow the scaling and shifting existing at the same time, not independently.

- **Approximate match**

  In the former descriptions, we describe the basic discipline to retrieve shapes that match a given query shape. We are only interested in the approximate matching here, and the reason has already been illustrated in the introduction. The "matching approximately" is defined as [16]:

Definition 3.1 *Given a tolerance $\epsilon \geq 0$ and a distance metric D between sequences, sequences S_1 and S_2 match approximately within tolerance ϵ when $D(S_1, S_2) \leq \epsilon$.*

Currently, a large number of distance metrics have been proposed based on similarity in the literature. One of the generally used distance metrics is called *Euclidean distance*, which is defined as follows:

Definition 3.2 *The Euclidean distance between two sequences S_1 and S_2 is*

$$D_E(S_1, S_2) = (\sum_{1 \leq l} (S_1[i] - S_2[i])^2)^{1/2} \tag{3.1}$$

l is the length of the sequence.

3.2 The Architecture of Similarity Miner System

The similarity miner system is one of the knowledge discovery modules of our relational data mining system, DBMiner, researched and developed in our laboratory [21]. Besides the similarity miner module, currently, the discovery modules of DBMiner include characterizer, comparator, classifier, associator, and predictor.

In this section, we will introduce the architecture of the similarity miner module.

Figure 3.1 shows the general architecture of similarity miner system which consists of

(1) a graphical user interface for interactive mining and the display of data mining result in the form of charts and text edit box;

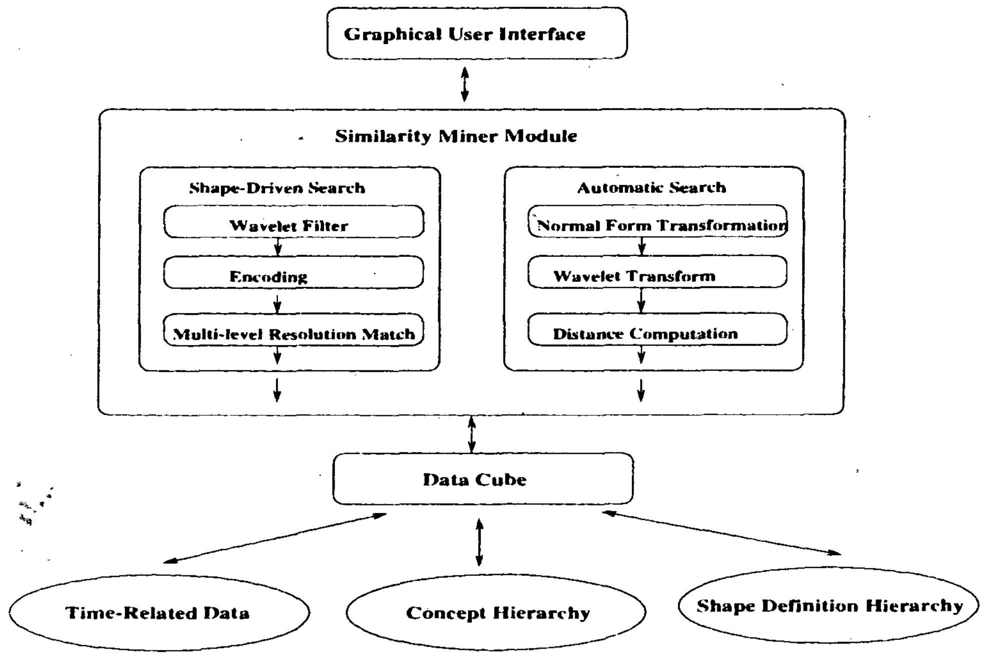


Figure 3.1: General architecture of similarity miner system

Our system is primarily focused on the data mining environment in which the user through our graphical user interface can find all similar time subsequences in a given sequence and be able to find all similar time sequences that match a given query sequence. Also through our user interface, the user can change at run time the similarity threshold, so called the "tolerance of outliers", the query sequence pattern, and the resolution level of similarity, while maintaining efficiency of matching. Furthermore, the user can use OLAP operations to drill down and roll up along any dimensions of the cube.

(2) the similarity miner modules;

We have designed two similarity mining modules: `subsequence search module` and `whole sequence search module`, which will be discussed in detail in later chapters.

(3) the DBMiner *multi-dimensional data cube*;

The DBMiner *multi-dimensional data cube* is the common engine shared among several DBMiner modules and it is a multi-dimensional array structure, in which each dimension represents a generalized attribute and each cell stores the value of some aggregate attributes. The advantages of it are less space and efficiency. It is implemented by our Intelligent Database Systems Research laboratory[23]. The similarity miner module and other mining modules are implemented on the basis of DBMiner *multi-dimensional data cube*. For the time-related data sets,

- **Dimensions** of the cube may have following categories:
  1. *time*.
  2. *time-related attribute*.
  3. *non-time-related attribute*.
- **Measures** can be :
  1. *count*. It is the default measure for all kinds of the cubes. It is a numerical measure, which represents the count of the tuples in the raw data.
  2. *time-related attribute value*. Since our research domain is the time-related real numbers, here we are only interested in numerical measure containing only numerical data and also being able to be computed by cube partition and aggregation, such as count, sum, max.

  With these dimensions and measures, OLAP operations can be performed by stepping up and down along any dimensions shown in Figure 3.2.
- **Roll-up** generalizes one or a few dimensions and performs appropriate aggregations in the corresponding measure(s). For time-related data cube in Figure 3.2, when the roll-up is performed along the *time* dimension, such as from "month" generalizes to "quarter ", the count measure and the sales amount measure are aggregated correspondingly.
- **Drill-down**, which specializes one or a few dimensions and presents low-level objects, collections, or aggregations, can be viewed as a reverse operation of *roll-up*.

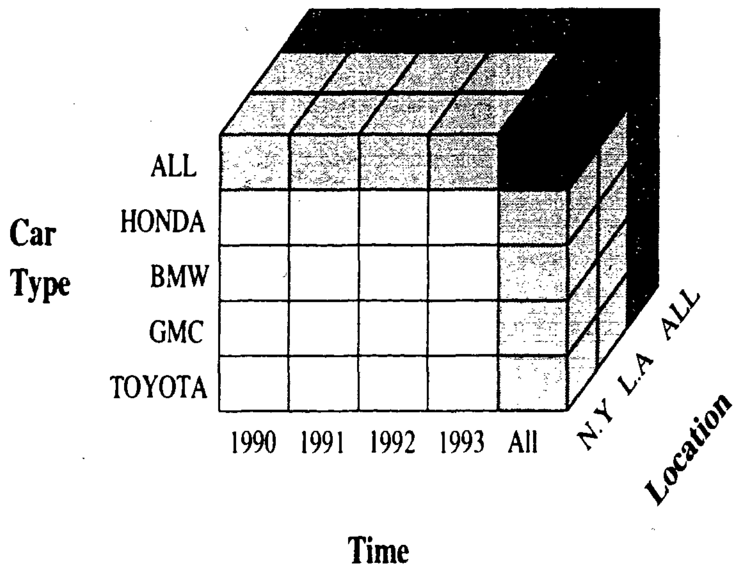


Figure 3.2: A cube about the sales of cars during 4 years

- **Slice_and_dice**, which corresponds to reducing the dimensionality of the data, i.e., taking a projection of the data on a subset of dimensions for selected values of the other dimensions. For example, we can slice_and_dice sales data for a specific product.

Example 3.2.1 Figure 3.2 shows an example of a cube which is constructed after sucking the user required information from the raw data. The dimensions are "Time", "Location" and "Car Type". The measures are "counts" and "sales amount". In this case, "Location" and "Car Type" are non-time-related attributes, "Time" is the time-related attribute, and "sales amount" is the time-related attribute value.

(4) the data- and knowledge-base: storing the time-related data, concept hierarchies, and the shape definition hierarchy.

The *concept hierarchy* is an important function module of `DBMiner`. It is a partial order organization of concepts in databases. Some partial orders among

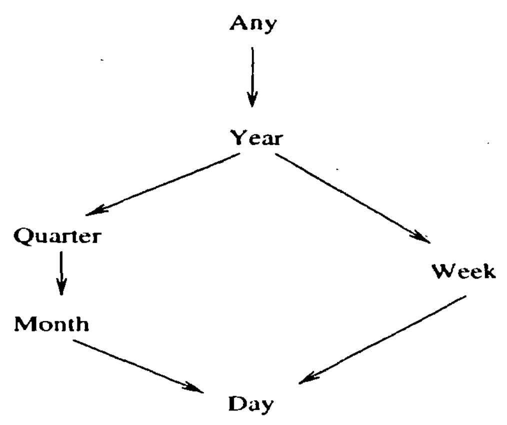


Figure 3.3: Partial ordered hierarchy for 'time' dimension

data exist in a database. For example, "B.C." is a part of "Canada". It provides essential background knowledge for data generalization and multiple-level data mining. For example, to mine the knowledge of the time-related databases at multiple-levels, time is a dimension that is of particular significance, and we need the built-in concept hierarchy for the time dimension.

Example 3.2.2 Figure 3.3 is the concept hierarchy for time domain, "Day" is the child of "Month", and "Month" is the child of "Quarter", and "Quarter" is the child of "Year".

Each dimension in a cube can be described by three ways: 1. a set of attributes, 2. a combination of attributes and concepts defined by expert, and 3. a pure set of concepts defined by expert, like in Figure 3.3 for the time dimension. These three ways form a partially ordered hierarchy for a dimension.

Example 3.2.3 For the cube in example 3.2.1, the partially ordered hierarchy of the "Car Type" dimension and "Location" dimension is shown in Figure 3.4.

The *shape definition hierarchy* is a hierarchy specially designed for the subsequence search module to achieve the goal of mining the similarity at multiple levels. We will introduce it in detail in later chapters.

$$\begin{tikzcd}
\textbf{Industry} \arrow[d] & \textbf{Country} \arrow[d] \\
\textbf{Category} \arrow[d] & \textbf{State} \arrow[d] \\
\textbf{Car Type} & \textbf{City}
\end{tikzcd}$$

Figure 3.4: Partial ordered hierarchy for Car Type dimension and Location dimension

The main part of our similarity miner system is the similarity miner module, which is designed and implemented. The challenges of the design of similarity miner module will be discussed in the following section.

3.3 Challenges in the Design of the Similarity Miner Module

To design a good similarity miner system, there will be many challenges.

Sequence matching problem can be characterized on the basis of the type of sequences, and the domain of elements in sequences. The data on which our knowledge discovery tools operate can be characterized by the traditional distinction between *categorical* data and *numerical* data [27]. We may have sequences of different lengths, the elements of sequences can be categorical data or numerical data. All these issues affect the design of the similarity miner system to efficiently handle the sequence data and similarity match queries. In this thesis, we address the general problem of matching sequences of the same lengths as well as different lengths, and put our emphasis on numerical sequences.

Queries on continuous data are more complex to process than the queries on discrete data. There is a lot of work that has been done on the discrete data in similarity search area, such as text similarity search. In recent years, the study of similarity matching on the continuous data begins to be popular, such as the stock data similarity analysis. In this thesis our study is focused on the time-series data sets in which the data are real numbers, and we have designed methods, which adopt some techniques of the discrete similarity search methods, and at the same time, the specific characteristics of the continuous data are also taken into consideration.

Chapter 4

Wavelet Analysis

Since the *wavelet analysis* technique is used in both of the two modules of our similarity system, in this chapter we will introduce the background knowledge about the wavelet analysis.

As everyone knows, any signal can be portrayed as an overlay of sinusoidal waveforms of assorted frequencies. But while classical Fourier analysis copes superbly with naturally occurring sinusoidal behavior – the kind seen in speech signals – it is ill suited to representing signals with discontinuities, such as the edges of features in images [6]. Whereas, another powerful concept: wavelet analysis has swept applied mathematics and engineering research. It involves representing general functions in terms of simpler, fixed building blocks at different scales and positions. This has been found to be a useful approach in several different areas. For example, in signal and image processing. It has been testified that getting rid of signals of noise is often much easier in the wavelet domain than in the original Fourier domain. With wavelets, noise can be removed from a great many signal types, including those with jumps, spikes and other non-smooth features [6].

Wavelets are a family of functions of orthogonal basis, which unlike the sine and cosine wave of the Fourier Transform (FT), do not have to have infinite duration. They can be non-zero for only a small range of the wavelet function [12, 13]. This "compact support" nature allows the Wavelet Transform (WT) to translate a time-domain function into a representation that is not only localized in frequency, like the

FT, but in time as well. This ability has brought forth new developments in the fields of signal analysis, image processing, and data compression.

Like Fourier analysis, however, wavelet analysis uses an algorithm to decompose a signal into simpler elements. Wavelet analysis is far more efficient than Fourier analysis whenever a signal is dominated by transient behavior or discontinuities. Wavelet analysis owes its efficiency to the fast pyramid algorithm. The algorithm is used to compute the wavelet transform that is to decompose the signal into component wavelets.

Wavelet noise removal is superior to traditional Fourier techniques, and it has been shown to work well for our stock time-related databases.

Since our research domain of time-series databases is one dimension, we only cover one-dimensional wavelet transforms and basis functions.

4.1 From the Fourier Transforms to the Wavelet Transforms

Fourier and wavelet analysis have some very strong links. Let us have a look at Fourier analysis first.

4.1.1 Fourier Analysis

- Fourier Transforms

  It has been known for quite some time that techniques based on Fourier series and Fourier transforms (FT) are reliable tools in signal analysis. The main idea of the FT is that it expands the original function in terms of orthonormal functions of sine and cosine waves. The Fourier coefficients of the transformed function then represent the contribution of each sine and cosine wave at each frequency.

- Discrete Fourier Transforms

The Discrete Fourier Transform (DFT) estimates the Fourier transform of a function from a finite number of its sampled points. The sampled points are supposed to be typical of what the signal looks like at all other times. In other words, DFT represents a given function in terms of discrete sine and cosine wave series, and is the most often used form in the FT. It works under the assumption that the original time-domain function is periodic in nature. Thus, the DFT has difficulty with signals have sharp transitions at certain time locations. As a result, for those transition components, a large number of Fourier modes may be needed. Another problem with the DFT is that only the frequency information is provided, and the information about the translation of the signal in time is not provided.

- Windowed Fourier Transforms

  If $f(t)$ is a non-periodic signal, the summation of the periodic functions, sine and cosine, does not accurately represent the signal. The windowed Fourier transform (WFT) is one solution to the problem of better representing the non-periodic signal. The WFT can be used to give information about signals simultaneously in the time domain and in the frequency domain.

  With the WFT, the input signal $f(t)$ is chopped up into sections, and each section is analyzed for its frequency content separately. This windowing is accomplished via a weight function that places less emphasis near the interval's endpoints than in the middle. The effect of the window is to localize the signal in time [18].

- Fast Fourier Transform

  To approximate a function by the discrete transform, requires applying a matrix whose order is the number of sample points n. Since multiplying an $n \times n$ matrix by a vector costs on the order of n^2 arithmetic operations, the problem gets quickly worse as the number of sample points increases. However, if the samples are uniformly spaced, then the Fourier matrix can be factored into a product of just a few sparse matrices, and the resulting factors can be applied to a vector in a total of order $n \log n$ arithmetic operations. This is the so-called *fast Fourier transform* or FFT [18].

4.1.2 Similarities between Fourier and Wavelet Transforms

The fast Fourier transform (FFT) and the discrete wavelet transform (DWT) are both linear operations that generate a data structure that contains $\log_2 n$ segments of various lengths.

The mathematical properties of the matrices involved in the transforms are similar as well. Both transforms can be viewed as a rotation in function space to a different domain. For the FFT, this new domain contains basis functions that are sines and cosines. For the wavelet transform, this new domain contains more complicated basis functions called wavelets, or mother wavelets.

4.1.3 Dissimilarities between Fourier and Wavelet Transforms

The most interesting dissimilarity between these two kinds of transforms is that wavelet functions are *localized in space*. Fourier sine and cosine functions are not. This localization feature, along with wavelets' localization of frequency, makes many functions and operators using wavelets "sparse" when transformed into the wavelet domain. This sparseness, in turn, results in a number of useful applications such as data compression, detecting features in images, and removing noise from time series.

Let us look at the standard Fourier transform,

$$(\mathcal{F}f)(\omega) = \frac{1}{\sqrt{2\pi}} \int dt e^{-i\omega t} f(t), \tag{4.1}$$

Like wavelet transform, it also gives a representation of the frequency content of f, but information concerning time-localization cannot be read off easily from $\mathcal{F}f$.

Windowed Fourier transform is a standard technique for time-frequency localization, which is achieved by first windowing the signal f, so as to cut off only a well-localized slice of f, and then taking its Fourier transform:

$$(T^{win}f)(\omega,t) = \int dsf(s)g(s-t)e^{-i\omega s}, \tag{4.2}$$

Its discrete version is when t and w are assigned regularly spaced values: $t = nt_0, w = mw_0$, where m, n range over Z, and w_0, $t_0 > 0$ are fixed. Then (4.2) becomes

$$T^{win}_{m,n}(f) = \int dsf(s)g(s - nt_0)e^{-im\omega_0 s}, \tag{4.3}$$

The **wavelet transform** provides a similar time-frequency description, with a few important differences. The wavelet transform formulas analogous to (4.2) and (4.3) are

$$(T^{wav}f)(a,b) = |a|^{-1/2}\int dtf(t)\psi(\frac{t-b}{a}) \tag{4.4}$$

and

$$T^{wav}_{m,n}(f) = a_0^{-m/2}\int dtf(t)\psi(a_0^{-m}t - nb_0). \tag{4.5}$$

In both cases we assume that ψ satisfies

$$\int dt\psi(t) = 0 \tag{4.6}$$

Formula (4.5) is again obtained from (4.4) by restricting a,b to only discrete values: $a = a_0^m$, $b = nb_0a_0^m$ in this case, with m, n ranging over Z, and $a_0 > 1$, $b_0 > 0$ fixed.

One *similarity* between the wavelet and windowed Fourier transforms is clear: both (4.2) and (4.4) take the inner products of f with a family of functions indexed by two labels, $g^{\omega,t}(s) = e^{i\omega s}g(s-t)$ in (4.2), and $\psi^{a,b}(s) = |a|^{-\frac{1}{2}}\psi(\frac{s-b}{a})$ in (4.4). The functions $\psi(a,b)$ are called "wavelets"; and the function ψ is sometimes called "mother wavelet".

The *difference* between the wavelet and windowed Fourier transforms lies in the shapes of the analyzing functions $g^{\omega,t}$ and $\psi^{a,b}$. All the $g^{\omega,t}$, regardless of the value of ω, have the same width. In contrast, the $\psi^{a,b}$ have time-widths adapted to their frequency: the high frequency ones $\psi^{a,b}$ are very narrow, while the low frequency ones $\psi^{a,b}$ are much broader. As a result, an advantage of wavelet transforms over the windowed Fourier transforms is that the windows vary, "zoom in" on very short-lived high frequency phenomena, such as transients in signals (or singularities in functions). The wavelet analysis provides immediate access to information that can be obscured by other time-frequency methods such as Fourier analysis [12].

Usually the following integral powers of 2 for frequency partitioning is used:

$$\psi(2^j x - k), \quad j, k \in \mathcal{Z}$$

Notice that $\psi(2^j x - k)$ is obtained from a single wavelet function $\psi(x)$ by a binary dilation (i.e. dilation by 2^j) and a dyadic translation. With the normalization, the following functions are usually used [10]:

$$\psi_{j,k}(x) = 2^{j/2}\psi(2^j x - k), \quad j, k \in \mathcal{Z} \tag{4.7}$$

Definition 4.1 *[10] A function $\psi \in L^2(\mathcal{R})$ is called an orthogonal wavelet, if the family $\{\psi_{j,k}\}$ in (4.7) is an orthogonal basis of $L^2(\mathcal{R})$; that is,*

$$\langle \psi_{j,k}, \psi_{l,m} \rangle = \delta_{j,l} \cdot \delta_{k,m}, \quad j, k, l, m \in \mathcal{Z},$$

and every $f \in L^2(\mathcal{R})$ can be written as

$$f(x) = \sum_{j,k=-\infty}^{\infty} c_{j,k}\psi_{j,k}(x), \tag{4.8}$$

where $\delta_{j,l}$ is the Kronecker symbol and the convergence of the series in (4.8) is in $L^2(\mathcal{R})$.

The series representation of f in (4.8) is called a *wavelet series*. Analogous to the notion of Fourier coefficients, the wavelet coefficients $c_{j,k}$ is given by

$$c_{j,k} = \langle f, \psi_{j,k} \rangle. \tag{4.9}$$

4.2 Multi-resolution Analysis

The constructions of the wavelet basis as in (4.7) and other different ones can all be realized by a "multi-resolution analysis". Multi-resolution analysis provides a natural framework for the understanding of wavelet bases. The idea behind it is to write L^2-functions f as a limit of successive approximations, each of which is a smoothed version of f, with more and more concentrated smoothing functions. The successive approximations thus use a different resolutions [11]. A multi-resolution analysis consists of a sequence of successive approximation spaces V_j. More precisely, the closed subspaces V_j satisfy (here the scaling factor is 2):

(1) a family of embedded closed subspaces $V_m \subset L^2(\mathcal{R})$, $m \in Z$,

$$\cdots V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots \tag{4.10}$$

(2)

$$\overline{\bigcup_{j \in Z} V_j} = L^2(\mathcal{R}) \tag{4.11}$$

$$\bigcap_{j \in Z} V_j = \{0\} \tag{4.12}$$

If we denote by P_j the orthogonal projection operator onto V_j, then (4.11) ensures that $\lim_{j\to\infty} P_j f = f$ for all $f \in L^2(\mathcal{R})$.

There are many ladders of spaces satisfying (4.10)-(4.12) which have nothing to do with "multi-resolution". The multi-resolution aspect is a consequence of the additional requirement as follows.

(3) all the spaces are scaled versions of the central space V_0:

$$f(\cdot) \in V_j \Longleftrightarrow f(2^{-j}\cdot) \in V_0 \tag{4.13}$$

or

$$f(\cdot) \in V_0 \Longleftrightarrow f(2^{j}\cdot) \in V_j \tag{4.14}$$

(4) V_0 is invariant under integer translations,

$$f \in V_0 \Longleftrightarrow f(\cdot - n) \in V_0 \tag{4.15}$$

(5) For every $j \in \mathcal{Z}$, define W_j to be the orthogonal complement of space V_j in V_{j+1}

$$V_{j+1} = V_j \bigoplus W_j \tag{4.16}$$

(6) For $\forall j \neq j'$,

$$W_j \perp W_j' \tag{4.17}$$

and

$$V_j \perp W_j \tag{4.18}$$

(7) If we denote P_{V_j} and P_{W_j} as the orthogonal projection onto V_j and W_j respectively, and $\phi_{j,k}$ and $\psi_{j,k}$ denote the orthogonal basis in V_j and W_j with $j, k \in \mathcal{Z}$, then we have

$$P_{V_{j+1}} f = P_{V_j} f + P_{W_j} f = \sum_{k=-\infty}^{\infty} \langle f, \phi_{j,k} \rangle \phi_{j,k} + \sum_{k=-\infty}^{\infty} \langle f, \psi_{j,k} \rangle \psi_{j,k} \tag{4.19}$$

(8) For $j_{min}, j_{max} \in \mathcal{Z}, j_{min} < j_{max}$, we have

$$V_{j_{max}} = V_{j_{min}} \oplus \bigoplus_{j=j_{min}}^{j_{max}-1} W_j \tag{4.20}$$

4.3 Fast Wavelet Transform Algorithms

Multi-resolution analysis leads naturally to a hierarchical and fast scheme for the computation of the wavelet coefficients of a given function. Before we discuss about the wavelet transform algorithm, let us write out some interesting properties of ϕ and ψ.

(1) For $\phi, \psi \in V_0 \subset V_1$, and $\phi_{1,n}$ is an orthonormal basis in V_1, we have

$$\begin{aligned}\phi &= \sum_n h_n \phi_{1,n} \\ &= \sqrt{2}\sum_n h_n \phi(2x-n) \end{aligned} \tag{4.21}$$

$$\begin{aligned}\psi(x) &= \sum_n (-1)^{n+1} h_{-n+1}\phi_{1,n} \\ &= \sqrt{2}\sum_n (-1)^{n+1} h_{-n+1}\phi(2x-n) \end{aligned} \tag{4.22}$$

with

$$h_n = \langle \phi, \phi_{1,n}\rangle, \quad \text{and} \quad \sum_n |h_n|^2 = 1.$$

(2) Suppose that we have computed or given the inner products of f with $\phi_{j,k}$ at some given fine scale $j = 0$. By rescaling f we can easily compute $\langle f, \psi_{j,k}\rangle$ for $j \leq -1$, from $\langle f, \phi_{j+1,k}\rangle$:

$$\langle f, \psi_{j,k}\rangle = \sum_n \overline{g_{n+2k}}\langle f, \phi_{j+1,n}\rangle \tag{4.23}$$

where $g_n = \langle \psi, \phi_{1,n}\rangle = (-1)^{n+1}h_{-n+1}$. We also can easily compute $\langle f, \phi_{j,k}\rangle$ for $j \leq -1$, from $\langle f, \phi_{j+1,k}\rangle$:

$$\langle f, \phi_{j,k}\rangle = \sum_n \overline{h_{n+2k}}\langle f, \phi_{j+1,n}\rangle \tag{4.24}$$

From the above observation, we have a general procedure: starting form $\langle f, \phi_{0,n}\rangle$, we compute $\langle f, \psi_{-1,k}\rangle$ by (4.23). We can then apply (4.23) and (4.24) again to compute $\langle f, \psi_{-2,k}\rangle$, $\langle f, \phi_{-2,k}\rangle$ from $\langle f, \phi_{-1,n}\rangle$, etc. At every step we compute not only the wavelet coefficients $\langle f, \psi_{j,k}\rangle$ of the corresponding j-level, but also the $\langle f, \phi_{j,k}\rangle$ for the same j-level, which are useful for the computation of the next level wavelet coefficients.

The whole process can also be viewed as the computation of successively coarser approximations of f, together with the difference in "information" between every two successive levels. In this view we start out with a fine-scale approximation to f,

$$\begin{tikzcd}
c^{0} \arrow[r, "\overline{H}"] \arrow[dr, "\overline{G}"] & c^{-1} \arrow[r, "\overline{H}"] \arrow[dr, "\overline{G}"] & c^{-2} \arrow[r, "\overline{H}"] \arrow[dr, "\overline{G}"] & c^{-3} \\
 & d^{-1} & d^{-2} & d^{-3}
\end{tikzcd}$$

Figure 4.1: Wavelet forward transform

$f^0 = P_0$, recall that P_j is the orthogonal projection onto V_j, and we decompose $f^0 \in V_0 = V_{-1} \oplus W_{-1}$ into $f^0 = f^{-1} + \delta^{-1}$, where $f^{-1} = P_{-1}f^0 = P_{-1}f$ is the next coarser approximation of f in the multi-resolution analysis, and $\delta^{-1} = f^0 - f^{-1} = (I - P_{-1})f^0$ is what is "lost" in the transition $f^0 \to f^{-1}$. In each of these V_j, W_j spaces we have the orthonormal bases $\{\phi_{j,k}\}_{k\in\mathcal{Z}}$ and $\{\psi_{j,k}\}_{k\in\mathcal{Z}}$ respectively, so that

$$f^0 = \sum_n c_n^0 \phi_{0,n}, \quad f^{-1} = \sum_n c_n^{-1}\phi_{-1,n}, \quad \delta^0 = \sum_n d_n^0 \psi_{0,n}.$$

Formulas (4.23) and (4.24) give the effect on the coefficients of the orthogonal basis transformation $\{\phi_{0,n}\}_{n\in\mathcal{Z}} \to \{\psi_{-1,n}\}_{n\in\mathcal{Z}}$ in V_0:

$$c_k^{-1} = \sum_n \overline{h_{n-2k}} c_n^0, \quad d_k^{-1} = \sum_n \overline{g_{n-2k}} c_n^0$$

In general for any $j \leq 0$ we have,

$$\begin{cases} c_k^{j-1} = \sum_n \overline{h_{n-2k}} c_n^j; \\ d_k^{j-1} = \sum_n \overline{g_{n-2k}} c_n^j. \end{cases}$$

Remark 4.1 *The transition $c^j \to c^{j-1}, d^{j-1}$ corresponds to a change of basis in V_j, namely*

$$\{\phi_{j,k} \mid k \in \mathcal{Z}\} \to \{\phi_{j-1,k} \mid k \in \mathcal{Z}\} \cup \{\psi_{j-1,k} \mid k \in \mathcal{Z}\}.$$

The decomposition of the signal c^j at the level j into a lower resolution signal c^{j-1} and a difference (or detail) signal d^{j-1} forms the basis for the **pyramid algorithm** *described in [30].*

With the notations $a = (a_n)_{n\in\mathbb{Z}}$, $\tilde{a} = (\overline{a_{-n}})_{n\in\mathbb{Z}}$ and $(Ab)_k = \sum_n a_{2k-n}b_n$, we can rewrite this as

$$c^{-1} = \overline{H}c^0, \quad d^{-1} = \overline{G}c^0$$

The coarser approximation $f^{-1} \in V_{-1} = V_{-2} \oplus W_{-2}$ can again be decomposed into $f^{-1} = f^{-2} + \delta^{-2}$, $f^{-2} \in V_{-2}$, $\delta^{-2} \in W_{-2}$, with

$$f^{-2} = \sum_n c_n^{-2}\phi_{-2,n}, \quad \delta^{-2} = \sum_n d_n^{-2}\psi_{-2,n}$$

Schematically, all these can be represented as in Figure 4.1.

Since all we have done is a succession of orthogonal basis transformations, the inverse operation, or the reconstruction, is given by the adjoint matrices. That is given by

$$\begin{aligned} f^j &= f^{j-1} + \delta^{j-1} \\ &= \sum_k c_k^{j-1}\phi_{j-1,k} + \sum_k d_k^{j-1}\psi_{j-1,k} \end{aligned}$$

Therefore,

$$\begin{aligned} c_n^j &= \langle f^j, \phi_{j,n}\rangle \\ &= \sum_k c_k^{j-1}\langle\phi_{j-1,k}, \phi_{j,n}\rangle + \sum_k d_k^{j-1}\langle\psi_{j-1,k}, \phi_{j,n}\rangle \\ &= \sum_k \left[h_{n+2k}c_k^{j-1} + g_{n+2k}d_k^{j-1}\right] \end{aligned}$$

An important aspect of the whole decomposition is that it is a fast algorithm. Let us look at the Haar basis for a moment. The Haar basis is the simplest wavelet basis. If we start with N data points c_n^0, then we have to compute $N/2$ averages c_n^1, and $N/2$ differences d_n^1; from the $N/2$ averages c_n^1 we compute $N/4$ averages c_n^2 and $N/4$ differences d_n^2, and so on. The total number of computations is therefore

$$\begin{tikzcd}
{} \arrow[r] & c^{-3} \arrow[r, "\bar{H}^*"] & c^{-2} \arrow[r, "\bar{H}^*"] & c^{-1} \arrow[r, "\bar{H}^*"] & c^{0} \\
& d^{-3} \arrow[ur, "\bar{G}^*"] & d^{-2} \arrow[ur, "\bar{G}^*"] & d^{-1} \arrow[ur, "\bar{G}^*"] &
\end{tikzcd}$$

Figure 4.2: Wavelet inverse transform (reconstruction)

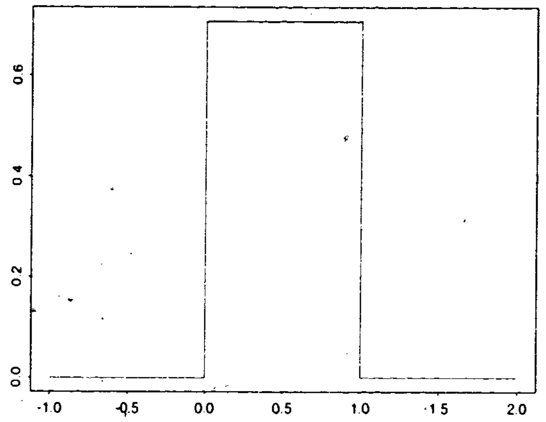
$2\left(\frac{N}{2}+\frac{N}{4}+\cdots\right) = 2N$. For more sophisticated wavelet ~~bases, the "averages"~~ and "differences" involve more than just two numbers, but the same argument holds. If every "generalized average or difference" involves K coefficients of the previous level (rather than 2 as in the Haar case), then the total number of computations is $2KN$ [13], which is faster than the *fast Fourier transform* whose computations is of order $n \log n$.

4.4 Implementation of Wavelet Forward Transform and Inverse Transform

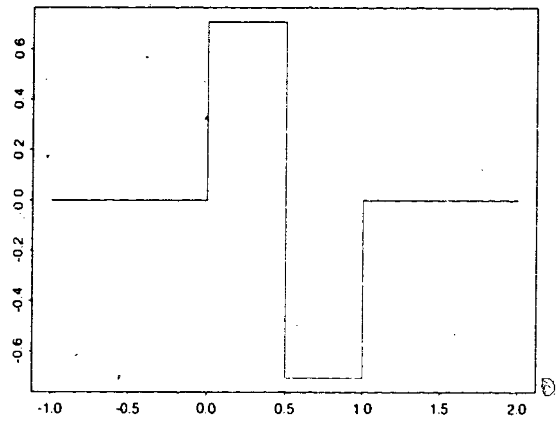
In this section, the implementation of the wavelet forward and inverse transform is introduced. The steps are as follows:

(1) Select the Wavelet Basis

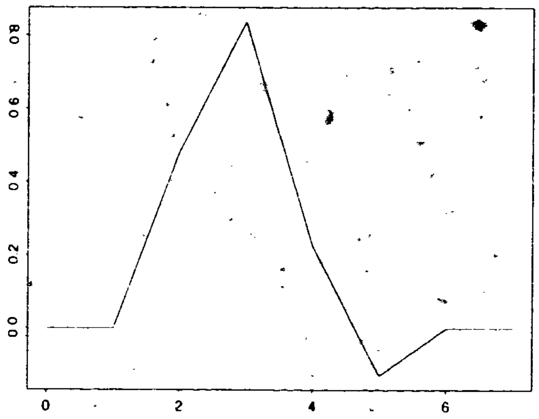
Wavelet transforms do not have a single set of basis functions like the Fourier transform, which utilizes just the sine and cosine functions. Instead, wavelet transforms have an infinite set of possible basis functions. The different wavelet families make different trade-offs between how compactly the basis functions are localized in time and how smooth they are.



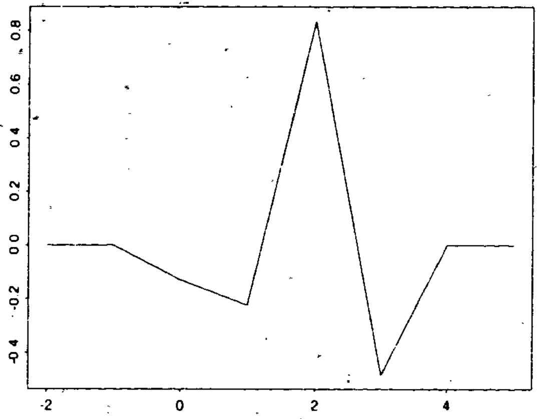
(a)*Harr_2* ϕ



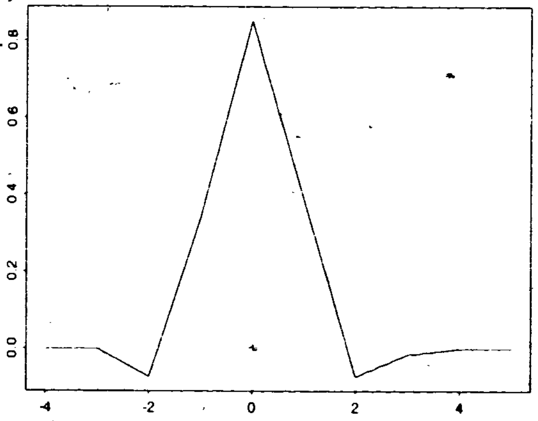
(b) *Harr_2* ψ



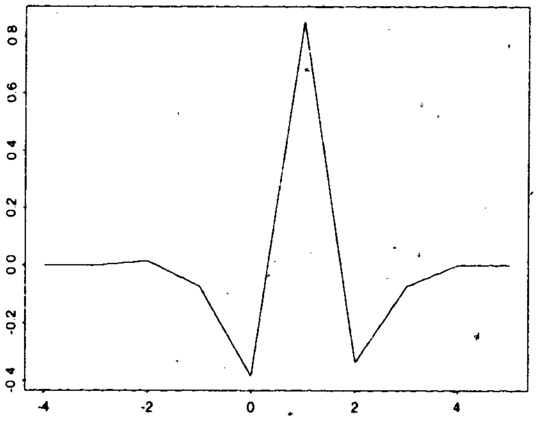
(c) Daubechies_4 ϕ



(d)Daubechies_4 ψ



(e)Coiflet_6 ϕ



(f)Coiflet_6 ψ

Figure 4.3: Several different families of wavelets

	n	h_n	g_n
Carr_2	0	0.7071067811865475	0.7071067811865475
	1	0.7071067811865475	-0.7071067811865475
Daubechies_4	0	0.4829629131445341	-0.1294095225512603
	1	0.8365163037378080	-0.2241438680420133
	2	0.2241438680420133	0.8365163037378080
	3	-0.1294095225512603	-0.4829629131445341
Coiflet_6	-2	-0.072732619512854	0.015655728135465
	-1	0.337897662457810	-0.072732965112707
	0	0.852572020212260	-0.384864846864200
	1	0.384864846864200	0.852572020212260
	2	-0.072732965112707	-0.337897662457810
	3	-0.015655728135465	-0.072732619512854

Table 4.1: The coefficients for the wavelets in Figure 4.3

Within each family of wavelets are wavelet subclasses distinguished by the number of coefficients, so called filter length. In Figure 4.3 several different families of wavelets are illustrated, and the number next to the wavelet name represents the filter length (the number of wavelet coefficients) for the subclass of wavelets. Their corresponding coefficients are summarized in Table 4.1. Usually, function ϕ is called "father wavelet", and the function ψ is called "mother wavelet".

Because we have a choice among an infinite set of basis functions, we may wish to find the best basis functions for a given signal. *A basis of adapted waveform* is the best basis function for a given signal. The chosen basis carries substantial information about the signal, and if the basis description is efficient (that is, very few terms in the expansion are needed to represent the signal), then that signal information has been compressed.

In general, higher-order wavelets (i.e., those with more non-zero coefficients) tend to have high compressibility which is more adapted according to the adapted waveform criterion.

The Haar wavelet is used for educational purpose because it represents a simple interpolation scheme, but the order of it is only 1, thus it is not quite applicable in real life. Daubechies families of wavelet systems are very good for representing polynomial behavior. One disadvantage of Daubechies is that they are not symmetric. The absence of similarity can lead to phase distortion.

From the theories and the experiments on the real data, we find the Coiflet wavelet basis functions adapt well to our stock data, and we select the filter lengths (the number of non-zero coefficients) of 6.

One reason that we select Coiflet wavelet is its symmetry property, and the other reason is the order of it is not quite low and has good compressibility. The filter length of 6 is reasonable for the time-related data, such as the stock data. The data in one week is quite related to each other, whereas to relate the stock data in more than one week sounds not reasonable.

To find the best-adapted wavelet basis functions for a given signal is not easy work. The research in this area is not quite mature yet. This can be our future work.

(2) Determine the Bandwidth (Scale) of a Finite Sequence.

Given a finite length sequence, the bandwidth can be determined. The *bandwidth* means the number of spaces to transform: from the finest space to the coarsest space.

Given N points, $f_0 \cdots f_{n-1} \in V_{j_{max}} \equiv V_0$, setting $2^{\Delta j} = N - 1$, so

$$\Delta j = \lfloor \log_2(N-1) \rfloor \tag{4.25}$$

$\Delta j = j_{max} - j_{min}$ is the number of possible bandwidth that can be obtained in a wavelet analysis with the scale factor is 2 [46].

Example 4.4.1 Consider a time sequence $\vec{s} = (10, 10, 11, 11, 8, 8, 13, 13)$, $\vec{s}$ could be the closing price of a stock. The number of data points N is 8, the original data $\vec{s} = (10, 10, 11, 11, 8, 8, 13, 13)$ is in space $V_{j_{max}} = V_0$, $\Delta j = j_{max} - j_{min} = \lfloor \log_2(8-1) \rfloor = 2$ according to equation (4.25). So the number of spaces to transform is 2.

$$\begin{tikzcd}
V_{j_{max}} \arrow[d] \arrow[dr] & \\
V_{j_{max}-1} \arrow[d] \arrow[dr] & W_{j_{max}-1} \\
V_{j_{max}-2} \arrow[d, dashed, no head] & W_{j_{max}-2} \\
V_{j_{min}+1} \arrow[d] \arrow[dr] & W_{j_{min}+1} \\
V_{j_{min}} & W_{j_{min}}
\end{tikzcd}$$

Figure 4.4: $V_{j_{max}} = V_{j_{min}} \oplus W_{j_{min}} \oplus W_{j_{min}+1} \oplus \cdots \oplus W_{j_{max}-1}$

(3) Compute the Coefficients of the Sub-Space

Let $c_{j-1,k} = \langle f, \phi_{j-1,k} \rangle$ and $d_{j-1,k} = \langle f, \psi_{j-1,k} \rangle$ be the coefficients in V_j and W_j respectively, according to Figures 4.1 and 4.4, where $c_{j-1,k}$ is determined by a dot product of the masking coefficients $\vec{h}$ with a subset of V_j namely $[V_{j,2k+\tau_{min}} \cdots V_{j,2k+\tau_{max}}]$ where $\vec{h} = \{h_n\}_{n=\tau_{min}}^{\tau_{max}}$, referring to equations (4.21) and (4.24).

$d_{j-1,k}$ is determined by a dot product of the masking coefficients $\vec{g}$ with a subset of V_j namely $[V_{j,2k+\tau_{min}} \cdots V_{j,2k+\tau_{max}}]$, where $\vec{g} = \{g_n\}_{n=\tau_{min}}^{\tau_{max}}$, referring to the equations (4.22) and (4.24).

In our implementation, we use the Coiflets wavelet basis, referring to Figure 4.3 and Table 4.1.

$\vec{h} = (-0.0727326195128 54, 0.33789766245781, 0.85257202021226,$
$0.38486484686420, -0.07273296511 2707, -0.01565572813 5465)$

and

- $\vec{g} = (0.0156557281354 65, -0.0727329651127 07, -0.38486484686420,$
$0.85257202021226, -0.33789766245781, -0.072732619512854),$
with $\tau_{min} = -2$, and $\tau_{max} = 3$.

Chapter 5

Subsequence Search Module

5.1 Problem Statement

The subsequence search module is designed to solve the problem of "First (All)-Occurrence Subsequence Matching", which means "given a query shape Q, find the similar shape subsequences in a time-series sequence."

The shape query Q, also called search template, used in our system is a string of symbols. Each different symbol represents a primitive shape unit. Since the query Q is a shape, so it is also called a shape-driven module.

The concept of "subsequence" search in similarity study is compatible to that of "substring" search in the complexity theory. The reason of our using the notion of "subsequence" search here is to follow the conventional notation proposed by the papers [1, 2, 14, 16] in the one-dimensional time-related data similarity search area.

Following is an example that this module can solve.

Example 5.1.1 Find the duration with the sales trend of mini-vans is "UUDD" (with two sharp ups and two sharp downs) between Jan. 1940 to Jan. 1996. After the subsequence search module, we may find that there is a subsequence from 1973 to 1980 having the trend similar to the query "two sharp downs followed by two sharp ups".

Here the symbol 'U' represents "sharp up", 'D' represents "sharp down", and

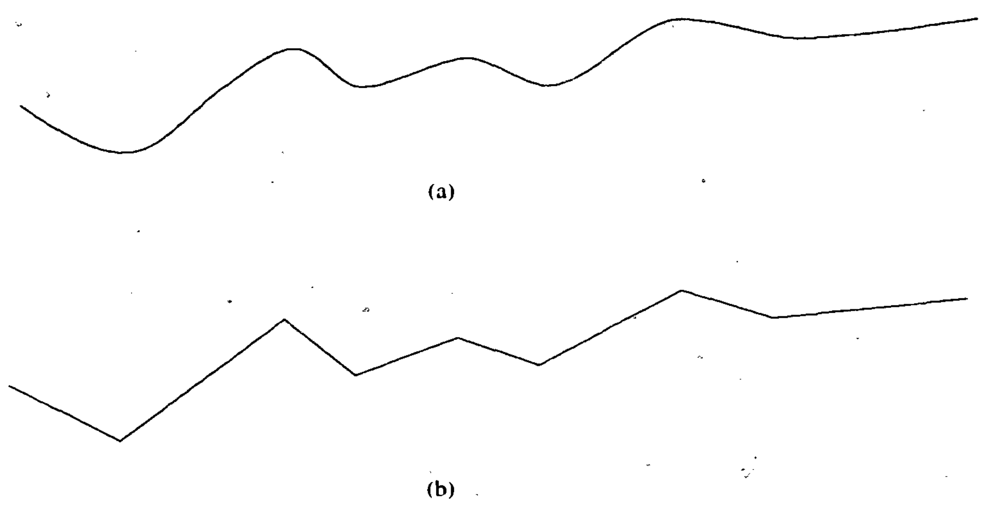


Figure 5.1: A segmented piecewise linear representation. (a) original data, and (b) the segmented version of this sequence.

"UUDD" is the shape query Q.

5.2 Terminology

In this section, we introduce some notations that will be used in the subsequence search module.

The `subsequence search` module functions as follows: the user through the user interface gives the shape query they want to ask, and the tolerance threshold, then the module finds the similar part of the sequence for the user. This is the *subsequence matching* problem, as we introduced in the former section. In our module we combine the OLAP techniques, allowing the user to drill up and down along the time dimension according to the time concept hierarchy, and also allowing the user to search the similarity along multi-level resolutions. Here are some definitions for implementing these functions:

- **Piecewise linear representation**

  The first thing we should solve is to find a representation method that can capture the sequence shape information, represent the data sequence, and furthermore facilitate the similarity search. There are numerous techniques for representing sequence data. The representation can critically influence the sensitivity of the distance metric and also can substantially determine the efficiency of the matching process. Thus a robust representation, which is computationally efficient to work with, is what we are looking for. We are interested in designing a representational language which can directly capture the notions of sequence shapes and which is intuitive as a language for human interaction.

To make the problem easier, we chop the sequence of data into many segments with equal length, and use a straight line, which can mostly approximate this bunch of data, to represent each segment. Thus the original data sequence is converted to a new kind of data sequence, just like in Figure 5.1 (a) and (b). This method is called piecewise linear segmentations, which provide both an intuitive and practical method for representing curves in a simple form. It generalizes the data from high order to low-order polynomial.

- **From continuous space to discrete space**

  After changing the sequence of data into piecewise linear representation, each segment is changed into a straight line. Since the shape information is contained in the slope trend of the lines, it naturally prompts us to use a meaningful symbol to represent each line. For instance, if the slope is positive, and is around 80 degree, then use 'U' to represent it, which means the trend of this line is *sharp up*.

  Thus the original data sequences are generalized by the piecewise linear representation, and the searching space is changed from the continuous space to the discrete space, making the problem much easier to solve, and intuitive for the user to understand.

- **Multi-resolution level of shapes**

  We classify the shapes into different resolution levels. During the time of transferring the sequence data from the continuous space to the discrete space, the symbols are selected from a specific resolution level. The lower the level of the search resolution, the more "blurry" of the searching result we get.

- **A library of primitive shapes**

  We provide a library of primitive shapes with the same length. Combination of these primitive shapes can form any kinds of trends the users want as a query.

- **The length of each unit**

  The length of each unit shape, also called the length of each chopped segment, is a natural segmentation of time. Natural segmentation of time means that for the time-related cube we created, the chop length is determined according to the current level of the time dimension on the time hierarchy. This is just one of the methods to determine the chop length, later we will discuss it in more detail.

Example 5.2.1 For a cube on which we are doing the mining, if the time dimension of it now is at 'month' level, then the chop length is 12. If the time dimension of it now is at 'quarter' level, then the chop length is 4.

- **The query in the subsequence search module**

  The query in the subsequence search module is a slope trend, which is a string of symbols the users give. Each symbol implies a primitive slope trend with a unit length. So a string of symbols in a specific resolution level means the union of the primitive trends at a resolution level. The trend length is the sum of each primitive unit length.

Example 5.2.2 Consider a query trend the users ask: "uussUUUdUU", which means two unit length of slower ups, followed by two stable downs, three sharp ups, one slower down and two sharp ups. Thus totally the query length is 10 unit trend lengths.

R	(Increase)	(0.5, 90)	**F**	(Decrease)	(−90, −0.5)
u	(slower up)	(0.5, 45)	**d**	(slower down)	(−45, −0.5)
u_1	(slower up $_1$)	(0.5, 15)	d_1	(slower down $_1$)	(−15, −0.5)
u_{11}	(slower up $_{11}$)	(0.5, 5)	d_{11}	(slower down $_{11}$)	(−5, −0.5)
u_{12}	(slower up $_{12}$)	[05, 10)	d_{12}	(slower down $_{12}$)	(−10, −5]
u_{13}	(slower up $_{13}$)	[10, 15)	d_{13}	(slower down $_{13}$)	(−15, −10]
u_2	(slower up $_2$)	[15, 30)	d_2	(slower down $_2$)	(−30, −15]
u_{21}	(slower up $_{21}$)	[15, 20)	d_{21}	(slower down $_{21}$)	(−20, −15]
u_{22}	(slower up $_{22}$)	[20, 25)	d_{22}	(slower down $_{22}$)	(−25, −20]
u_{23}	(slower up $_{23}$)	[25, 30)	d_{23}	(slower down $_{23}$)	(−30, −25]
u_3	(slower up $_3$)	[30, 45)	d_3	(slower down $_3$)	(−45, −30]
u_{31}	(slower up $_{31}$)	[30, 35)	d_{31}	(slower down $_{31}$)	(−35, −30]
u_{32}	(slower up $_{32}$)	[35, 40)	d_{32}	(slower down $_{32}$)	(−40, −35]
u_{33}	(slower up $_{33}$)	[40, 45)	d_{33}	(slower down $_{33}$)	(−45, −40]
U	(sharp up)	[45, 90)	**D**	(Sharp down)	(−90, −45]
U_1	(sharp up $_1$)	[45, 60)	D_1	(sharp down $_1$)	(−60, −45]
U_{11}	(sharp up $_{11}$)	[45, 50)	D_{11}	(sharp down $_{11}$)	(−50, −45]
U_{12}	(sharp up $_{12}$)	[50, 55)	D_{12}	(sharp down $_{12}$)	(−55, −50]
U_{13}	(sharp up $_{13}$)	[55, 60)	D_{13}	(sharp down $_{13}$)	(−60, −55]
U_2	(sharp up $_2$)	[60, 75)	D_2	(sharp down $_2$)	(−75, −60]
U_{21}	(sharp up $_{21}$)	[60, 65)	D_{21}	(sharp down $_{21}$)	(−65, −60]
U_{22}	(sharp up $_{22}$)	[65, 70)	D_{22}	(sharp down $_{22}$)	(−70, −65]
U_{23}	(sharp up $_{23}$)	[70, 75)	D_{23}	(sharp down $_{23}$)	(−75, −70]
U_3	(sharp Up $_3$)	[75, 90]	D_3	(sharp down $_3$)	(−90, −75]
U_{31}	(sharp up $_{31}$)	[75, 80)	D_{31}	(sharp down $_{31}$)	(−80, −75]
U_{32}	(sharp up $_{32}$)	[80, 85)	D_{32}	(sharp down $_{32}$)	(−85, −80]
U_{33}	(sharp up $_{33}$)	[85, 90)	D_{33}	(sharp down $_{33}$)	(−90, −85]

H	(Horizontal)	[−0.5, 0.5]
s	(stable down)	[−0.5, 0)
S	(stable up)	[0, 0.5]

Table 5.1: The shape definition table

- **The shape definition table**

  In this system, we chop the data sequences into many chunks of equal length. Each chunk is called a unit. The $(-\frac{\pi}{2}, \frac{\pi}{2})$ space is divided into a number of non-overlapping intervals, each corresponding to a symbol k. The symbols and intervals are kept in the shape definition table. Then we use the corresponding character symbol to represent the trend degree of each unit according to the shape definition table of Table 5.1. Four layers of symbols are designed in the table. The higher layers are divided into some subranges at the lower layers. The two values beside each symbol are degree range values represent the degree intervals belong to these symbols. This idea is demonstrated more clearly in Figure 5.2, in which different levels of shape symbols are expressed at different degree ranges. For example, $R(0.5, 90)$ in Figure 5.2 means if a slope value falls in between 0.5 degree to 90 degree, then use symbol 'R' to represent this slope value. 'R' means "Rise", whereas 'F', which is the opposite of 'R', means 'Fall', and 'H' means 'Horizontal'. 'R', 'F' and 'H' symbols represent the coarsest shape trend. To make the shape trend finer, the degree range value of 'R' can be divided by 2, and we get $U[45, 90)$ and $u(0.5, 45]$, representing *sharp up* and *slow up* respectively. Following the same way, the degree range values of 'U' and 'u' can be further divided by 3 respectively to make the trend shape expressed much finer. Thus users can search the similar trend in different resolution levels.

  Figure 5.2 only shows half of the shape definition of Table 5.1, in which the degree values are positive. The other half is almost the same, except that the degree values are negative.

  We use a shape definition hierarchy tree to express this table and the hierarchy of this table more clearly.

- **The shape definition hierarchy tree**

  In this system, the users are given the choice of selecting to mine different resolution levels of the similarity among sequences. The higher the level the users want, the more "blurry" matching of sequences the users get. Thus to

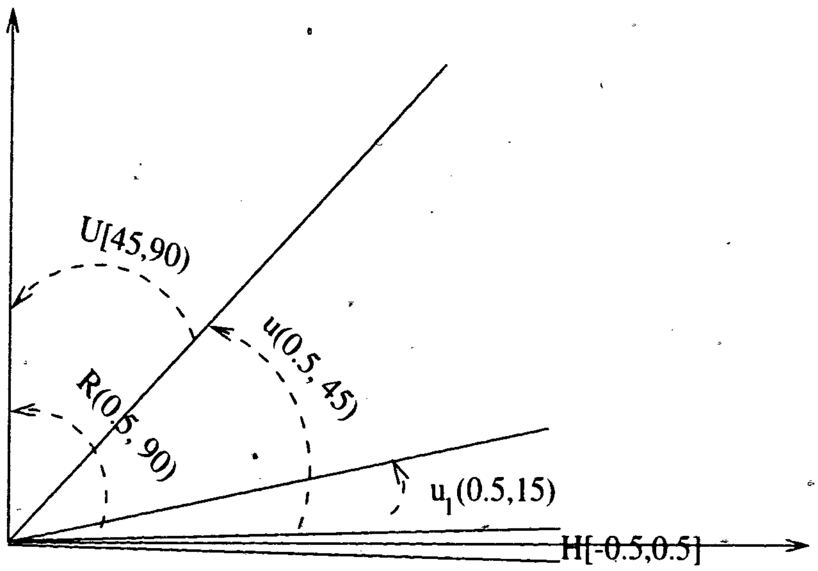


Figure 5.2: The illustration of the shape definition table

implement the similarity searching in multiple levels, we introduce the shape hierarchy tree. It is presented in Figure 5.3. The highest level is *Any*, we call it the 0 level. The second level only has three general shapes: 'R' (Rise), 'F' (Fall), and 'H' (Horizontal). This level is the most "blurry" level. The third level has higher resolution, and the shapes include 'u' (slower up), 'U' (sharp up), 'd' (slower down), 'D' (sharp down), 'S' (stable up), and 's' (stable down). Each shape type is then further divided into three subtypes in the fourth level, and so on.

The purpose of giving the shape definition hierarchically is to allow the user to find the similarity at different resolutions interactively through the user interface. If we look at a sequence with a large "window", we would notice gross features. Similarly, if we look at a sequence with a small "window", we would notice small features. The goal of mining the similarity at multiple-levels is to mine from forest to trees as users specified.

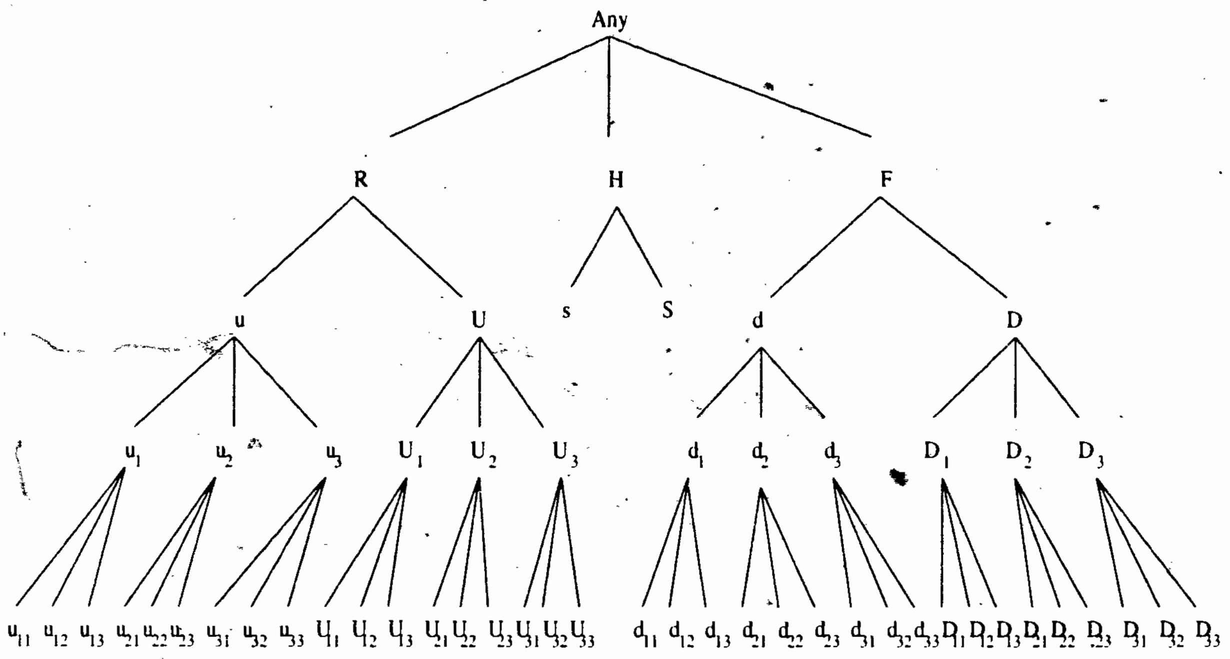

Any
R
H
F
u
U
s
S
d
D
u1 u2 u3 U1 U2 U3
d1 d2 d3 D1 D2 D3
u11 u12 u13 u21 u22 u23 u31 u32 u33 U11 U12 U13 U21 U22 U23 U31 U32 U33
d11 d12 d13 d21 d22 d23 d31 d32 d33 D11 D12 D13 D21 D22 D23 D31 D32 D33

Figure 5.3: The shape definition hierarchy tree

5.3 Algorithm

We now give the formal algorithm for the subsequence search module.

Algorithm 5.3.1 *Finding the similarity subsequences according to the search template : shape.*

Input: (i) A time-series data, (ii) the query shape sequence (query pattern), (iii) the shape definition table and the shape hierarchy tree, (iv) the tolerance threshold k for approximate match, and (v) the matching resolution level.

Output: The sets of all the subsequences that are approximately similar to the query pattern with at most k differences.

Method: Changing the searching space from the continuous space to the discrete space, and implementing multi-level resolution search.

1. Building the cube, which is the summarization of the raw data of the database according to the user's require. The details are illustrated in Section 5.3.1.

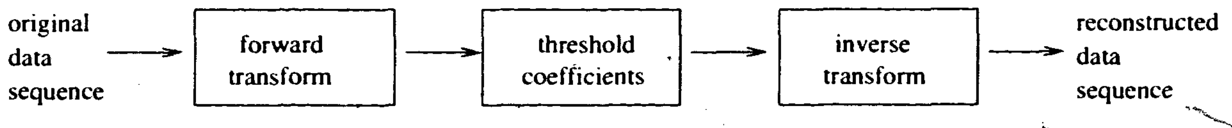


Figure 5.4: Wavelet filter

2. Wavelet filter, which is using wavelet technique to remove the noisy data. Section 5.3.2 describes it in detail.

3. Encoding, which is to transform the data from the continues space to the discrete space. It is discussed in Section 5.3.3.

4. Multi-Level resolution search, which is to find the similarity pattern and allow users doing multi-level resolution search. The details are discussed in section 5.3.4.

These four steps are illustrated in detail in the following subsections.

5.3.1 Building the Cube

Our similarity searching system is created on the basis of the DBMiner cube structure. Cube is a multi-dimensional data structure that is a summarization of the data people are interested in. The purpose of the cube structure is for efficient retrieval of the data which are used by other front end mining tools such as association and similarity mining tools.

Before building the cube, two most important items for the cube should be determined first: dimensions and measures. As we have introduced before, since we are working on the time-series data similarity search, one of the dimensions must be the "time", the selection for other dimensions can be the same as other mining tools [22, 38]. One of the measures can be the time-series data value that you want to find the trend of, such as the closing price, or the sales amount, etc.

5.3.2 Wavelet Filter

Informally, we consider two sequences similar if the non-matching parts are less than the similarity threshold. The small non-matching regions are treated as noisy data and are ignored. Sometimes, there exist sharp jumps, spikes and non-smooth features in the time-related sequences. So the goal is to find a way to make our similarity searching algorithm insensitive to the noisy data as much as possible.

Wavelet filter is just the right tool to filter out the noisy data and exclude the sharp jumps and spikes, to make the final sequence a smoother one.

Using the wavelet analysis technique to filter out noise or short-term fluctuations is presented as Figure 5.4. The meaning of the terms of noise or short-term fluctuations in our application domain is defined as follows:

Definition 5.1 *The noisy data is the short interval gap, in which some very small regions of data has sharp jumps or valleys, but the time intervals are so small that can be ignored without influence the whole sequence trend.*

This algorithm tries to eliminate the noisy data, in other words, this algorithm is for *de-noising* or, more precisely, *coherent structure extraction*. This is a difficult and ill-defined problem, since what is "noise" is not always well defined. We choose to use threshold to quantify it.

The technique of Figure 5.4 works in the following way. When we take the wavelet transform of a data set, we decompose a data set into "averaging" part and "details" part, which is the forward transform step. Some of the resulting wavelet coefficients of the details are small, and they might be omitted without substantially affecting the main features of the data set. The idea of *thresholding* then is to set to zero all coefficients that are less than a particular threshold, which is the thresholding step. The coefficients that are less than a particular threshold are considered as noise. Then we invert the transform to reconstruct the original signal minus the noise, which is the inverse transform step. Wavelet noise-removal has been shown to work well in Figure 5.8 which shows a pair of "before" and "after" of a stock data sequence of S&P index from Jan. 1, 1940 to Dec. 31, 1992, and we can see that the noise is removed and at the same time the basic features of the signal are kept. Besides wavelet methods, other approaches such as the Fourier transform are also possible.

However, our approach is better than the Fourier transform in many aspects which has been discussed in Chapter 4.

Algorithm 5.3.2 *Noisy data removal.*

Input: (i) A sequence of data which is a time-related measure of the constructed cube, and (ii) chop threshold ϵ ($\epsilon < 1$).

Output: A reconstructed data sequence of this time-related measure of the cube.

Method: Wavelet forward and inverse transform.

1. Wavelet forward transform.

   Using the wavelet forward transform method, which is described in Chapter 4, to transform the sequence data from the time domain to the time-frequency domain. The original data is supposed at $V_{j_{max}}$ space, changing the original data from $V_{j_{max}}$ space to $V_{j_{min}}$ space, with j_{max} defined to be 0, and j_{min} determined by the bandwidth calculation, referred to Figure 4.1 and Figure 4.4 in Chapter 4.

2. Thresholds the coefficients.

   (a) To filter out the high frequency part, set the first two levels of the wavelet coefficients of the $W_{j_{max}-1}$ and $W_{j_{max}-2}$ spaces to zero, referred to Figure 4.4.

   (b) Do L^2 energy chop on the coefficients [46], filtering out the small coefficients that less than the de-noising threshold on the spaces from $W_{j_{max}-3}$ to $W_{j_{min}}$, referred to Figure 4.4. These small coefficients are called *non-significant coefficients*. It has the following three sub-steps:

      (i) Find M, the largest absolute value Wavelet coefficients in "mother wavelet" space from $W_{j_{max}}$ to $W_{j_{min}}$.

      (ii) If a coefficient $|d[i,k]|$ is less than $sqrt(\epsilon) * M$, then $d[i,k]$ is set to zero. $d[i,k]$ is the coefficients of "mother wavelet" space at ith resolution level of kth position;

(iii) Continue to do step (ii) truncating any coefficients that meet (ii), which are used in the reconstruction step, until the "mother wavelet" space from $W_{j_{max}-3}$ to $W_{j_{min}}$ are covered.

Using C-like syntax, this step (b) can be written as follows.

```
$M = max|d[i,k]|;$
for ($i = j_{max} - 3; i <= j_{min}; i++$) {
  if $|d[i,k]| < sqrt(\epsilon) * M$
    $d[i,k] = 0;$
  else
    $d[i,k] = d[i,k]$ };
```

3. Wavelet Inverse Transform.

   Wavelet inverse transform to the original time domain $V_{j_{max}}$ from the space $V_{j_{min}}$ referred to Figure 4.2.

Explanation of the Algorithm 5.3.2.

1. In order to delete the parts of sharp jumps and valleys with short intervals of the data sequence, the coefficients of the first two levels of "mother wavelet" are chopped at step 2 (a). Since the data sequence is decomposed into "averaging" part and "details" part, these coefficients, which are chopped, represent the very high frequency information of the original data sequence. Thus this step is very important, since it deletes one kind of the noisy data which have very high frequencies.

2. What we basically do at step 2 (b) is using the L^2 *energy chop on the coefficients* method [46]. The equation is as follows:

$$\begin{array}{lll} If & |c[j,m,k]|^2 < \epsilon * E \quad then & c[j,m,k] = 0, \\ where & E = max\{|c[j,m,k]|^2\} & \end{array}$$

which is equivalent to the following equation:

$$\begin{array}{lll} If & |c[j,m,k]| < sqrt(\epsilon) * M & then\ \ c[j,m,k] = 0, \\ where & M = max\{|c[j,m,k]|\} & \end{array}$$

The basic reason of doing the threshold chopping is that each coefficient of the "mother wavelet" represents the "detail" information about the data sequence at a given location and at a given scale. The coefficients with small values less than a particular threshold represent non-significant part and can be considered as noise. Thus they can be omitted and still can get good quality of approximation of the original data sequence. In other words, the wavelet transform allows us to focus on the most relevant parts of the sequence.

There are other methods of threshold chopping besides L^2 *Energy Chop on the coefficients*, such as based on the *noise standard deviation estimation at each scale* method [18, 39], which is more complicated to implement.

3. The process of noise removal of this algorithm is quite like that of keeping only important coefficients in data compression algorithms. Like the data compression algorithms, for the noise removal algorithms, there are two concepts that we are most interested in, which are `compression ratio` and `relative error`.

Definition 5.2 *The compression ratio is the number of bits the initial data sequence takes to store on the computer divided by the number of bits required storing the compressed data sequence.*

The equation of *relative* L^2 *error* is as follows:

$$E = \frac{(\sum_{k=0}^{n}(org[k] - rec[k])^2)^{1/2}}{(\sum_{k=0}^{n}(org[k])^2)^{1/2}}$$

where n is the number of the data in the sequence, $org[k]$ is the original data sequence at position k, and $rec[k]$ is the reconstructed data sequence at position k.

time level	chop length N
'year'	2
'quarter'	4
'month'	12
'day'	28-31

Table 5.2: The chop length scheme based on the time hierarchy level

5.3.3 Encoding

After the noise-removal of the wavelet filter, the next step is to transform the problem from the continuous space to the discrete space. In other words, change the continuous real values to a limited number of symbols so that it excludes the preprocessing procedures of doing amplitude scaling and offset translation like other algorithms in [2, 18]. It solves the problem of how to compare the sequences in different value domains.

We first divide the one-dimensional time-related sequence into many chunks with equal length, and according to the approximate trend degree of each chunk, using a corresponding symbol at a user-specified resolution level on the shape hierarchy tree to represent each chunk of data. The key is to find the right way of getting appropriate chop lengths in different resolution levels.

Algorithm 5.3.3 *Encoding the data sequence.*

Input: (i) A reconstructed data sequence of a time-related measure of the cube, and (ii) the shape definition table and the shape hierarchy tree.

Output: A string of symbols represent the shape of the data sequence.

Method:

1. Determine the chop length, which is a natural segmentation of `time` according to the `time` concept hierarchy. It has two sub-steps as follows:

   (a) Get the current `time` level on the `time` concept hierarchy.

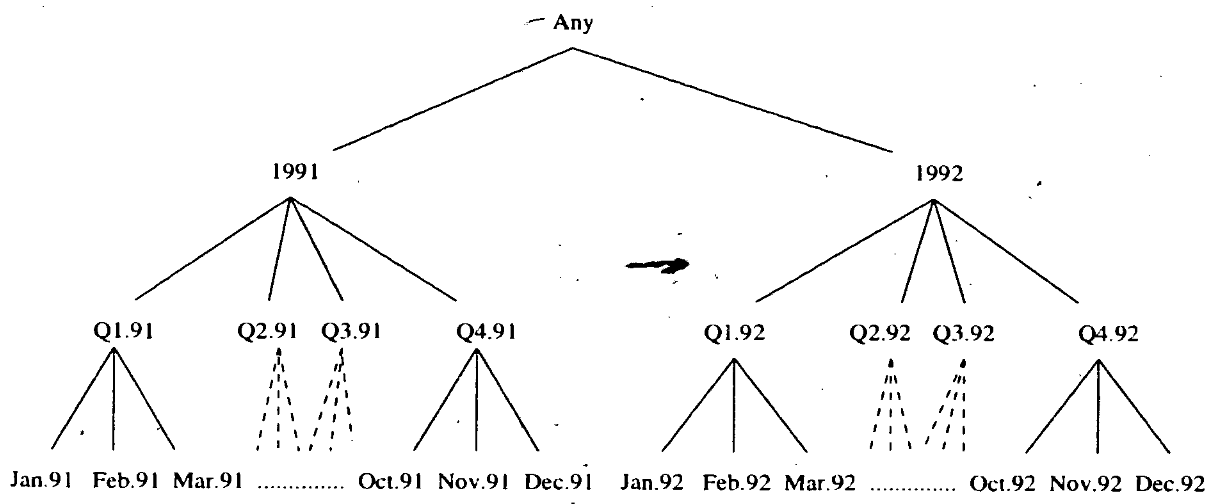


Figure 5.5: A concept hierarchy of the time dimension: (a) with the same numbers of siblings.

   (b) According to the current `time` level and the chop length scheme in Table 5.2, get the corresponding chop length N.

2. To represent each bunch of data of length N, using the `least square` method [7] to get the best linear approximating line $ax + b$ for each bunch of data.

3. Represent each bunch of data by a corresponding symbol.

   (a) Get the slope value of each approximating line.

   (b) Use the corresponding symbol to represent each approximating line according to the shape definition table of Table 5.1.

4. Store the resulted string of symbols in an array.

Explanation of Algorithm 5.3.3.

1. We allow the OLAP operations "drill down", "roll up" and other operations to work on the cube, such as drilling down and rolling up along the `time` dimension of the cube, so that the users can get the similarity information at different `time` granularities. Since the higher the level generated along one dimension, the fewer

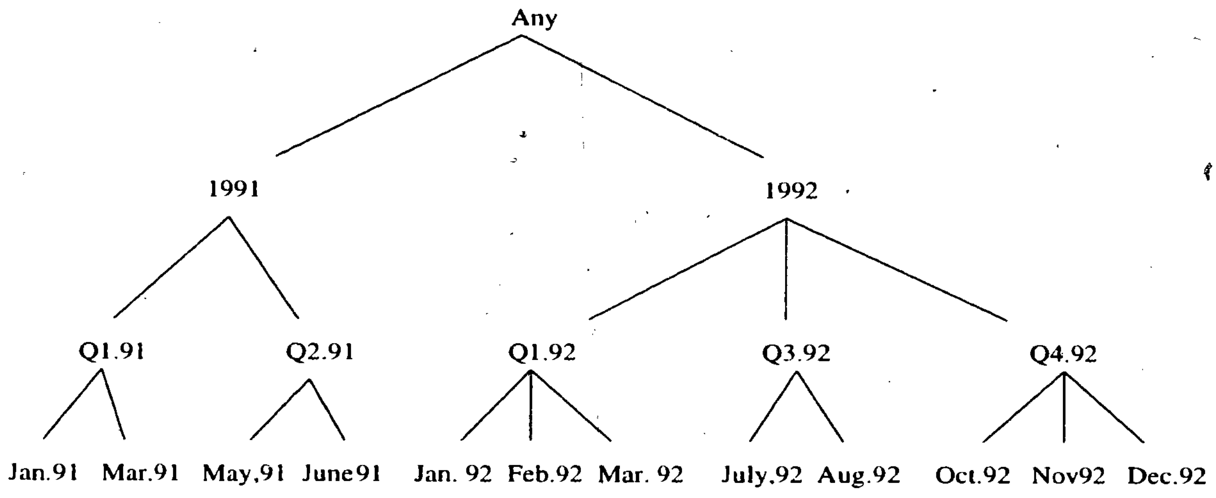


Figure 5.6: A concept hierarchy of the time dimension: (b) with different numbers of siblings.

the number of data are on that dimension, the chop length N should be variant at different `time` granularities, and N must be greater than 2, otherwise we cannot get a approximating line with only one data.

2. There are two methods to determine the chop length N at different `time` granularities.

   (1) The first method is that N is determined by the current concept hierarchy level of the `time` dimension. In other words, the chop length N is a natural segmentation of time based on the current concept hierarchy level of the `time` dimension. For example, if the `time` dimension is at `month` level now, then the chop length N is 12. Table 5.2 is the scheme we used in this method. One disadvantage of this method is that it is not appropriate for one special and complicated occasion as in Figure 5.6, when the `time` hierarchy is not like what we usually meet with that the number of siblings is not the same at a concept level. What we usually run into is shown as in Figure 5.5 that the number of siblings is the same at every concept level. Fortunately, the situations like in Figure 5.6 are very rare in the time-related databases. To handle this kind of

Number of attribute values	Chop length N	The number of symbols
≥ 50000	2000	≥ 25
$10000 \sim 50000$	1000	$10 \sim 50$
$5000 \sim 10000$	500	$10 \sim 20$
$1000 \sim 5000$	250	$4 \sim 20$
$500 \sim 1000$	100	$5 \sim 10$
$100 \sim 500$	25	$4 \sim 20$
$50 \sim 100$	10	$5 \sim 10$
≤ 50	2	≤ 25

Table 5.3: The chop length scheme based on the number of attribute values

`time` hierarchy can be our future work.

(2) The second method is that N is only dependent on the number of attribute values on the current `time` dimension. A possible scheme for the chop length N can be in Table 5.3. For example, if the number of attribute values in current `time` dimension is between 10000 to 50000, then the chop length N is set to 1000 and the number of symbols we get will be between 10 to 50. From the table we can see that the number of symbols after the encoding step will be around 4 to 50 no matter how many of the attribute values on the time dimension, and the chop length is varied based on the current number of attribute values. Thus the scheme looks reasonable. One disadvantage of this method is that it is not natural for the user, since it has nothing to do with the "time" concept hierarchy.

We select the chop length scheme (1) in our algorithm for its natural, intuitiveness and simplicity.

3. The **`least squares`** approach at step 2 is to determine the best approximating line when the error involved is the sum of the squares of the differences between the y-values on the approximating line and the given y-values. Hence, constants a and b must be found that minimize the least squares error:

$$\sum_{i=1}^{10}[y_i - (ax_i + b)]^2 \tag{5.1}$$

Let $ax_i + b$ denote the ith value on the approximating line and y_i the ith given y-value.

The solution of constants a and b is as follows:

$$a = \frac{m(\sum_{i=1}^{m} x_i y_i) - (\sum_{i=1}^{m} x_i)(\sum_{i=1}^{m} y_i)}{m(\sum_{i=1}^{m} x_i^2) - (\sum_{i=1}^{m} x_i)^2} \tag{5.2}$$

$$b = \frac{(\sum_{i=1}^{m} x_i^2)(\sum_{i=1}^{m} y_i) - (\sum_{i=1}^{m} x_i y_i)(\sum_{i=1}^{m} x_i)}{m(\sum_{i=1}^{m} x_i^2) - (\sum_{i=1}^{m} x_i)^2} \tag{5.3}$$

The **least squares** method is the most convenient procedure for determining best linear approximation, and it is a good method from the theoretical point of view. It puts substantially more weight on a point that is out of line with the rest of data but will not allow that point to completely dominate the approximation [7]. There are also other methods to get the linear approximation, such as **minimax** approach and **absolute deviation** approach, etc. It is testified that they are not as good as the **least squares** method.

4. We divide the $(-\frac{\pi}{2}, \frac{\pi}{2})$ space into a number of non-overlapping intervals, each corresponding to a symbol k, and each unit of numbers is now replaced by the symbol k associated with the interval to which it belongs. The symbols and intervals are kept in the **shape definition table**. Since the size of the intervals implies the resolution, to give the user the choice of different levels of resolution, the table is created in hierarchical intervals.

5. The *purpose* of using the **shape definition table** is to provide a small set of shape primitives: unit lines with different slopes. The users can randomly combine these unit lines and form a shape they are interested in. This shape can be used as queries proposed to our system requesting to find out the similar part in the data sequences. This method of generating similarity queries is quite intuitive and originative.

5.3.4 Multi-Level Resolution Search

Now the one-dimensional time-related sequence has been changed into a string of symbols after the above steps. Each symbol in the string implies the trend of the corresponding group of data. Thus a query trend the users ask can be indicated with a combination of such symbols at a specified resolution level. So the problem now is to match the query trend string with the source string, which means it becomes the string matching problem.

There are many algorithms in the *string matching* area, especially in the area of approximate string matching. The approximate string matching task is to find all approximate occurrences of the pattern in the text with at most k differences, given a pattern string, a text string, and an integer k. An approximate occurrence means a substring P' of T such that at most k editing operations (insertions, deletions, changes) are needed to convert P' to P.

Example 5.3.1 Consider the text string T = *bcbacbbb* and the pattern string P = *cacd*, and $k = 2$. The similar pattern found is *cbac* and *bacb*.

To give the user the flexibility to select different similarity levels, we support the multilevel string matching by using the *shape definition hierarchy tree* in Figure 5.3.

The algorithm is as follows:

Algorithm 5.3.4 *Multi-level resolution search*

Input: (i)A string of symbols that represent the shape of the data sequence; (ii) the query shape sequence(query pattern); (iii) the shape definition table and the shape hierarchy tree; (iv) the tolerance threshold k for approximate matching; and (v) the matching resolution level.

Output: The sets of all the subsequences that are approximately similar to the query pattern with at most k differences.

Method:

1. Generalize or specialize the input string of symbols to the corresponding query shape matching level.

2. Use the string matching algorithm: Enhanced Dynamic Programming (EDP) Algorithm [26] to find the similar subsequences referred to Chapter 2.

   Using C-like syntax, the EDP algorithm can be written as follows. The function name is *DoStringMatch.*

```
DoStringMatch(QueryPattern)
{
  $top = k + 1;$                      $//k$ is the threshold for approximation.
  for($i = 0; i <= m; i++$)           // $m$ is the length of the pattern string.
    $h[i] = i;$                       //initialize $h[i]$.
  for($j = 1; j <= n; j++$)           //$n$ is the length of the source string.
  {
    $c = 0;$
    for($i = 1; i <= top; i++$)
    {
  // if the pattern string at $i$ is the same as source string at $j$.
      if($QueryPattern[i] == SourceString[j]$)
        $\epsilon = c;$
      else
// $h[i-1]$ is the entry of $D(i-1, j)$, $h[i]$ is the entry of $D(i, j-1)$, and
//$c$ is the entry of $D(i-1, j-1)$ in Table D 2.1 of Chapter 2.
        $\epsilon = min(h[i-1], h[i], c) + 1;$
      $c = h[i];$
      $h[i] = \epsilon;$
    }
    while($h[top] > k$)
      $top--;$
    if($top == m$)
      Report_Match($j$);              //$j$ is the match position.
    else
      $top++;$
```

```
        }
    }
```

3. Stitching the neighbored similarity result together.

   Using C-like syntax, the stitching algorithm can be written as follows, the function name is *DoStitching*.

```
DoStitching()
{
  // initialize FinalMatch to empty. FinalMatch is the array to store
  //the stitched matching strings.
    FinalMatch = NULL;
  // NumMatch is the number of matching subsequences.
    while($i < NumMatch$)
    {
  // if the matching positions are neighbors, then they should be stitched.
      while($dif(match\_position[i], match\_position[i+1]) == 1$)
      {
  // stitch the neighbor subsequences together.
        $StitchedString = match(i) + match(i+1)$
                              $-overlap(match(i), match(i+1));$
  //Get the final stitched string.
        $FinalMatch = FinalMatch + StitchedString$
                              $-overlap(FinalMatch, StitchedString);$
      }
    }
}
```

4. Scaling the query pattern, continue to do steps 2 and 3, till the scaling factor is four.

Using *C*-like syntax, the scaling algorithm can be written as follows.

```
DoScaling()
{
  for (int scale = 0; scale <= 4; scale ++)
  {
// len_queryPattern is the string length of the query pattern
    for (int i = 0; i < len_QueryPattern; i++)
    {
      for (int j = 0; j <= scale; j ++)//duplicate the symbol at i to j times.
      {
        ScaledQueryPattern + = QueryPattern.GetAt(i);
      }
    }
    DoStringMatch(ScaledQueryPattern); // do step 2
    DoStitching(); // do step 3
  }
}
```

5. Output all approximate occurrences of the query pattern including the stitched and the scaling ones.

Explanation of Algorithm 5.3.4.

1. At step 1, the string of symbols which represents the shape of the original data sequence, should be generalized or specialized to the corresponding query shape level based on the shape hierarchy tree we defined.

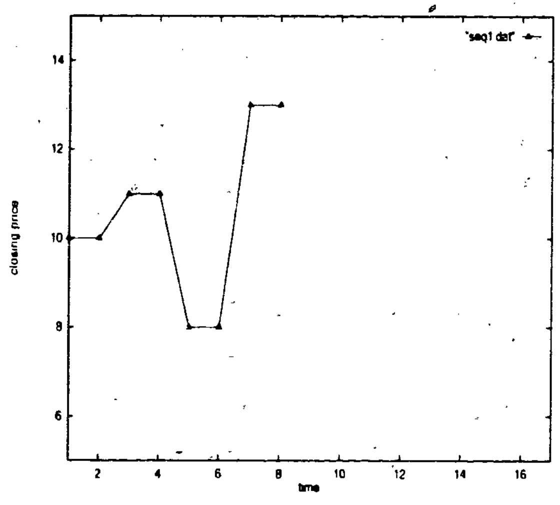
   Example 5.3.2 Assume the shape string we get from the original source data after Algorithm 5.3.3 is "uuuuUUddddDD", which means "four slow ups, followed by two sharp ups, four slow downs and two sharp downs". It is at the third level on the `shape definition hierarchy tree` shown in Figure 5.3. If our query pattern is "RRFFF" which means "two rises followed by three falls", which is at the second level on the `shape definition hierarchy tree` shown in Figure 5.3. They are at different resolution levels (the shape string of the original source data has higher resolution than the query pattern string), thus they cannot be compared with each other directly. So the shape strings of the original source data should be generalized to a lower resolution level. After the generalization of the original shape string "uuuuUUddddDD", we get "RRRRRRFFFFFF". The comparison result we get is that the similar part with the query pattern is from position 5 to position 9 in the original shape string.

2. At step 2, we use the Enhanced Dynamic Programming (EDP) method to get the similar subsequences. It is an edit distance-based approach introduced in Chapter 2. This approach is applied for matching sequences of different lengths which computes the minimum number of operations (deletions, changes and insertions) required to change a sequence into another one. The detail information can be referred to Chapter 2.

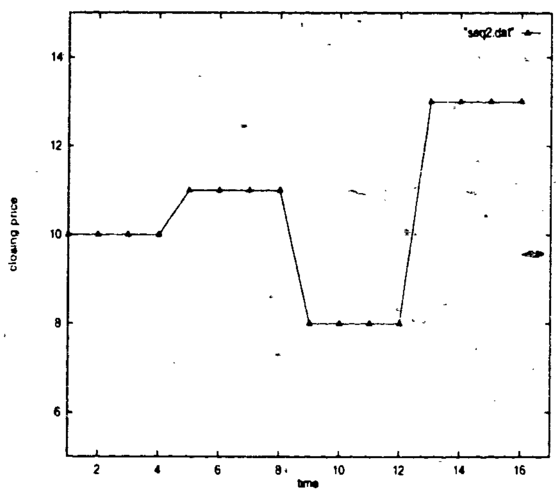
3. Usually the results of matching positions are likely to be neighbors to each other, especially when the shape symbols of the query pattern are with consecutively the same symbols, such as "ddd"(three slow downs). In this case, we usually like to stitch them together. Actually, it is a special case of scaling problem. Following is a stitching example.

Example 5.3.3 If the query is to find the duration with the sales trend of mini-vans is "dd" (two slow downs) between Jan. 1940 to Jan. 1996. The query pattern here is two consecutively same symbols. After the subsequence search module while the searching level is at **year** level, the searching results are as follows :

"from 1942 to 1943, the similar pattern is 'dd';
from 1943 to 1944, the similar pattern is 'dd';
from 1944 to 1945, the similar pattern is 'dd' ".



(a)



(b)

Figure 5.7: (a) Time sequence $\vec{s} = (10, 10, 11, 11, 8, 8, 13, 13)$, and (b) time sequence $\vec{p} = (10, 10, 10, 10, 11, 11, 11, 11, 8, 8, 8, 8, 13, 13, 13, 13)$.

After the stitching algorithm, the searching result is :
"*from 1942 to 1943, the similar pattern is 'dd';*
from 1943 to 1944, the similar pattern is 'dd';
from 1944 to 1945, the similar pattern is 'dd';
from 1942 to 1945, the similar pattern is 'dddd' ".

4. At step 4, scaling problem is considered in our algorithm, an example of scaling problem is shown in Example 5.3.4.

Example 5.3.4 Consider two time sequences in Figure 5.7, $\vec{s} = (10, 10, 11, 11, 8, 8, 13, 13)$ and $\vec{p} = (10, 10, 10, 10, 11, 11, 11, 11, 8, 8, 8, 8, 13, 13, 13, 13)$. $\vec{s}$ and $\vec{p}$ could be the closing price of two stocks. A typical query is "is $\vec{p}$ similar to $\vec{s}$?". The sequence $\vec{p}$ is twice as long as $\vec{s}$, so they cannot be compared directly. If the time axis of $\vec{s}$ is scaled by 2, i.e., every value "v_i" is replaced by "v_i, v_i", the resulting sequence will be identical to $\vec{p}$. This operation is usually called scaling.

According to the above example, we allow scaling along the time axis in our subsequence search module. Following is an example of having this scaling capability.

Example 5.3.5 If the query is to find the duration with the sales trend of mini-vans is "UD" (one sharp up followed by one sharp down) between Jan. 1940 to Jan. 1996. The query pattern here is not with consecutively same symbols as example 5.3.3. After the subsequence search module while the searching level is at **year** level, the searching result is as follows :
"from 1945 to 1946, the similar pattern is 'UD' ".
After the scaling algorithm, the searching result is as follows:
"from 1945 to 1946, the similar pattern is 'UD' ";
from 1944 to 1947, the similar pattern is 'UUDD' ";
from 1943 to 1948, the similar pattern is 'UUUDDD' ".

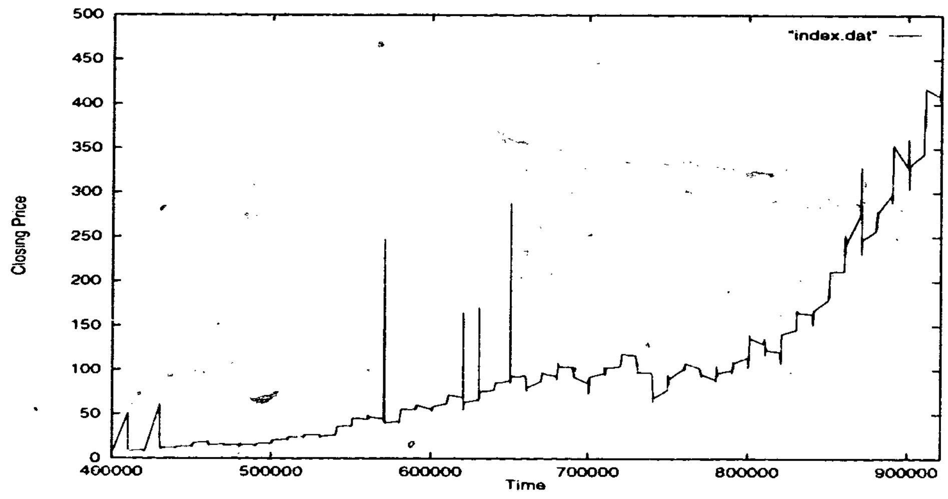
5.4 Experiment Results

We implement our algorithm on top of the DBMiner cube [22, 38] developed in our lab. The data we run are from a stock data sequence called S&P index obtained from the www site "http://www.isse.gmu.edu:80/ jllin/mining/data.html", which is shown in Figure 5.8, x axis is the time from Jan. 1, 1939 to Dec. 31, 1992, y axis is the closing price at the time.

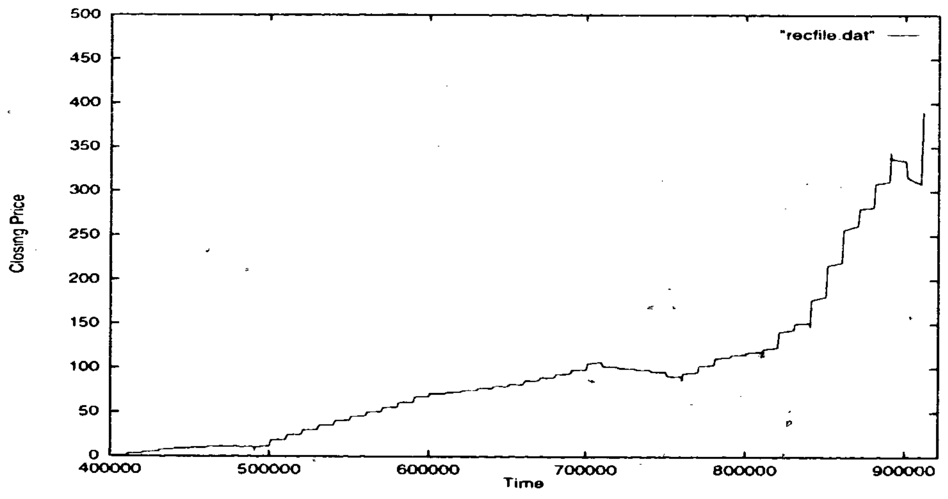
In this section we first introduce the execution of the *subsequence search algorithm* Algorithm 5.3.1 step by step using the real stock data S&P index which is time-related, and then we discuss the performance of the algorithm.

5.4.1 Execution Experiment

To get the similarity shape information, we process from the following four sub-steps according to Algorithm 5.3.1:



(a) The original Data



(b)After the Wavelet Filter

Figure 5.8: A stock data sequence of S&P index from Jan. 1, 1940 to Dec. 31, 1992, with 40 : 1 compression ratio, and 15.37% relative L^2 error, with $\epsilon = 0.01$ after the reconstruction.

1. Building the cube. For this sample database, the cube can have only one dimension, which is "time", and no other dimensions are needed in this case. The "closing price" of the stock can be a measure of the cube. The drill down, roll up and other OLAP operations can be operated on the cube. Along the "time" dimension, one can roll up from "month" to "quarter", or drill down from "year" to "quarter" according to the hierarchical knowledge for the "time" dimension.

2. Wavelet filter. We use the wavelet technique as a filter to filter out noisy data, and making the data smoother so that the later steps not have to worry about the noisy data, such as sharp jumps and valleys. The effect of the wavelet filter on this S&P index data is shown in Figure 5.8.

3. Encoding. The sequence data are changed from the continuous space to the discrete space at this step, in other words, we design a shape language in which symbols are used to represent the slope trends, changing the sequence data to a string of symbols which reflect the overall trend of the sequence data. For this example, the shape of the S&P index stock data at the "year" level and at the third resolution level of the shape hierarchy, is "uuuuuuuuuuuuuuuusSsduuU-UuU". The length of each symbol implies two years, so use these 26 symbols to represents the 52 years data from 1940 to 1991. The meaning of this string of symbols is *"In the first 32 years, which is from 1940 to 1971, is all the way slow ups; there is a stable down in 1972 and 1973, a stable up in 1974 and 1975, and another stable down in 1976 and 1977; then followed by a slow down in 1978 and 1979, two slow ups from 1980 to 1983, two sharp ups from 1984 to 1987, one slow up in 1988 and 1989, and finally a sharp up in 1990 and 1991."* Thus the overall shape character of the whole sequence is represented so intuitively by such a short string of symbols.

4. Multi-level resolution search. At this stage, the similarity searching problem is changed to the string searching problem. So the searching query can be a string of symbols representing shapes, which is very intuitive for the users to understand.

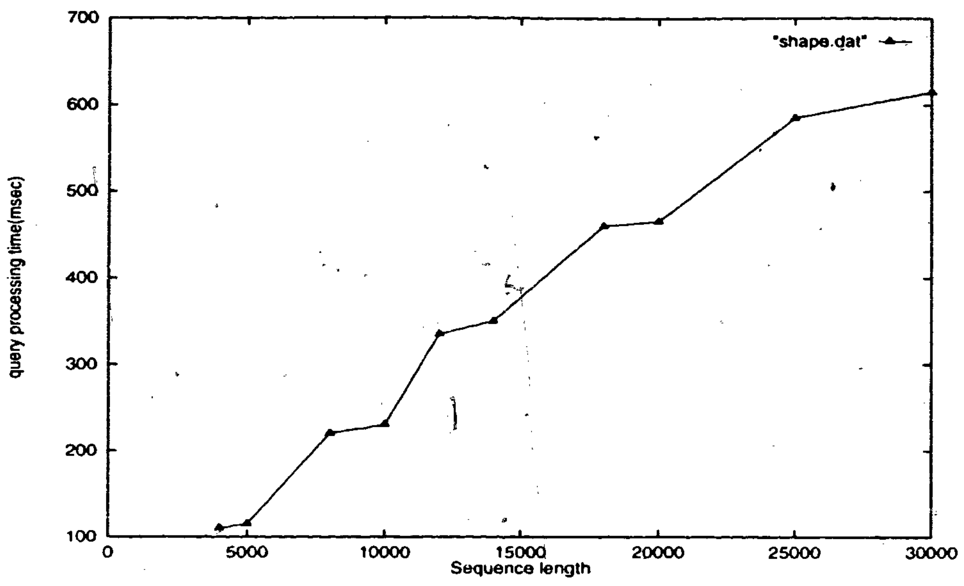


Figure 5.9: The test result of subsequence search module

- Given a searching template query Q, such as "sss"(3 stable downs), and a tolerance threshold $k = 1$, which means allow one symbol to be different in the final searching result. After the similarity searching process, the answer we get is "From 1972 to 1977 there is a match string 'sSs' ";
- Given another searching template query $Q =$"uuuuu"(five slow ups), with $k = 0$, the answer is "From 1940 to 1971 there is a match string 'uuuuuuu-uuuuuuuuu' ", we use the stitching algorithm here to stitch the neighbored similar patterns together.

5.4.2 Scalability Study

The subsequence search module is composed of four steps: building the cube, wavelet filter, encoding and multi-level resolution search. Since building the cube step is outside the scope of our discussion, we will not include this step in our computation of the execution time here. Thus the execution time we compute will start from the wavelet filter step. We have described in Chapter 4 that the total execution time of the wavelet filter step is $O(MN)$, where M is the filter length, and N is the sequence length. In the encoding step, we use the least square approach, and it is easy to see the execution time is $O(N)$, where N is the sequence length. For the last step of multi-level resolution search step, the major sub-steps is the EDP (Enhanced Dynamic Programming) string matching algorithm, and it works in average time $O(kN)$ [26], where k is the matching threshold and N is the sequence length. Since M and k are always very small that can be omitted, Thus we can say the execution time of the subsequence search module is of order $O(N)$, where N is the length of the sequence.

The original sequence length of S&P index is 648. To testify the implementation speed, we duplicate the data to different lengths. We varies the length of the sequence from 600 to 30000. Figure 5.9 shows the execution result. The execution time almost linearly increases with the increasing of the sequence length.

Chapter 6

Whole Sequence Search Module

6.1 Problem Statement

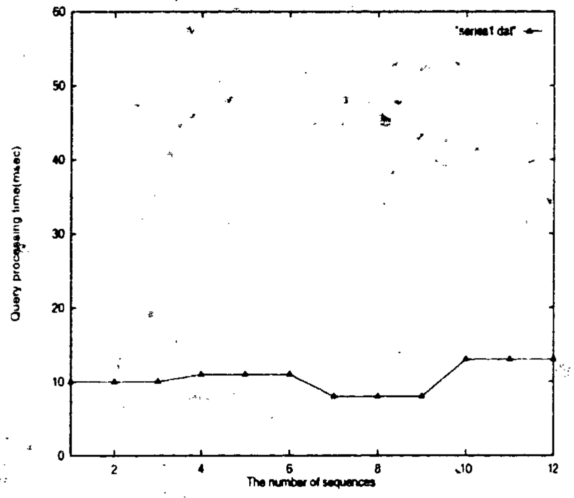
The whole sequence search module is designed to solve the problem of "First(All)-Occurrence Whole Sequence Matching", which is "given a query time-series data Q and a set of time-series data sequence with equal length as Q, finding first (all) of the sequences that match Q approximately".

The main difference between the whole sequence search module and the sub-sequence search module described in Chapter 5 is the assumption of the problem. The assumption of the **subsequence search** module is that only one data sequence is given, and the search template of query Q is a shape composed of a meaningful string of symbols. Whereas, in the **whole sequence search** module, a bunch of data sequences with the same length n are given, and the search query Q is a sequence of data of length n. Thus they are for different domain problems.

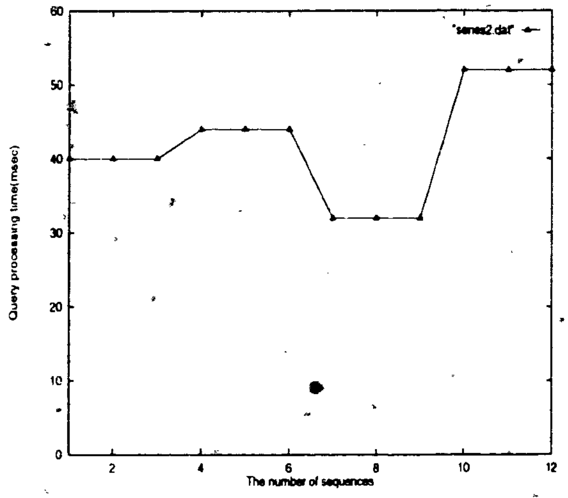
An example of this domain problem is as follows:

Example 6.1.1 *Given the stock price of Microsoft in the duration of 1980 to 1996, and a set of stock prices of other companies in the same duration, find the companies whose stock price fluctuations resemble Microsoft's.*

In the following sections, the terminology used in this domain and the algorithms we designed for this module will be introduced.



(a)



(b)

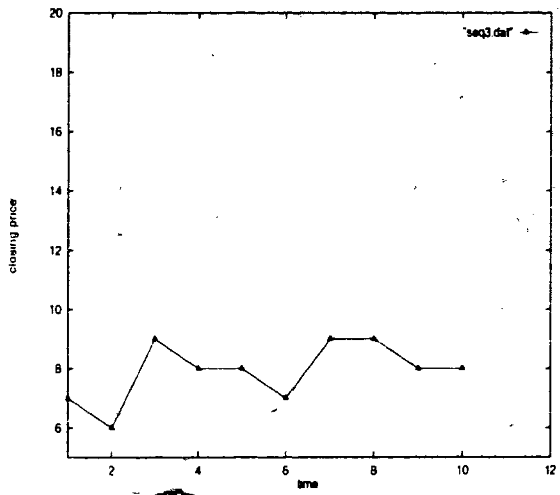
Figure 6.1: (a) Time sequence $\vec{s} = (10, 10, 10, 11, 11, 11, 8, 8, 8, 13, 13, 13)$, and (b) time sequence $\vec{p} = (40, 40, 40, 44, 44, 44, 32, 32, 32, 52, 52, 52)$.

6.2 Terminology

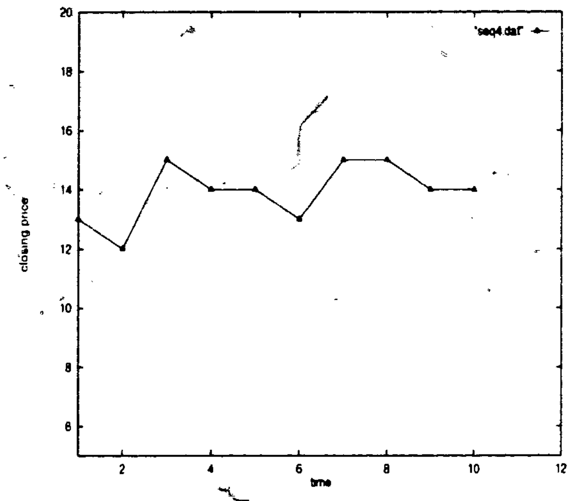
- **Similarity definition.** Before we seriously discuss what is similarity, let us look at the following motivating examples:

Example 6.2.1 Consider two time sequences in Figure 6.1, $\vec{s} = (10, 10, 10, 11, 11, 11, 8, 8, 8, 13, 13, 13)$ and $\vec{p} = (40, 40, 40, 44, 44, 44, 32, 32, 32, 52, 52, 52)$. $\vec{s}$, and $\vec{p}$ could be the closing price of two stocks. A typical query is "is $\vec{p}$ similar to $\vec{s}$?". Each value of sequence $\vec{p}$ is four times as large as that of $\vec{s}$, so they cannot be compared directly. If each value on $\vec{s}$ is scaled by 4 along the y axis, i.e., every value "v_i" is replaced by "$4v_i$", the resulting sequence will be identical to $\vec{p}$. This operation is usually called `scaling`.

Example 6.2.2 Consider two time sequences in Figure 6.2, $\vec{s} = (7, 6, 9, 8, 8, 7, 9, 9, 8, 8)$ and $\vec{p} = (13, 12, 15, 14, 14, 13, 15, 15, 14, 14)$. Suppose they are two time sequences that correspond to the closing prices of two stocks. Is "$\vec{p}$ similar to $\vec{s}$?". The value of sequence $\vec{s}$ is around 8, whereas the value of $\vec{p}$ is around 14, but they go up and down in exactly the same way. If every value in the



(a)



(b)

Figure 6.2: (a) Time sequence $\vec{s} = (7, 6, 9, 8, 8, 7, 9, 9, 8, 8)$, and (b) time sequence $\vec{p} = (13, 12, 15, 14, 14, 13, 15, 15, 14, 14)$.

sequence $\vec{s}$ is added by 6, the resulting sequence will be identical to $\vec{p}$. This operation is usually called **shifting**.

The two sequences in each of the above two examples are not close to each other in an Euclidean sense, but a good similarity search model should allow **scaling** and **shifting** one of the sequences to match another sequence. Combinations of scaling and shifting are shape-preserving transformations, known as *similarity transformations* in the mathematical field of Transformational Geometry [32, 16].

Definition 6.1 *A similarity transformation $T_{a,b}$ over n-sequences is by mapping each element x_i to $a * x_i + b$. $(a, b) \in [\mathcal{R}^+ \times \mathcal{R}]$.*

We restrict $a > 0$, which implies that a sequence symmetric to X w.r.t. the x-axis is not considered similar to it.

Definition 6.2 *Let D be a distance metric between sequences, we say that X is approximately similar to Y if there exist some $(a, b) \in [\mathcal{R}^+ \times \mathcal{R}]$ and $\epsilon \geq 0$, such that $D(X, T_{a,b}(Y)) \leq \epsilon$ [16].*

Definition 6.3 *An approximate similarity class is the set of sequences approximately similar to a given one. This set of sequences constitutes an equivalence class called similarity class. We shall denote the similarity class of X by X^*.*

In a transformation $T_{a,b}$, a is called the *scaling factor* and b is the *shifting factor.* If a is 1, the transformation is a pure *shift*; if b is 0, it is a pure *scale.* The identity transformation is a pure shift; the product of two scales is also a scale. From these, we conclude that the set of all scalings of a given sequence is an equivalence class. The same is true of the set of shiftings.

Definition 6.4 *The Euclidean distance between two sequences S_1 and S_2 is:*

$$D_E(S_1, S_2) = (\sum_{1 \leq i \leq l} (S_1[i] - S_2[i])^2)^{1/2}.. \tag{6.1}$$

In the similarity search area, the Euclidean distance is a general concept used as the similarity distance metric. We use the Euclidean distance as the distance metric in this module.

6.3 Algorithms

Our algorithm is focused on the processing of the time series that is a sequence of real numbers, each representing a value at a time point. Two approaches will be introduced in this section as follows.

6.3.1 Brute-Force Approach

A brute-force approach is a naive approach for solving the whole sequence matching problem which is to compute the Euclidean distance (or other kinds of distance) between any two time sequences, and call two sequences similar if their distance is less than a user-defined threshold. The algorithm is formally illustrated as follows:

Algorithm 6.3.1 *A Brute-Force whole sequence similarity search algorithm.*

Input: (i) A query sequence Q, and a set of n sequences S_i with the same length l as Q, and (ii) distance threshold ϵ.

Output: The set of sequences which is approximately similar to the query sequence Q.

Method:

Using C-like syntax, the steps can be written as follows.

```
// Assume X is the similarity class.
  $X = NULL$;        // initialize X.
  for ($i = 0; i < n; i++$)
  {
//compute the Euclidean distance of Q and $S_i$.
      $D_E(Q, S_i) = (\sum_{l \leq j \leq 0}(Q[j] - S_i[j])^2)^{1/2}$
//if $D_E \leq \epsilon$ then Q and $S_i$ are similar.
      if ($D_E(Q, S_i) \leq \epsilon$)
          $X = X + S_i$;        // put $S_i$ into the similarity class.
  }
//X contains all the sequences that are approximately similar to query Q.
  output(X);
```

Explanation of Algorithm 6.3.1.

1. In this algorithm, every sequence S_i is compared with the query sequence Q through computing the Euclidean distance. If the distance of Q and S_i is less than the threshold ϵ, then put S_i into X set, which is the similarity class set.

2. The time sequences are usually very long, so this algorithm can be time consuming if without some preprocessing. Another problem with this algorithm is that it cannot solve the scaling and shifting problems as in Example 6.2.1 and 6.2.2, which are not close to each other if using this Brute-Force algorithm. A good algorithm should be designed to allow scaling and shifting one of the sequences to match another sequence.

3. This algorithm doesn't have any preprocessing of the noisy data.

6.3.2 A Feasible Approach

To address the disadvantages of the Brute-Force algorithm, we apply some transformations before computing the Euclidean distance.

In this section we demonstrate how our approach can be used to eliminate noise or short-term fluctuations and shift or scale the data before computing Euclidean distance.

Algorithm 6.3.2 *A feasible whole sequence similarity search algorithm.*

Input: (i) A query sequence Q, and a set of N sequences with the same length as Q, and (ii) a distance threshold $\epsilon \geq 0$.
Output: The set of sequences which is approximately similar to the query sequence Q.
Method:

1. Normalization: transform the query sequence and the set of series into the normal forms. This transformation is a similarity transformation as defined in Definition 6.1.

   Using C-like syntax, this step can be written as follows.

```
for(int $i = 0; i < N; i++$)//$N$ is the number of sequences
{
  $mean(S[i]) = ComputeMean(S[i]);$    // compute the average of $S_i$.
  $std(S[i]) = ComputeStd(S[i]);$    // compute the standard deviation of $S_i$.
  for (int $j = 0; j < len(S[i]); j++$)    //len(S[i]) is the length of S[i].
  {
    $NormS[i][j] = (S[i][j] - mean(S[i]))/std(S[i]);$    //compute the
                                   //normalization for each value on the sequence $S_i$.
  }
}
```

2. Wavelet transform: transform each sequence to a coarser version of the original version.

   (i) Transform the normalized data sequences from the initial space $V_{j_{max}}$ to a coarser space $V_{j_{mid}}$ which is the middle space between the initial space and the coarsest space $V_{j_{min}}$.

   (ii) Only keep the coefficients of the $V_{j_{mid}}$ space, delete the coefficients of the "mother wavelet" spaces.

3. Compute the Euclidean distance of the coarser version.

   Using C-like syntax, this step can be written as follows.

```
// Assume $C_{S_i}$ is the coefficients of sequence $S_i$ at $V_{j_{mid}}$ space.
// Assume $C_Q$ is the coefficients of the query sequence $Q$ at $V_{j_{mid}}$ space.
// $len_{j_{mid}}$ is the length of sequences at space $V_{j_{mid}}$, and $len_{j_{max}}$ is the
//original length of sequences at the initial space $V_{j_{max}}$.
    $X = NULL$;      // $X$ is the similarity class for $Q$.
    for(int $i = 0; i < N; i++$)      //$N$ is the number of sequences.
    {
      $len_{j_{mid}} = len_{j_{max}}/2^{j_{mid}}$;
      int $sum = 0$;
      for(int $m = 0; m < len_{j_{mid}}; m++$)
           $sum = sum + (C_Q[m] - C_{S_i}[m])^2$;
      $D_W = (sum)^{1/2}$;   //compute the Euclidean distance.
      if ($D_W <= \epsilon$)   //if $D_W \leq \epsilon$ then $Q$ and $S_i$ are similar
        $X = X + S_i$;   // put $S_i$ into the similarity class.
    }
```

Explanation of Algorithm 6.3.2.

1. To account for scaling and shifting, we normalize the sequence values of each sequence and form a new set of normalized sequences at step 1. Let's look at some definitions here [16]:

   Definition 6.5 *An n-sequence X is a sequence $\{x_1, \cdots, x_n\}$ of real numbers. Each n-sequence X has an average mean(X) and a standard deviation std(x):*

   $$\begin{aligned} mean(X) &= (1/n) \sum_{1 \le i \le n} x_i; \\ std(X) &= ((1/n) \sum_{1 \le i \le n} (x_i - mean(X))^2)^{1/2}. \end{aligned}$$

   Definition 6.6 *An n-sequence X is normal if $mean(X) = 0$ and $std(X) = 1$.*

   Given any data sequence X, $\nu(X)$ denotes the normal form of X^*, where X^* is the similarity class of X. Since a similarity class has exactly one normal form [16], $\nu(x)$ is unique for a similarity class. If $mean(X)$ is the average of X, and $std(X)$ is the deviation of X, then $X = std(X) * \nu(X) + mean(X)$. Therefore, we can compute $\nu(X)$ from X by the inverse transformation:

   $$\nu(X) = \frac{X - mean(X)}{std(X)} \tag{6.2}$$

   Thus, the normalization procedure is the reverse procedure of the **similarity transformation** according to Definition 6.1.

2. At step 2, the bandwidth can be determined before doing the wavelet transformation given a finite length of sequence, referred to Chapter 4. The bandwidth Δj is $\lfloor \log_2(n-1) \rfloor$, where n is the length of the sequence. Thus we define the middle level to be $j_{mid} = \lceil \Delta j / 2 \rceil$.

   We assume the original sequence is at the $V_{j_{max}}$ space, Figure 6.3 shows the original data sequence at $V_{j_{max}}$ space decomposed into subspaces till the $V_{j_{mid}}$ space, and the relationship of $V_{j_{max}}$ with the subspaces is : $V_{j_{max}} = V_{j_{mid}} \oplus W_{j_{mid}} \oplus W_{j_{mid}+1} \oplus \cdots \oplus W_{j_{max}-1}$.

$$\begin{tikzcd}
 & V_{j_{max}} \arrow[d] \arrow[dr] & \\
 & V_{j_{max}-1} \arrow[d, dashed] \arrow[dr, dashed] & W_{j_{max}-1} \\
{} \arrow[r, Rrightarrow] & V_{j_{mid}} \arrow[d, dashed, no head] \arrow[dr, dashed] & W_{j_{mid}} \\
 & V_{j_{min}} & W_{j_{min}}
\end{tikzcd}$$

Figure 6.3: From the $V_{j_{max}}$ space transforms to the $V_{j_{mid}}$ space

We have discussed in Chapter 4 that the coefficients in space V is the coarser part and the coefficients in space W is the detail part of the original data sequence. The lower the level of space V, the coarser of the original sequence. The coarsest space $V_{j_{min}}$ space has only one coefficient, which is the average value of the original sequence. Since the middle level $V_{j_{mid}}$ is not too coarse and not too fine, we select to compute distance of the coefficients at the $V_{j_{mid}}$ space level. We do not care about the detail part at the W spaces, since they can be treated as noisy data. What we are interested in is the overall trend of the sequences, and the coefficients at $V_{j_{mid}}$ are just what we need to represent the overall trend of the sequences.

The lower the level of space V, the fewer the number of coefficients, thus the number of coefficients at $V_{j_{mid}}$ level will be $n/2^{j_{mid}}$, where n is the original length of each sequence. Thus the number of coefficients to be computed is decreased sharply at the middle level.

3. After getting the coefficients of each sequence at space $V_{j_{mid}}$, we compute the Euclidean distance of these sequences according to equation (6.1) at step 3. If the Euclidean distance between the query sequence Q and S_i is less than the threshold ϵ, then put S_i into the similarity class set X.

   Any proper distance metric D for N sequences can be used instead of D_E. Since our data is discrete data, and the Euclidean distance is a standard distance metric in discrete data processing, we chose to use it here.

4. Actually, if we skip step 2, only do step 1 and step 3, the similarity search can also be implemented. In other words, after the normalization of the data sequences at step 1, the Euclidean distances of these normalized sequences can be computed. If the Euclidean distance is less than a user-defined threshold, then they are said to be similar. This Euclidean distance is called *similarity distance* which will be introduced later. This method works fine for smooth sequences which have no noisy data. However, in real life, it is very common to have short intervals of sharp jumps and valleys in the time-related data and the Euclidean distance metric is very sensitive to the noisy data, it is very necessary to add step 2 to solve the noisy data problem. Let us look at the following examples.

   Example 6.3.1 *In Figure 6.4(a), the overall trend of two sequences* `series1` *and* `series2` *are similar from the human eye. Actually every value in* `series2` *is three times of series1 and added by two. Thus after the similarity transformation also called the normalization of step 1, they are overlapped as can be seen in Figure 6.4(b). This is the no noisy data occasion. The Euclidean distance in this example is zero.*

   Example 6.3.2 *In Figure 6.5(a), the values of* `series1` *and* `series2` *are almost the same as Figure 6.4(a) except that there is one noisy data on* `series2`. *The Euclidean distance metric is very sensitive to the noisy data, only one short interval sharp jump can make the two sequences not similar in the Euclidean sense. After the normalization step these two sequences are shown in*

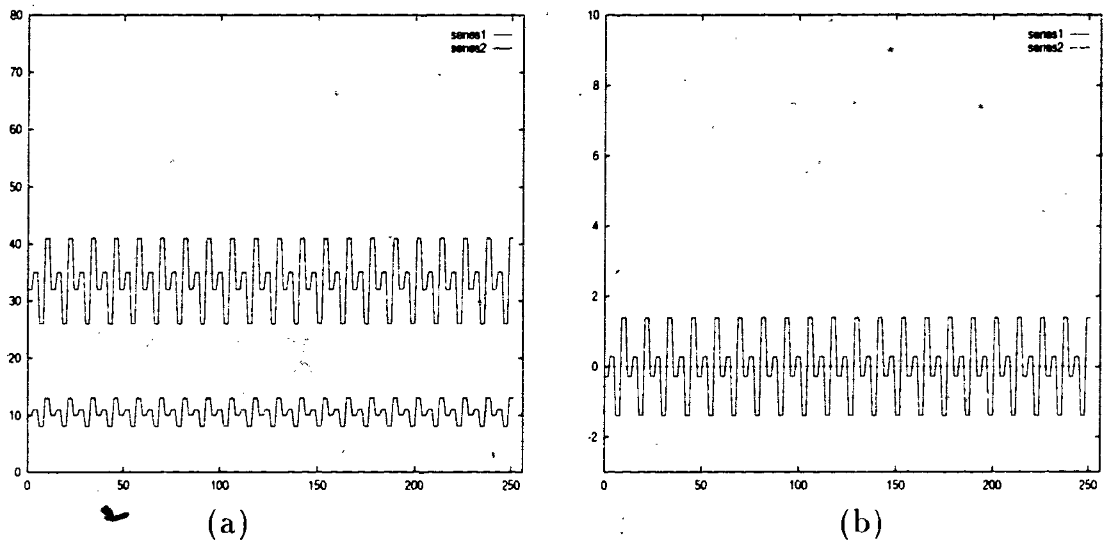


Figure 6.4: (a) The original sequences of series1 and series2, and (b) after normalization $De = 0$.

Figure 6.5(b). It can be clearly seen that the noisy data still there after the normalization, and the Euclidean distance after the normalization is 9.1476*, if we set threshold ϵ to* 3*, then these two sequences is dissimilar if without step 2. But, after we add step 2 in, the noise are decreased greatly, and the Euclidean distance after the step 2, becomes* 2.398*. It is less than the threshold* 3*, and we get the answer that* `series1` *and* `series2` *are similar. This answer is just what we want.*

5. The threshold ϵ can be varied according to the length of the data sequences. The longer of the sequences, the bigger value of the threshold ϵ should be.

6.3.3 Validity and Accuracy of Algorithm 6.3.2

In the similarity search area, the analysis of validity and accuracy of the algorithm is very important. Otherwise, people cannot evaluate how good the algorithm is. In this section we will give some descriptions about the validity and accuracy of Algorithm 6.3.2.

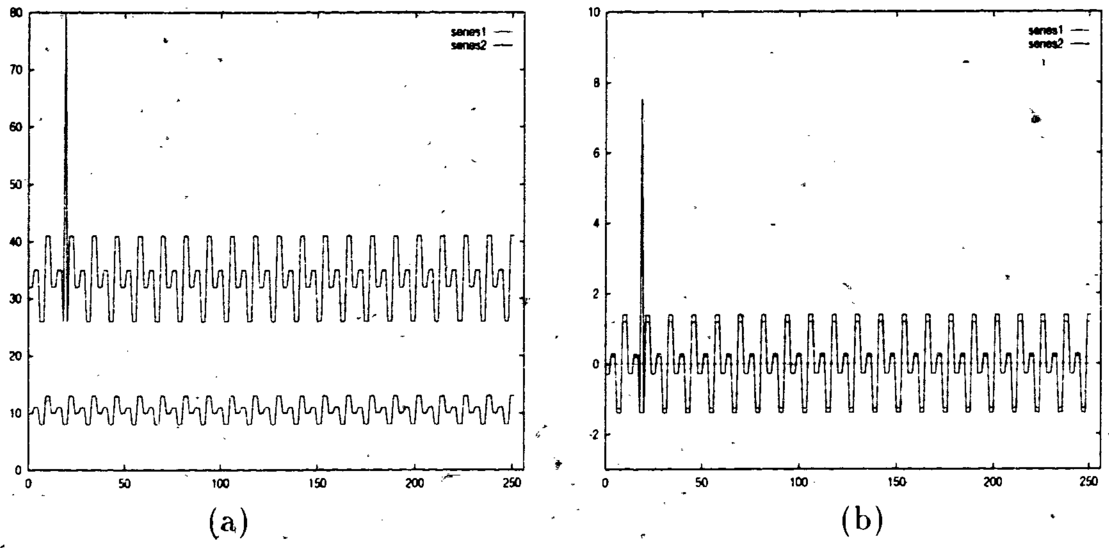


Figure 6.5: (a) The original sequences of series1 and series2, and (b) after normalization De= 9.1476, after wavelet Dw=2.398.

We have said before that after step 1 of normalization of the sequences, we can skip step 2 of wavelet transform, and directly do step 3 of computing the Euclidean distance. We define the *similarity distance* is [16]:

$$D_S(X,Y) = D_E(\nu(X),\nu(Y)), \tag{6.3}$$

where X and Y are two data sequences, $\nu(X)$ is the normalization of sequence X. Sequences with different scales and shifts cannot be compared without the normalization procedure. Other algorithms [16, 2] are developed on the basis of the normalization step, to make the algorithm more efficient. Thus the similarity distance as described in equation (6.3) is used as a criterion to analysis the validity and accuracy of the similarity algorithms.

(i) Validity

To establish the validity of this algorithm, we need to show that

$$D_S(X,Y) \leq \epsilon \Longrightarrow D_W(C(X),C(Y)) \leq \epsilon, \tag{6.4}$$

where D_W is the Euclidean distance that we get from algorithm 6.3.2, and $C(X)$ is the coefficients of the sequence X at $V_{j_{mid}}$ space.

As we know, $C(X)$ is either the average value of two neighbor values as in Haar wavelets, or the sum of some neighbor values as in other wavelet transformations like Coiflets, in which each value is given different weights and the total of the weights is 1 at each different scales.

To prove equation (6.4), we just need to prove that $D_W \leq D_S$. To make things easy, we use the Haar wavelet transform in which V space coefficients are the average value of two neighbor values, and W space coefficients are the half of the difference of two neighbor values. If we can prove Haar wavelets, we can prove other kinds of wavelets transforms.

For the normalized sequences $X(a_0, a_1, \cdots, a_n)$ and $Y(a'_0, a'_1, \cdots, a'_n)$, $D_S = (\sum_{k=0}^{n}(a_k - a'_k)^2)^{1/2}$, and $D_W = (\sum_{k=0}^{n/2}(\frac{a_{2k}+a_{2k+1}}{2} - \frac{a'_{2k}+a'_{2k+1}}{2})^2)^{1/2}$, where $\frac{a_{2k}+a_{2k+1}}{2}$ and $\frac{a'_{2k}+a'_{2k+1}}{2}$ are a value of $C(X)$ and $C(Y)$ respectively.
Since the coefficients at W space are assumed to be zero in our algorithm, which means $(\frac{a_{2k}-a_{2k+1}}{2} - \frac{a'_{2k}-a'_{2k+1}}{2})^2 = 0$, we have:

$$\begin{aligned} D_W &= (\sum_{k=0}^{n/2}((\frac{a_{2k}+a_{2k+1}}{2} - \frac{a'_{2k}+a'_{2k+1}}{2})^2 + (\frac{a_{2k}-a_{2k+1}}{2} - \frac{a'_{2k}-a'_{2k+1}}{2})^2))^{1/2} \\ &= (\sum_{k=0}^{n/2}(2(\frac{a_{2k}-a'_{2k}}{2})^2 + 2(\frac{a_{2k+1}-a'_{2k+1}}{2})^2))^{1/2} \\ &= \frac{1}{\sqrt{2}}(\sum_{k=0}^{n/2}((a_{2k}-a'_{2k})^2 + (a_{2k+1}-a'_{2k+1})^2))^{1/2} \end{aligned}$$

Since

$$\begin{aligned} & (\sum_{k=0}^{n/2}((a_{2k}-a'_{2k})^2 + (a_{2k+1}-a'_{2k+1})^2))^{1/2} \\ &= (\sum_{k=0}^{n}(a_k - a'_k)^2)^{1/2} \\ &= D_S \end{aligned}$$

we have:

$$D_W = \frac{1}{\sqrt{2}} D_S \leq D_S.$$

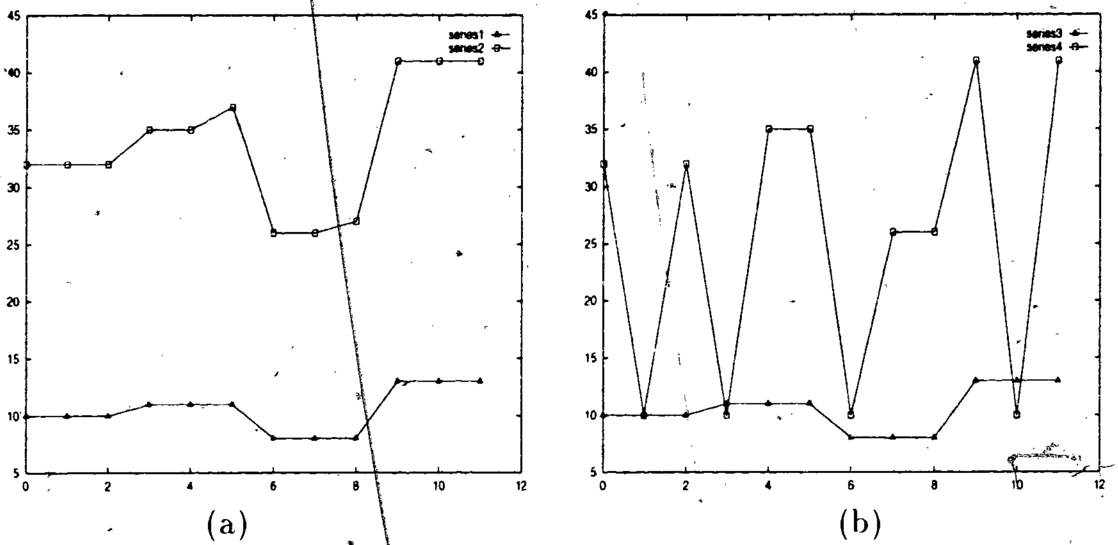


Figure 6.6: (a) Series1 and series2 before the normalization, and (b) series3 and series4 before the normalization.

It immediately follows that $D_W(C(X), C(Y)) \leq \epsilon$ whenever $D_S(X, Y) \leq \epsilon$

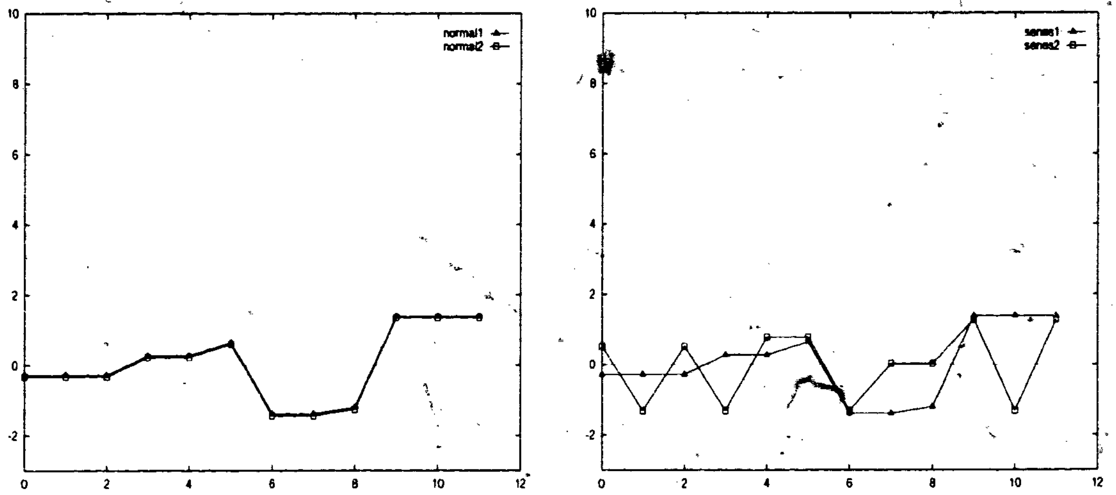
(ii) Accuracy

To establish accuracy, we want to know how likely it is that $D_S(X, Y) \leq \epsilon$ provided that $D_W(C(X), C(Y)) \leq \epsilon$. The cases when $D_W(C(X), C(Y)) \leq \epsilon$ but $D_S(X, Y) \geq \epsilon$ represent *false alarms*, and we want to minimize their occurrences. Therefore, we would like the ratio $D_W(C(X), C(Y))/D_S(X, Y)$ to be close to 1.

The actual ratio strongly depends on the nature of data and the selection of the wavelet basis. In our future work, we will do further analysis about this to make our algorithm provide good accuracy.

6.4 Experiment Results

Like the subsequence search algorithm 5.3.1, Algorithm 6.3.2 is also implemented on top of the DBMiner cube [22] developed in our lab.



(a) (b)

Figure 6.7: (a) Series1 and series2 after the normalization, and (b) series3 and series4 after the normalization.

We ran experiments on synthetic sequences for this algorithm for the purpose of easy testing.

In this section we first introduce the execution of the **whole sequence search Algorithm** 6.3.2 step by step using a synthetic data which is time-related and then we discuss the performance of the algorithm.

6.4.1 Execution Experiment

To show the validity and accuracy of Algorithm 6.3.2, let us look at the following experiment result.

Example 6.4.1 In Figure 6.6(a), series1 = (10, 10, 10, 11, 11, 11, 8, 8, 8, 13, 13, 13) and series2 = (32, 32, 32, 35, 35, 37, 26, 26, 27, 41, 41, 41), almost every value of series2 is three times and plus two of the series1, except that only two values are different. The trends of them looks quite similar in the human eye. In (b), series3 = (10, 10, 10, 11, 11, 11, 8, 8, 8, 13, 13, 13) and series4 = (32, 10, 32, 10, 35, 35, 10, 26, 26, 41, 10, 41), the two sequences are quite different, and they cannot be considered similar in the human eye.

If we use the Brute-Force Algorithm 6.3.1, the Euclidean distance of series1 and series2 in (a) is 81.49, and the Euclidean distance of series3 and series4 in (b) is 65.95. Thus in the Euclidean sense, series1 and series2 are dissimilar, although they should be considered similar in the human eye. So the Brute-Force algorithm is not applicable for this case. Thus to account for the scaling and shifting, we should normalize the sequences.

The four series are normalized in Figure 6.7, (a) is the normalization of series1 and series2, (b) is the normalization of series3 and series4. The similarity distance after the normalization are $D_S = 0.38$ and $D_S = 4.09$ for (a) and (b) respectively. It shows that the similarity distance after the normalization is decreased sharply for the similar sequences: series1 and series2 in this case, whereas, for series3 and series4 the similarity distance is still very high. If we set the threshold ϵ to 0.2, series1 and series2 will still be considered dissimilar under this threshold.

The purpose of wavelet transform procedure is to keep the main features of the sequences while smoothing them. It has shown that after the wavelet transform, the Euclidean distance of (a) in Figure 6.7 becomes much smaller which is 0.168, and the Euclidean distance of (b) in Figure 6.7 still large which is 3.188. Under the threshold ϵ of 0.2, we get the conclusion that series1 and series2 are similar, whereas series3 and series4 are dissimilar.

From here, we can see the validity of this algorithm: D_W is always less than D_S. It also shows the good accuracy of this algorithm, for the dissimilar sequences, like series3 and series4, the search answer is "dissimilar"; For the similar sequences, like series1 and series2, the search answer is "similar".

6.4.2 Scalability Study

The `whole sequence search module` is to compare a bunch of sequences with a query sequence and the sequences are with the same length.

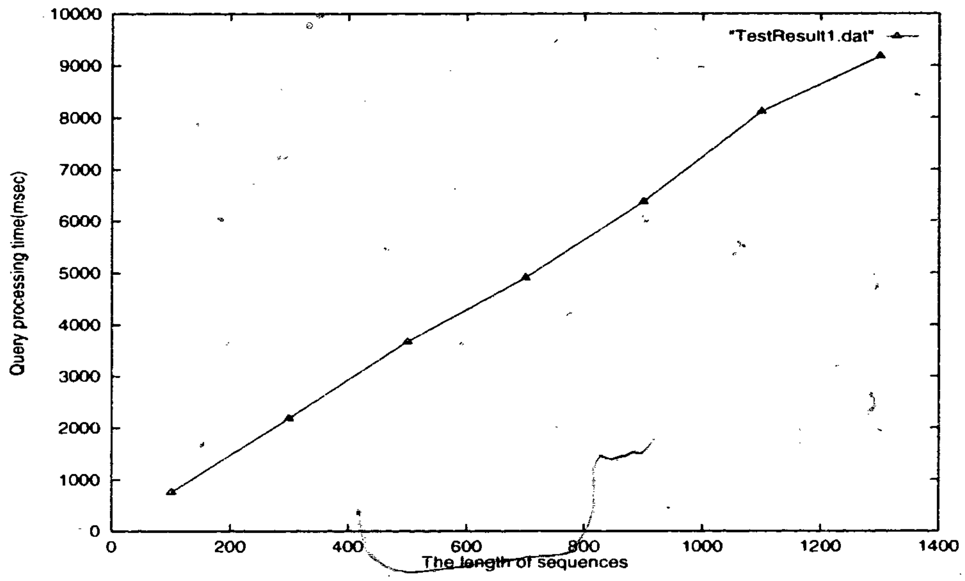


Figure 6.8: (1) The test result of whole sequence search module

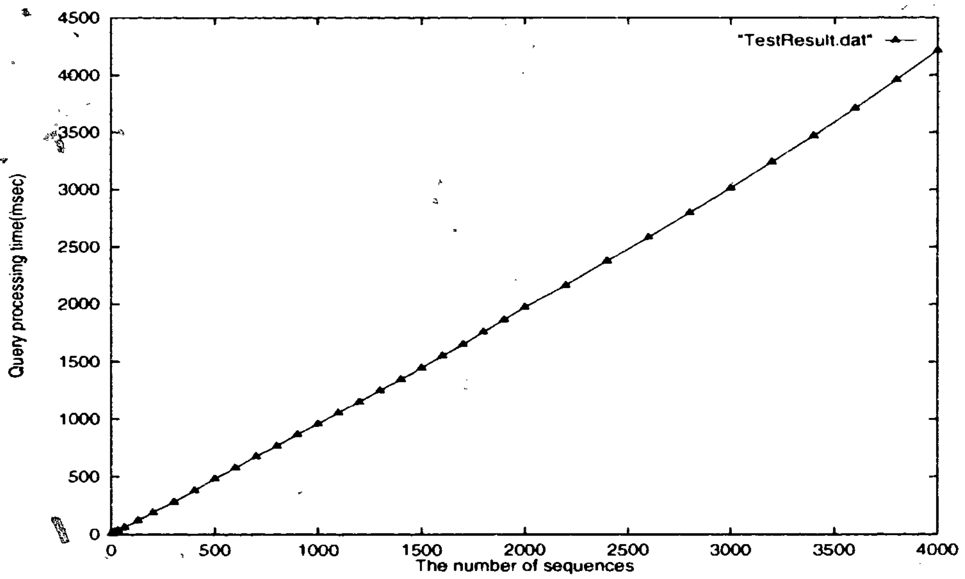


Figure 6.9: (2) The test result of whole sequence search module

Like the `subsequence search module`, the whole sequence search module is composed of three steps: normalization, wavelet transform and computation of the Euclidean distance. Thus the execution time should account for these three steps. Assume K is the number of sequences participated in the similarity search, and they have the same length N. The execution time of normalization step is $O(KN)$. At the wavelet transform step, the execution time is $O(mKN)$, where m is the filter length described in Chapter 4. At the computation of the Euclidean distance step, the execution time is $O(KN)$. Thus the total execution time of the whole sequence search module can be of order $O(KN)$.

Each synthetic sequence X is a random sequence produced randomly used the *rand()* function.

We first vary the length of the sequences from 100 to 1300 while we keep the number of sequences fixed to 1000. The performance result is shown in Figure 6.8.

In the next experiment, we kept the sequence length fixed to 128 while we vary the number of sequences from 2 to 4,000. The result we get is demonstrated in Figure 6.9.

We can see that if the sequence length is fixed, the execution time is linearly increased with the growth of the number of sequences. Similarly, when the number of sequences is fixed, the execution time is linearly increased with the growth of the sequence length. We have tested that our algorithm even can work in very large databases.

Chapter 7

Conclusions

This chapter summarizes the research and implementation work of this thesis, and discusses the major contributions of our work compared to other research work in this area.

7.1 Summary

This thesis has discussed the algorithms and implementations of similarity analysis in time-related databases. We have proposed two modules: subsequence search module and whole sequence search module. The wavelet techniques have been analyzed and used in these two modules.

Firstly, the background knowledge of wavelet analysis has been described, and the reason why we use wavelet technique instead of Fourier Transform is explained. The "compact support" feature of wavelet determines that it has better performance than Fourier analysis whenever a signal is dominated by transient behavior or discontinuities. It is also verified that wavelet technique works fine for our data in time-related databases. The implementation of wavelet transform in our system is also introduced, in which we use the fast pyramid algorithm to decompose the signal into component wavelets.

Secondly, the algorithm and implementation of the subsequence search module is introduced. The search template is a meaningful string of symbols like tokens that represent the shape information. It is implemented on top of the DBMiner cube structure in our DBMiner system. The search can be achieved in multiple abstract concept levels, and also in multiple shape resolution levels. The search query designed in this way is very intuitive for the users to understand.

Thirdly, two approaches are introduced for the whole sequence search module. The search template is a sequence of data with the same length as the source sequences. We have verified that the Brute-Force approach is so poor that it does not account for the scaling, shifting and noisy data problems. The other feasible approach has been testified that it not only accounts for the scaling, shifting and noisy data, but also proves to be valid and has good accuracy.

The experiments show that the execution time of our two modules is linearly increased by increasing both the number and the length of sequences.

7.2 Discussion

To conclude the thesis, we discuss the major contributions of our work compared to other similarity search systems, and summarize the research work that has been done so far as well as the future research areas on this method.

1. In the similarity-based queries in subsequence search area, it has the following significant features:
   - We design a set of shape tokens, which can be represented as similarity language. The users can use these tokens as a way to express their search queries. It is a simple and intuitive way for user to understand. Interesting thing is that these tokens can muffle noise data in a certain way.
   - The shape tokens are designed to represent multi-level resolutions so that the users can express their queries in multi-levels.
   - An interactively querying and answering user interface is designed and implemented.

2. Based on our knowledge, it is the first time that the wavelet technique is used in the similarity analysis of large data sets. We have given sufficient analysis of the wavelet features and the algorithms to apply these features in our similarity search modules. Its superability compared to the Fourier Transform has been verified.

3. In both of the subsequence and whole sequence search modules, the scaling and shifting problems have been accounted for.

4. Because of the adoption of the wavelet technique, both modules are robust in the presence of noise and short-term fluctuations.

7.3 Future Work

The major limitation of this method is that its application domain is limited to the one dimensional regular time-series data in which the data are in uniform time intervals, and later the method should be extended to the irregular time-series data.

For the noise term, it is difficult to give a quantified definition. It is highly depended on the features of data. We will do further empirical analysis on this in our future work.

Another problem is that we only test our method on one kind of stock data, and later we will use more stock data or data in other domains to further refine our algorithms to fit any kinds of time-series data.

Furthermore, the result of the similarity search can be further used for mining other rules, such as association, prediction, and classification. For example, according to the shape pattern result, we may find that a sharp jump of the stock market always associates with the election of a new president. This is an association rule. Other rules can also be found. This will be a good research direction, and will be used in decision support of the industry area.

Bibliography

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Proc. 4th Intl. Conf. on Foundations of Data Organization and Algorithms*, pages 69–84, October 1993.

[2] R. Agrawal, K.-I. Lin, H.S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proc. 21st Int. Conf. Very Large Data Bases*, pages 490–501, Zurich, Switzerland, Sept. 1995.

[3] R. Agrawal, G. Psaila, E. L. Wimmers, and M. Zait. Querying shapes of histories. In *Proc. 21st Int. Conf. Very Large Data Bases*, pages 502–514, Zurich, Switzerland, Sept. 1995.

[4] N. J. Ayache and O. D. Faugeras. HYPER: A new approach for the recognition and position of two-dimensional objects. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 8:44–54, 1986.

[5] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. 1990 ACM-SIGMOD Int. Conf. Management of Data*, pages 322–331, Atlantic City, NJ, June 1990.

[6] A. Bruce, D. Donoho, and H. Gao. Wavelet analysis. *IEEE Software*, pages 26–35, October 1996.

[7] R. Burden and J. Faires. *Numerical Analysis, 5 ed.* PWS Publishing Company, 1993.

[8] S. K. Chang, Y. Cheng, S. S. Iyengar, and R. L. Kashyap. A new method of image compression using irreducible covers of maximal rectangles. *IEEE Trans. on Software Engineering*, 14:651–658, May 1988.

[9] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26:65–74, 1997.

[10] C. K. Chui. *An Introduction to Wavelets.* Academic Press, Inc, 1992.

[11] I. Daubechies. Orthonormal bases of compactly supported wavelets. *Comm. Pure and Appl. Math.*, 41:909–996, 1988.

[12] I. Daubechies. *Ten Lectures on Wavelets.* The Society for Industrial and Applied Mathematics, Rutgers University and AT&T Bell Laboratories, 1992.

[13] I. Daubechies. *Different Perspectives on Wavelets.* American Mathematical Society, Providence, Rhode Island, 1993.

[14] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. 1994 ACM-SIGMOD Int. Conf. Management of Data*, pages 419–429, Minneapolis, May 1994.

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

[16] D. Q. Goldin and P. C. Kanellakis. On similarity queries for time-series databases: Constraint specification and implementation. In *1st Intl. Conf. on the Principles and Practice of Constraint Programming*, pages 137–153, LNCS 976, 1995.

[17] G. H. Gonet, M. A. Cohen, and S. A. Benner. Exhaustive matching of the entire protein sequence database. *Science*, 256:1443–1462, 1992.

[18] W. Gong. Periodic pattern search in time-related data sets. In *M.Sc. Thesis*, Simon Fraser University, November 1997.

[19] A. Graps. An introduction to wavelets. *IEEE Computational Science and Engineering*, 2:50–61, 1995.

[20] W. I. Grosky, P. Neo, and R. Mehrotra. A pictorial index mechanism for model based matching. In *Proc. Fifth IEEE Int'l Conf. on Data Engineering*, pages 180–187, Los Angeles, CA, 1989.

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

[22] J. Han. Data mining techniques. *Tutorial Notes, 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 1–71, June 1996.

[23] J. Han, J. Chiang, S. Chee, J. Chen, Q. Chen, S. Cheng, W. Gong, M. Kamber, k. Koperski, G. Liu, Y. Lu, N. Stefanovic, L. Winstone, B. Xia, O. R. Zaiane, S. Zhang, and H. Zhu. DBMiner: A system for data mining in relational databases and data warehouses. In *Proc. CASCON'97: Meeting of Minds*, pages 249–260, Toronto, Canada, November 1997.

[24] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O. R. Zaiane. DBMiner: A system for mining knowledge in large relational databases. In *Proc. 1996 Int'l Conf. on Data Mining and Knowledge Discovery (KDD'96)*, pages 250–255, Portland, Oregon, August 1996.

[25] H. V. Jagadish. A retrieval technique for similarity shapes. In *Proc. 1991 ACM-SIGMOD Int. Conf. Management of Data*, pages 208–217, Denver, Colorado, 1992.

[26] H. V. Jagadish, A. O. Mendelzon, and T. Milo. Similarity-based queries. In *Proc. 14th ACM Symp. Principles of Database Systems*, pages 36–45, San Jose, California, 1995.

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

[28] F. N. Kerlinger. *Foundations of Behavioral Research*. Holt, Rinehart and Winston, New York, third edition, 1986.

[29] W. Klösgen and J. Zytkow. Knowledge discovery in database terminology. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 573–592. AAAI/MIT Press, 1996.

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

[31] S. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 11:674–693, 1989.

[32] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23:262–272, April 1980.

[33] Modenov and Pakhomenko. *Geometric Transformations*. Academic Press, 1965.

[34] D. E. O'Leary. Knowledge discovery as a threat to database security. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 507–516. AAAI/MIT Press, 1991.

[35] D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 13–23, Tucson, Arizona, 1997.

[36] W. G. Roth. Mimsy: A system for analyzing time series data in the stock market domain. In *M.Sc. Thesis*, University of Wisconsin, Madison, 1993.

[37] P. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1:359–372, 1980.

[38] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database research: Achievements and opportunities into the 21st century. *ACM SIGMOD Record*, 25:52–63, March 1996.

[39] E. Simoudis, J. Han, and U. Fayyad (eds.). *Proc. 2nd Int. Conf. on Data Mining and Knowledge Discovery (KDD'96)*. AAAI Press, August 1996.

[40] J. L. Starck, F. Murtagh, and A. Bijaoui. Multiresolution support applied to image and filtering and restoration. *Graphical Models and Image Processing*, 57:420–431, 1995.

[41] G. A. Stephen. String searching algorithms. *Lectures Notes Series on Computing. World Scientific*, 3, 1994.

[42] E. Ukkonen. Algorithms for approximate string matching. *Information Control*, 64:100–118, 1985.

[43] E. Ukkonen. Finding approximate patterns in strings. *Journal of Algorithms*, 6:132–137, 1985.

[44] E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theoretical Computer Science*, 92:191–211, 1992.

[45] E. Ukkonen. Approximate matching over suffix trees. In *Combinatorial Pattern Matching, 4th Annual Symposium*, volume 684, pages 228–242, Springer-Verlag, June 1993.

[46] E. Ukkonen and D. Wood. Approximate string matching with suffix automata. *Journal of Algorithms*, 10:353–364, 1993.

[47] E. Veum. Compactly supported m-band wavelets. *Department of Physics and Astronomy, Missouri University*, 1996.

[48] R. Wagner and M. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21:168–173, 1975.