Fast Time-Series Searching with Scaling and Shifting

Kelvin Kam Wing Chu **Man Hon Wong** *

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong, China
Email: {kwchu, mhwong}@cse.cuhk.edu.hk
URL: http://www.cse.cuhk.edu.hk/~{kwchu, mhwong}

Abstract

Recently, it has been found that the technique of searching for similar patterns among time series data is very important in a wide range of scientific and business applications. In this paper, we first propose a definition of similarity based on scaling and shifting transformations. Sequence A is defined to be similar to sequence B if suitable scaling and shifting transformations can be found to transform A to B. Then, we present a geometrical view of the problem so that the scaling factor and the shifting offset can be determined. Moreover, sequence searching based on tree-based indexing structure can be performed. Finally, some technical aspects are discussed and some experiments are performed on real data (stock price movement) to measure the performance of our algorithm.

1 Introduction

Recently, it has been found that the technique of searching for similar patterns among time series data is very important in a wide range of scientific and business applications [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]. A time series Q is a sequence of real number $(q_1, ..., q_n)$ collected regularly in time, where each number represents a value at a point of time. For example, stock and weather data are in the form of sequences and so they are time series data.

In general, most of the current work on time series searching consider the problem below: Given a query sequence $Q = (q_1, ..., q_n)$ and a set of data sequences $S = \{S_1, ..., S_l\}$ stored in a database, they want to search for data subsequences S' that are similar to Q, where S' are subsequences of S_i $(1 \leq i \leq l)$. The distance functions of $\mathcal{L}_p$ metric are usually used to measure the dissimilarity of two sequences. Given two sequences $X = (x_1, ..., x_n)$ and $Y = (y_1, ..., y_n)$, their distance $D_p(X, Y)$ is calculated by

$$D_p(X, Y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}$$

In particular, [1, 2] consider the special case that $p = 2$, i.e., Euclidean Distance is the sole consideration for sequence similarity. However, in many applications, a weaker version of similarity which considers the effect of scaling and shifting is required. In the example below, we show how scaling and shifting affect the sequence similarity.

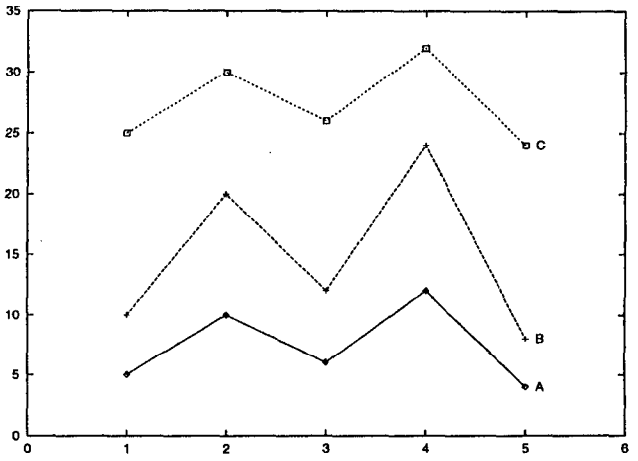

Figure 1: Examples of scaling and shifting transformation

The authors are partially supported by RGC Earmarked Grant CUHK4166/97E and CUHK Direct Grant 2050198.

PODS '99 Philadelphia PA

Example: In Figure 1, three sequences are shown. They are $A = (5, 10, 6, 12, 4)$, $B = (10, 20, 12, 24, 8)$, and $C = (25, 30, 26, 32, 24)$. Although they are different time series, they are closely related. B can be obtained from A by scaling it up 2 times, and C can be obtained from A by shifting it up 20 units. Moreover, if B is scaled down by 0.5 and then shifted up by 20 units, it becomes C. That means they are actually the same after applying suitable scaling and shifting transformations.

As mentioned in [6, 15], for stock analysis, although the stock price of company C is higher than that of company A, if they have the same fluctuation (sequences C and A in Figure 1), they should be considered to have the same trend of price. Even though the stock price of company B is always two times larger than that of company A (sequences B and A in Figure 1), if their fluctuation is proportional to their price, the trend of their price should also be regarded as the same.

In [4], the authors suggest a general idea that sequence A is similar to B if A can be transformed to B by a series of pre-defined transformations. In particular, the authors in [5] consider *moving average* and *time wrapping* as the pre-defined transformations. They define the dissimilarity between sequences A and B as the minimum possible distance after A and B are transformed by a series of pre-defined transformations. In this paper, we will also use a similar approach to define dissimilarity. However, we consider the transformations of scaling and shifting. Consider Figure 1 again, if A is the query sequence, then our algorithm will report B and C with their corresponding scaling factors and shifting offsets that transform A to B and C.

Notice that the searching algorithm should avoid brute-force checking for the scaling factors and the shifting offsets. It is because real applications usually involve a large amount of data and the brute-force checking will significantly degrade the performance. Thus, in this paper, we will present a geometrical view of the problem so that the scaling factor and the shifting offset can be determined without brute-force checking. Moreover, sequence searching based on tree-based indexing structure can be performed efficiently.

The rest of the paper is organized as follows. Section 2 provides a survey of related work. We will present our definition of sequence similarity in Section 3. In Section 4, the knowledge of vector geometry will be reviewed. Then, in Section 5, we will present the geometrical view of the problem. Based on the geometrical view, we will show our algorithm in Section 6. Finally, several concluding remarks are given in Section 7.

2 Related Work

Various methods have been proposed for time series searching. In [1], an indexing scheme called *F-index* is suggested to handle data sequences and query sequences of the same length. Firstly, each data sequence is transformed by n-point Discrete Fourier Transform. The first f_c coefficients are kept and regarded as a f_c-dimensional point. The feature points are then indexed by an R^*-tree [16]. For a range query, the query sequence is first mapped to a point in the f_c-dimensional space similarly. Then, the R^*-tree is searched and all feature points that are within the error distance from the query sequence are retrieved. This method guarantees no false dismissal, but it may cause false alarms. Thus, the original data sequences corresponding to the points retrieved have to be checked against the query sequence.

The results in [1] are further generalized in [2] and the *ST-index* is proposed to handle data sequences of different lengths. A sliding window with length n is placed over the data sequences. The subsequence within each window is transformed by n-point Discrete Fourier Transform. After all data subsequences are transformed, a trail will be formed. The trail is divided into sub-trails, which are then represented by minimum bounding rectangles(MBR) of an R^*-tree. For range query, all MBR that intersect the query region will be retrieved. This method also guarantees no false dismissal, yet false alarms are still possible, and so the original data sequences have to be checked against the query sequence, too. The methods proposed in [1, 2] are very elegant. However, they use Euclidean Distance for sequence similarity without considering any transformation. As shown by the example in Section 1, it is better to consider sequence similarity with scaling and shifting in some applications such as stock analysis.

The definition of similarity used in this paper is similar to those proposed in [4]. In [4], the authors develop a general framework for similarity queries. The framework consists of a transformation rule language T. An object A is said to be similar to an object B if A can be transformed to B by a series of transformations defined in T. Each transformation applied has a cost and the total cost is used to measure the distance between A and B.

In [5], the authors consider the case that T contains the transformation of moving average and time wrapping. They first show that the definition of sequence similarity with moving average and time wrapping has a wide range of real applications. Then, they illustrate by real stock data that the transformations help to identify similar runs of stock price. They also propose a first indexing method that can handle moving average and time warping. An index I is constructed as in [1, 2] first. For each query, a transformation in T is given. Then a new index I' is built in real time based on the given transformation T and the search is performed on the new index I'. Our definition of sequence similarity is similar to that of this paper. However, we consider the case that T contains the scaling and shifting transformations.

In [3], another definition of similarity which also considers scaling and shifting is proposed. The similarity definition of two sequences of the same length n is presented as follows. Given a tolerance ϵ, two sequences S_1, S_2 of the same length are said to be *similar* or *lie within an envelope of a specified width* if after their offsets are adjusted appropriately and their amplitudes are scaled by a suitable amount, they become S_1' and S_2' such that for all $1 \leq i \leq n, |S_1'[i] - S_2'[i]| \leq \epsilon$. The similarity definition of two sequences with different length is also given. For a maximum gap size of γ, a window size of ω and a tolerance of ξ, two sequences T_1, T_2 with different lengths are said to be similar if after some non-matching regions(length $\leq \gamma$) are ignored, the sum of the lengths of all similar pairs of subsequences(length $= \omega$) is greater than ξ times the sum of the length of T_1 and T_2. Informally, two sequences are considered to be similar if they have enough non-overlapping similar subsequences. In [3], spatial similarity join [17] is used to find all similar pairs of gap-free subsequences of the same length ω. The spatial similarity join works as follows. Each subsequence of size ω is first mapped into a point in a multi-dimensional space. The feature points are then indexed by a R^+-tree [18]. The dimensionality of this space is typically high for a reasonable window size ω. The problem of finding all similar pairs of subsequences is reduced to finding points which are within ϵ distance from each other in the index, i.e. spatial join.

In [11, 12], the transformation of shifting is considered. Each data subsequence is projected into a hyper-plane and a signature is created. Then, the searching is performed among those signatures. The projection transforms the data subsequences such that the searching with shifting can be easily performed.

In [13], a definition of sequence similarity based on the slope of sequence segments is discussed. The definition can be extended to handle sequence matching with linear scaling in both amplitude and time dimensions. Moreover, a fast sequence searching algorithm based on extendable hashing is proposed. The algorithm can match all linearly scaled sequences and guarantee that no qualified data subsequence is falsely rejected.

In [15, 19], they also argue that the definition of sequence similarity with scaling and shifting is better. They propose an efficient algorithm to determine whether two given sequences are similar or not. However, they do not propose any indexing method.

Dimension reduction techniques play a significant role in time-series indexing. Besides the method suggested in [1], recently, two novel methods have been proposed. In [20], the authors propose a novel algorithm with singular value decomposition to dynamically reduce the dimension of a data set. In [14], an efficient technique of searching for and reducing dimension of time-series data based on wavelet transform is proposed.

Other research work about time series searching include [7, 8, 9, 10, 6, 13, 14]. They focus on different definitions of similarity under different criteria. Since the space is limited, we cannot discuss them here.

3 Problem Statement

A time sequence is a sequence of real numbers which can be regarded as a multi-dimensional point

in $\Re^n$. In *Vector Analysis* [21], a point can also be represented by a position vector in $\Re^n$. Therefore, in the following discussion, we will regard time sequences, points and vectors as the same.

If a sequence $Q = (q_1, ..., q_n)$ is scaled by a real factor a, it will become $(aq_1, ..., aq_n)$. Thus, if we treat a sequence as a vector, we can regard a sequence scaling operation as a scalar-vector multiplication. That means: if a vector (sequence) $\vec{Q}$ is scaled by factor a, it becomes $a\vec{Q}$. Similarly, if a sequence Q is shifted vertically by a real offset b, it will become $(q_1 + b, ..., q_n + b)$. Thus, we can regard a vertical shift operation as a vector addition. Let the vectors (1,0,0,...,0), (0,1,0,...,0), ..., (0,0,0,...,1) in $\Re^n$ be the *standard basis* of $\Re^n$, denoted by $\vec{e_1}, ..., \vec{e_n}$. The *shifting vector of* $\Re^n$ is defined to be $\vec{N}(n) = \sum_{i=1}^{n} \vec{e_i}$. Then, a shifting operation on the vector (sequence) $\vec{Q}$ can be regarded as: $\vec{Q} + b\vec{N}(n)$, where $b \in \Re$. In the following, $\vec{N}$ will be used instead of $\vec{N}(n)$ when n is understood.

In the following, we propose a definition of similarity which considers the transformation of scaling and shifting.

Definition 1 *If sequences $\vec{u}$ and $\vec{v}$ have the same dimension n and there exists a scale-shift transformation $F_{a,b}(\vec{x}) = a\vec{x} + b\vec{N}$, where a, $b \in \Re$, such that $D_2(F(\vec{u}), \vec{v}) \leq \epsilon$, then sequence $\vec{u}$ is said to be similar to sequence $\vec{v}$ with error bound ϵ, denoted by $\vec{u} \sim_\epsilon \vec{v}$.*

Based on Definition 1, we can state the problem formally: Given a set of data sequences $S = \{S_1, ..., S_l\}$, a query sequence Q, and an error bound ϵ, we want to search for the set of data subsequences $\{S' : S'$ is a subsequence of S_i and $S_i \in S$ such that $Q \sim_\epsilon S'\}$, and find the corresponding scaling factor a and shifting offset b for each subsequence. The ranges of a and b can be regarded as the cost of the scaling and shifting transformations and the maximum cost allowed can be specified by the user as a part of the dissimilarity measures.

Notice that the retrieval scheme of this problem should satisfy the three requirements below:

1. Efficient data structure should be used to index the data sequences in order to achieve good searching performance, since the size of the time sequence database is very large in real applications.
2. The indexing structure should also be dynamic in order to cope with frequent and regular data insertion as the time series data are collected regularly.
3. The scheme should avoid brute-force checking of the scaling factors and the shifting offsets because this will lead to a long searching time. They should be determined efficiently during the search.

Well-known data structures such as R-tree [22], R^*-tree [16] and X-tree [23] are suitable for traditional spatial indexing problems because given a set of multi-dimensional data points, they can index and search for the data efficiently. However, they cannot be applied to our problem directly because they cannot determine the scaling factors and shifting offsets dynamically during the search. In the following, we will give a geometrical view of our problem and present a tree-based algorithm that satisfies all requirements above.

4 Preliminaries

Since our proposed method is based on vector geometry, we first review the properties about vector manipulations. Let vectors $\vec{u} = (u_1, u_2, ..., u_n)$ and $\vec{v} = (v_1, v_2, ..., v_n)$. The following results are well known in Vector Analysis [21, 24]:

1. The scalar product, $\vec{u} \cdot \vec{v}$, is defined to be $u_1v_1 + u_2v_2 + ... + u_nv_n$ and it is also equal to $\|\vec{u}\|\|\vec{v}\| \cos\theta$, where θ is the angle between $\vec{u}$ and $\vec{v}$.
2. The length of $\vec{u}$, $\|\vec{u}\|$, is defined to be $\sqrt{\vec{u} \cdot \vec{u}}$.
3. The projection of $\vec{u}$ along $\vec{v}$, denoted $\vec{u}_{\|\vec{v}}$, is $\frac{(\vec{u} \cdot \vec{v})}{\|\vec{v}\|^2}\vec{v}$ and the projection of $\vec{u}$ perpendicular to $\vec{v}$, denoted $\vec{u}_{\perp\vec{v}}$, is $\vec{u}_{\perp\vec{v}} = \vec{u} - \vec{u}_{\|\vec{v}}$.
4. In Vector Analysis [21], the *position vector of a point* is the vector extending from the origin to the point. For example, in $\Re^3$, $\vec{e_1} = (1, 0, 0)$, $\vec{e_2} = (0, 1, 0)$ and $\vec{e_3} = (0, 0, 1)$, the position vector of the point (x, y, z) is the vector $x\vec{e_1} + y\vec{e_2} + z\vec{e_3}$. The distance between two points pointed by $\vec{u}$ and $\vec{v}$ is equal to $\|\vec{u} - \vec{v}\|$.

5. In $\Re^n$, a line $L = \{\vec{L}(t) : \vec{L}(t) = \vec{p_0} + t\vec{d}\}$ is defined to be the set of all position vectors pointing to the points on the line, where $\vec{p_0}$ is a position vector pointing to a point on L, $\vec{d}$ is a vector parallel to L and t is a real variable. When $t = t'$, $\vec{L}(t')$ is a position vector $\vec{p_0} + t'\vec{d}$.

6. Moreover, a plane $P = \{\vec{P} : (\vec{P} - \vec{p_0}) \cdot \vec{d} = 0\}$ in $\Re^n$ is defined as the set of all position vectors pointing to the points on the plane, where $\vec{p_0}$ is a position vector pointing to a point on P and $\vec{d}$ is the *normal vector* of P, i.e. $\vec{d}$ is orthogonal to P.

In the following, we introduce two functions for the discussion in later sections. First, $PLD(\vec{p}, L)$ is defined to be the shortest D_2 distance between the point $\vec{p}$ and the line L. Secondly, $LLD(L_1, L_2)$ is defined to be the shortest D_2 distance between the two lines L_1 and L_2. The lemmas below show how $PLD()$ and $LLD()$ can be computed.

Lemma 1 *Given a point $\vec{q}$ and a line $L = \{\vec{L} : \vec{L} = \vec{p} + t\vec{d}\}$ in $\Re^n$,*

$$PLD(\vec{q}, L) = \|(\vec{q} - \vec{p}) - \frac{(\vec{q} - \vec{p}) \cdot \vec{d}}{\|\vec{d}\|^2}\vec{d}\|$$

Proof: [Omitted]

Lemma 2 *Given two lines $L_1 = \{\vec{L_1} : \vec{L_1} = \vec{p_1} + t_1\vec{d_1}\}$ and $L_2 = \{\vec{L_2} : \vec{L_2} = \vec{p_2} + t_2\vec{d_2}\}$ in $\Re^n$ and let $\vec{d_{2\perp}}$ be the projection of $\vec{d_2}$ perpendicular to $\vec{d_1}$, $LLD(L_1, L_2) = PLD(\vec{p_1}, L_2)$, if $\vec{d_1}$ is parallel to $\vec{d_2}$; Otherwise, $LLD(L_1, L_2) =$*

$$\|(\vec{p_1} - \vec{p_2}) - \frac{(\vec{p_1} - \vec{p_2}) \cdot \vec{d_1}}{\|\vec{d_1}\|^2}\vec{d_1} - \frac{(\vec{p_1} - \vec{p_2}) \cdot \vec{d_{2\perp}}}{\|\vec{d_{2\perp}}\|^2}\vec{d_{2\perp}}\|$$

Proof: [Omitted]

5 Geometrical View of the Problem

Given a vector $\vec{u} = (u_1, ..., u_n)$, after it is scaled by a factor a, it becomes $a\vec{u} = (au_1, ..., au_n)$. Geometrically, the set of all possible scalings of $\vec{u}$ can be represented by a line $Line_{sa,\vec{u}} = \{\vec{L}_{sa,\vec{u}}(a) : \vec{L}_{sa,\vec{u}}(a) = a\vec{u}, a \in \Re\}$. This line, $Line_{sa,\vec{u}}$, is called the *scaling line of $\vec{u}$*. It is the locus of $\vec{u}$ when $\vec{u}$ is scaled by the value $a \in \Re$. Similarly, given a vector $\vec{v} = (v_1, ..., v_n)$, after it is shifted vertically by an offset b, it becomes $(v_1 + b, ..., v_n + b)$. Geometrically, the set of all possible shiftings of $\vec{v}$ can be represented by a line $Line_{sh,\vec{v}} = \{\vec{L}_{sh,\vec{v}}(b) : \vec{L}_{sh,\vec{v}}(b) = \vec{v} + b\vec{N}, b \in \Re\}$. This line, $Line_{sh,\vec{v}}$, is called the *shifting line of $\vec{v}$*. It is the locus of $\vec{v}$ when $\vec{v}$ is shifted by the offset $b \in \Re$.

In other words, a scaling transformation applying on a vector $\vec{u}$ with a scaling factor a can be regarded as a movement of $a\vec{u}$ on $Line_{sa,\vec{u}}$ and the resulting vector is equal to the point $\vec{L}_{sa,\vec{u}}(a)$ on $Line_{sa,\vec{u}}$. Similarly, a shifting transformation applying on a vector $\vec{v}$ with a shifting offset b can be regarded as a movement of $(\vec{v} + b)$ on $Line_{sh,\vec{v}}$ and the resulting vector is equal to the point $\vec{L}_{sh,\vec{v}}(b)$ on $Line_{sh,\vec{v}}$.

In the following lemma, we show that the dissimilarity between the sequences $\vec{u}$ and $\vec{v}$ after a scale-shift transformation is closely related to the Euclidean distance between the point $\vec{L}_{sa,\vec{u}}(a)$ on $Line_{sa,\vec{u}}$ and the point $\vec{L}_{sh,\vec{v}}(-b)$ on $Line_{sh,\vec{v}}$.

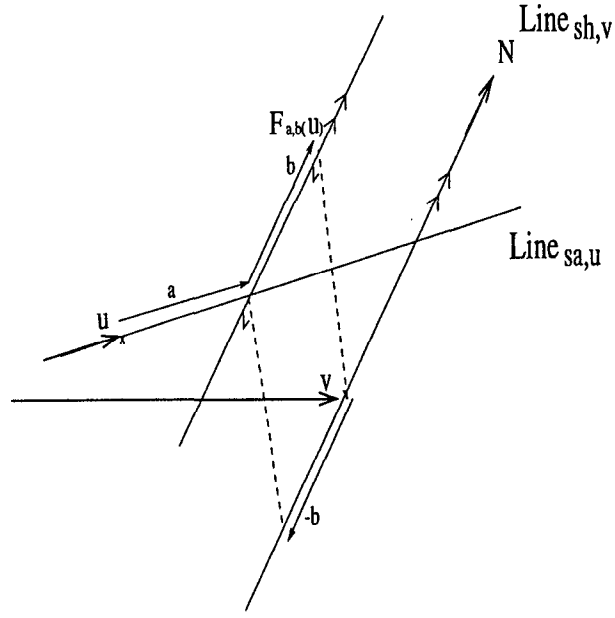

Figure 2: The geometrical meaning of Lemma 3

Lemma 3 *Given $\vec{u}$, $\vec{v} \in \Re^n$, $\|F_{a,b}(\vec{u}) - \vec{v}\| = \|\vec{L}_{sa,\vec{u}}(a) - \vec{L}_{sh,\vec{v}}(-b)\|$.*

Proof: [Omitted]

The geometrical meaning of Lemma 3 is shown in Figure 2. From Lemma 3, we notice that if there are two points on the lines $Line_{sa,\vec{u}}$ and $Line_{sh,\vec{v}}$

respectively such that their distance is less than or equal to ϵ, then there always exists a scaling factor a and shifting factor b such that the distance between the sequences $F_{a,b}(\vec{u})$ and $\vec{v}$ is less than or equal to ϵ. Thus, we have the following main theorem in this paper.

Theorem 1 *$\vec{u} \sim_\epsilon \vec{v}$ iff $LLD(Line_{sa,\vec{u}}, Line_{sh,\vec{v}}) \leq \epsilon$.*

Proof: [if part] If $\vec{u} \sim_\epsilon \vec{v}$, then by Definition 1, $\exists a, b \in \Re$ such that $\|F_{a,b}(\vec{u}) - \vec{v}\| \leq \epsilon$. By Lemma 3, $\|\vec{L}_{sa,\vec{u}}(a) - \vec{L}_{sh,\vec{v}}(-b)\| \leq \epsilon$. Since $LLD(Line_{sa,\vec{u}}, Line_{sh,\vec{v}}) \leq \|\vec{L}_{sa,\vec{u}}(a) - \vec{L}_{sh,\vec{v}}(-b)\|$ for all $a, b \in \Re$, $LLD(Line_{sa,\vec{u}}, Line_{sh,\vec{v}}) \leq \epsilon$. [only if part] If $LLD(Line_{sa,\vec{u}}, Line_{sh,\vec{v}}) \leq \epsilon$, $\exists a, b \in \Re$ such that $\|\vec{L}_{sa,\vec{u}}(a) - \vec{L}_{sh,\vec{v}}(b)\| \leq \epsilon$. By Lemma 3, $\|F_{a,-b}(\vec{u}) - \vec{v}\| \leq \epsilon$. Then, by Definition 1, $\vec{u} \sim_\epsilon \vec{v}$. ∎

According to Theorem 1, the similarity between $\vec{u}$ and $\vec{v}$ can be calculated easily by computing the shortest distance between $Line_{sa,\vec{u}}$ and $Line_{sh,\vec{v}}$. Moreover, we can have the following corollary directly from the result of Theorem 1.

Corollary 1 *If $LLD(Line_{sa,\vec{u}}, Line_{sh,\vec{v}}) = \epsilon$, then there is no $\epsilon' < \epsilon$ such that $\vec{u} \sim_{\epsilon'} \vec{v}$.*

Hence, the minimum distance between sequences $\vec{u}$ and $\vec{v}$ is $LLD(Line_{sa,\vec{u}}, Line_{sh,\vec{v}})$. Moreover, Corollary 1 implies that among a set of sequences $\vec{s_1}, ..., \vec{s_l} \in \Re^n$, the nearest neighbor of a sequence $\vec{u}$ is $\vec{s_i}$ whose shifting line has the shortest Euclidean distance to the scaling line of $\vec{u}$. Because of the limited space, we will not discuss nearest neighbor search in this paper.

5.1 Scale-Shift Transformation

By Theorem 1 and Lemma 2, we can compute the shortest distance between $Line_{sa,\vec{u}}$ and $Line_{sh,\vec{v}}$ and then determine whether $\vec{u} \sim_\epsilon \vec{v}$. Next, we want to derive an efficient indexing scheme to facilitate sequence searching with scaling and shifting. The indexing structures such as the R-tree [22], R^*-tree [16] and X-tree [23] are designed to index a set of static vectors (points) while the sequence similarity we consider is determined by the shortest distance between two lines. That means we cannot apply the above indexing structures directly. In order to solve this problem, we define a transformation which transforms every shifting line to a point on a hyper-plane.

Definition 2 *A transformation $T_{se} : \Re^n \to \Re^n$ is defined to be*

$$T_{se}(\vec{p}) = \vec{p} - \frac{\vec{p} \cdot \vec{N}}{\|\vec{N}\|^2} \vec{N}$$

where $\vec{p}$ is a position vector in $\Re^n$.

This transformation T_{se} is called *Shift-Eliminated Transformation* (or *SE-Transformation*). Geometrically, this transformation projects the point $\vec{p}$ to the plane $(\vec{P} - \vec{0}) \cdot \vec{N} = 0$ along the direction of $\vec{N}$. This plane which passes through the origin $\vec{0}$ and has its normal vector in the direction of $\vec{N}$ is called *Shift-Eliminated Plane* (or *SE-Plane*). We have four properties about the SE-Transformation and the SE-Plane:

1. The SE-Transformation T_{se} is a *linear transformation* [24] because $T_{se}(\vec{u}+\vec{v}) = T_{se}(\vec{u}) + T_{se}(\vec{v})$ and $T_{se}(t\vec{u}) = tT_{se}(\vec{u})$ for all vectors $\vec{u}, \vec{v} \in \Re^n$ and scalars $t \in \Re$.

2. A shifting line $Line_{sh,\vec{v}}$ will be transformed to a point on the SE-plane by T_{se}. Formally speaking, for all $\vec{v} + t\vec{N} \in Line_{sh,\vec{v}}, T_{se}(\vec{v} + t\vec{N}) = T_{se}(\vec{v}) + tT_{se}(\vec{N}) = T_{se}(\vec{v})$. Thus, the set $\{T_{se}(\vec{l}) : \vec{l} \in Line_{sh,\vec{v}}\}$ contains only one point $T_{se}(\vec{v})$.

3. A scaling line $Line_{sa,\vec{u}}$ will be transformed to a line lying on the SE-Plane by T_{se}. The set $\{T_{se}(\vec{l}) : \vec{l} \in Line_{sa,\vec{u}}\}$ is a line because for all $t\vec{u} \in Line_{sa,\vec{u}}, T_{se}(t\vec{u}) = tT_{se}(\vec{u})$. This line is called *the SE-line of $\vec{u}$* and denoted by $Line_{sa,T_{se}(\vec{u})}$.

4. The dimension of the SE-Plane $\{T_{se}(\vec{p}) :$ for all $\vec{p} \in \Re^n\}$ is n-1. The reason is as follows: For all $\vec{p} \in \Re^n$ and a scalar $t \in \Re$, $T_{se}(\vec{p}) \cdot t\vec{N} = \vec{p} \cdot t\vec{N} - \frac{\vec{p} \cdot \vec{N}}{\|\vec{N}\|^2}(\vec{N} \cdot t\vec{N}) = 0$. Thus, the SE-Plane is the *orthgonal complement* to the space spanned by $\vec{N}$. Then, by the theorem in [24], the dimension of the SE-Plane plus the dimension of the Subspace spanned by $\vec{N}$ is equal to n. It is obvious that the dimension of the subspace spanned by $\vec{N}$ is 1. Thus, the result follows.

The SE-Transformation can transform the scaling line $Line_{sa,\vec{u}}$ and the shifting line $Line_{sh,\vec{v}}$ to the line $tT_{se}(\vec{u})$ and the point $T_{se}(\vec{v})$ on the SE-Plane respectively. A natural question arise: Is there any relation between $tT_{se}(\vec{u})$ and $T_{se}(\vec{v})$ such that we can use it to determine whether $\vec{u} \sim_\epsilon \vec{v}$? This is exactly what we derive in Lemma 4.

Lemma 4 *Given a scaling line* $Line_{sa,\vec{u}} = \{ \vec{L}_{sa,\vec{u}}(t) : \vec{L}_{sa,\vec{u}}(t) = t\vec{u}\}$ *and a shifting line* $Line_{sh,\vec{v}}$, $PLD(\vec{L}_{sa,\vec{u}}(a), Line_{sh,\vec{v}}) = \|aT_{se}(\vec{u}) - T_{se}(\vec{v})\|$ *for* $t \in \Re$.

Proof: [Omitted]

The geometric meaning of Lemma 4 is that the distance between the point $\vec{L}_{sa,\vec{u}}(a)$ and the shifting line $Line_{sh,\vec{v}}$ in the original vector space is equal to the distance between the point $aT_{se}(\vec{u})$ (the SE-Transformation of $\vec{L}_{sa,\vec{u}}(a)$) and the point $T_{se}(\vec{v})$ (the SE-Transformation of the shifting line $Line_{sh,\vec{v}}$) in the vector space of the SE-Plane. Lemma 4 builds up a relation between the original vector space and the vector space of SE-Plane such that we can prove the following theorem.

Theorem 2 *Let* $\vec{u}$ *and* $\vec{v}$ *be two vectors in* $\Re^n$ *and* $Line_{sa,T_{se}(\vec{u})}$ *be the SE-line of* $\vec{u}$, $\vec{u} \sim_\epsilon \vec{v}$ *iff* $PLD(T_{se}(\vec{v}), Line_{sa,T_{se}(\vec{u})}) \leq \epsilon$.

Proof: Suppose $PLD(\vec{L}_{sa,\vec{u}}(t), Line_{sh,\vec{v}})$ is minimum when the scaling factor $t = a$. Observe that for such a scaling factor a, $PLD(\vec{L}_{sa,\vec{u}}(a), Line_{sh,\vec{v}}) = LLD(Line_{sa,\vec{u}}, Line_{sh,\vec{v}})$. Then, by Lemma 4, the length $\|aT_{se}(\vec{u}) - T_{se}(\vec{v})\|$ is also minimum and it is equal to $LLD(Line_{sa,\vec{u}}, Line_{sh,\vec{v}})$.

By Theorem 1, we have $\vec{u} \sim_\epsilon \vec{v}$ iff $LLD(Line_{sa,\vec{u}}, Line_{sh,\vec{v}}) \leq \epsilon$. Thus, $\vec{u} \sim_\epsilon \vec{v}$ iff $\|aT_{se}(\vec{u}) - T_{se}(\vec{v})\| \leq \epsilon$. Observe that $\|aT_{se}(\vec{u}) - T_{se}(\vec{v})\| = PLD(T_{se}(\vec{v}), Line_{sa,T_{se}(\vec{u})})$ and the result follows. ∎

Theorem 2 means that we can check the similarity of the vectors $\vec{u}$ and $\vec{v}$ in the vector space of the SE-Plane by computing the shortest distance between the SE-line $Line_{sa,T_{se}(\vec{u})}$ and the point $T_{se}(\vec{v})$. Moreover, in the proof, the scaling factor a such that $\|aT_{se}(\vec{u}) - T_{se}(\vec{v})\|$ is minimum is the scaling factor we want to determine.

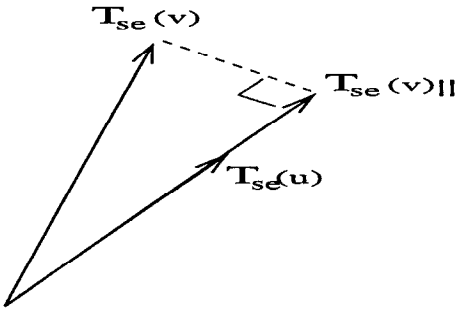

Figure 3: Determine the scaling factor.

5.2 Determine Scaling Factor and Shifting Offset

We want to determine the scaling factor a such that $\|aT_{se}(\vec{u}) - T_{se}(\vec{v})\|$ is minimum. By referring to Figure 3, we can compute it in the vector space of SE-Plane:

$$\text{scaling factor } a = \frac{\|T_{se}(\vec{v})_{\|}\|}{\|T_{se}(\vec{u})\|} = \frac{T_{se}(\vec{u}) \cdot T_{se}(\vec{v})}{\|T_{se}(\vec{u})\|^2}$$

After the scaling factor a is known, by referring to Figure 2, the shifting offset b can be computed in the original vector space of dimension n:

$$\text{shifting offset } b = \frac{(\vec{v} - a\vec{u}) \cdot \vec{N}}{\|\vec{N}\|^2}$$

In the following section, we will present a tree-based indexing and searching method in the vector space of the SE-Plane such that all qualified data subsequences can be retrieved efficiently and the corresponding scaling factors and shifting offsets can be determined by the formulae above.

6 Algorithm

By Theorem 2, for a query sequence Q and an error bound ϵ, Q is similar to a data subsequence S' if $PLD(T_{se}(S'), Line_{sa,T_{se}(Q)}) \leq \epsilon$. Therefore, our algorithm should search for all S' such that the shortest distance between $T_{se}(S')$ and $Line_{sa,T_{se}(Q)}$ is less than or equal to ϵ. Consider that the transformed data subsequences (the set of $T_{se}(S')$) are indexed by a tree-based data structure such as R-tree [22, 16]. A bounding volume will not contain potential candidates unless the bounding volume is "near" to the line $Line_{sa,T_{se}(\vec{Q})}$. Based

on this idea, our searching algorithm determines which child nodes will contain such qualified S' and then continues to traverse those child nodes only. The general idea of our algorithm can apply to any tree-based structure with any shape of bounding volume. In the following, we illustrate the idea by using R-tree which uses hyper-rectangles as its bounding volume. R-tree is chosen because it is widely used and its behavior is well understood in the database community.

R-tree is a height-balanced tree for spatial indexing. Each node (leaf or non-leaf) must contain at least m entries and at most M entries. In our implementation, each non-leaf node (or called internal node) contains entries of the form $< \text{PTR}_i, \text{MBR}_i > (m \leq i \leq M)$, where PTR_i is the pointer pointing to the ith child node and MBR_i is the minimum bounding hyper-rectangle of the ith child node. A leaf node contains entries of the form $< ID_i, S'_i > (m \leq i \leq M)$, where S'_i is the ith data subsequence stored in the leaf node and ID_i is the identity number of S'_i.

6.1 MBR Penetration

Each MBR is defined by the two endpoints, L and H, of its major diagonal, where $L = (l_1, l_2, ..., l_n)$ and $H = (h_1, h_2, ..., h_n)$ and $l_i < h_i$ for $1 \leq i \leq n$. An MBR defined by $L = (l_1, l_2, ..., l_n)$ and $H = (h_1, h_2, ..., h_n)$ is said to *contain an MBR'* defined by $L' = (l'_1, l'_2, ..., l'_n)$ and $H' = (h'_1, h'_2, ..., h'_n)$ if $l_i \leq l'_i$ and $h'_i \leq h_i$ for $1 \leq i \leq n$. In addition, an MBR defined by $L = (l_1, l_2, ..., l_n)$ and $H = (h_1, h_2, ..., h_n)$ is said to *contain a point* $P = (p_1, p_2, ..., p_n)$ if $l_i \leq p_i \leq h_i$. In an R-tree, the MBR of every non-leaf node contains all the MBR of its child nodes. The MBR of every leaf node contains all the subsequences stored in the node.

Given a node with its MBR defined by $L = (l_1, l_2, ..., l_n)$ and $H = (h_1, h_2, ..., h_n)$, we define the *ϵ-enlargement* of the MBR, denoted by *ϵ-MBR*, to be an MBR defined by $L = (l_1 - \epsilon, l_2 - \epsilon, ..., l_n - \epsilon)$ and $H = (h_1 + \epsilon, h_2 + \epsilon, ..., h_n + \epsilon)$. Moreover, an MBR is said to *be penetrated by a line* $\vec{L}(t) = \vec{p} + t\vec{d}$ if there exists t' such that $\vec{L}(t')$ is contained by the MBR. Then, we have the following theorem.

Theorem 3 *If the ϵ-MBR of an MBR is not penetrated by the SE-Line of $\vec{u}$, then there does not exist a point $T_{se}(\vec{v})$ contained in the MBR such that $\vec{u} \sim_\epsilon \vec{v}$.*

Proof: Suppose an MBR is defined by $L = (l_1, l_2, ..., l_n)$ and $H = (h_1, h_2, ..., h_n)$. If its *ϵ-MBR* is not penetrated by the SE-Line of $\vec{u}$, then there does not exist a $t \in \Re$ such that $tT_{se}(\vec{u})$ is contained in the *ϵ-MBR*. That means, there does not exist a $t \in \Re$ such that $l_i - \epsilon \leq tT_{se}(\vec{u})_i \leq h_i + \epsilon$ for all $1 \leq i \leq n$, where $tT_{se}(\vec{u})_i$ is the ith element of vector $tT_{se}(\vec{u})$. Thus, $\forall t \in \Re, \exists i \in [1, n]$ such that $tT_{se}(\vec{u})_i \geq h_i + \epsilon$ or $tT_{se}(\vec{u})_i \leq l_i - \epsilon$. However, for all $T_{se}(\vec{v})$ contained in the *ϵ-MBR*, $l_i - \epsilon \leq T_{se}(\vec{v})_i \leq h_i + \epsilon$ for all $1 \leq i \leq n$. Thus, $\forall t \in \Re, \exists i \in [1, n]$ such that $(tT_{se}(\vec{u})_i - T_{se}(\vec{v})_i)^2 \geq \epsilon^2$. Since $\|tT_{se}(\vec{u}) - T_{se}(\vec{v})\|^2 = \sum_j (tT_{se}(\vec{u})_j - T_{se}(\vec{v})_j)^2$, which is greater than or equal to $(tT_{se}(\vec{u})_i - T_{se}(\vec{v})_i)^2$. As a result, $\forall t \in \Re, \|tT_{se}(\vec{u}) - T_{se}(\vec{v})\| \geq \epsilon$. Then, by Theorem 2, $\vec{u}$ is not similar to $\vec{v}$. ∎

Thus, by Theorem 3, we only need to traverse those child nodes whose *ϵ-MBR* are penetrated by the line $Line_{sa,T_{se}(Q)}$. The whole searching algorithm is divided into three steps. They are pre-processing, searching and post-processing steps.

Pre-processing: Suppose a set of data sequences $S = \{S_1, ..., S_l\}$ with different length are given to be indexed in a database. A window of length n is placed and slid over each data sequence $S_i \in S$. Thus, a set of data subsequences of length n will be extracted from S. Then, these subsequences will be transformed by the SE-Transformation and the subsequences resulted are inserted into an R-tree.

Searching: For each query Q and an error bound ϵ, the algorithm starts from the root of the R-tree. At each level, only those child nodes whose *ϵ-MBR* are penetrated by $Line_{sa,T_{se}(Q)}$ are traversed. When a leaf-node is reached, for each transformed subsequence $T_{se}(S')$ stored in the leaf, $PLD(T_{se}(S'), Line_{sa,T_{se}(Q)})$ will be computed and by Theorem 2, the original sequences of those transformed subsequences that are within ϵ from $Line_{sa,T_{se}(Q)}$ will be retrieved.

Pro-processing: For each subsequence found in the searching step, its scaling factor and shifting offset are computed. If the user has specified the cost of the scaling and shifting transformations, check whether the cost of transforming the subsequence is less than the cost specified. If the cost

is less than the cost specified, report the subsequence to the user with the corresponding scaling factor and shifting offset.

7 Concluding Remarks

This algorithm can search for all similar data subsequences when the dimension of the query sequence is equal to the length of the extracting window, n. In [2], a general method is proposed for query sequences whose dimensions are larger than n. The query is first partitioned into several smaller sub-queries and then each sub-query is searched independently. Our algorithm can work with this method and it can be proved that no qualified similar subsequence will be missed.

As mentioned in Section 5.1, the dimension of the SE-Plane is n-1. That means, if the length of the extracting window is n, the dimension of the R-tree used is at least n-1. In [23], it is found that the searching time increases as the overlap of the R-tree increases. Moreover, the overlap increases significantly when the dimension of the R-tree is larger than 10. Thus, in our implementation, we use a technique which is also used in [1, 2, 5, 6] to reduce the dimension of the sequence data.

Our algorithm depends on the penetration checking of MBR. Two methods, called *Entering/Exiting Points* and *Bounding Spheres* respectively, can be used for this task. They are techniques of ray tracing in computer graphics [25]. Enter/Exiting Points method can determine whether a line penetrate through a rectangle. It can be easily generalized to the case of hyper-rectangle. However, in computer graphics, a heuristic method is always used to improve the performance of the penetration checking. Two bounding spheres can be introduced such that the inner sphere is tightly bounded by the ϵ-MBR and the ϵ-MBR is in turn tightly bounded by the outer sphere. Note that if the outer sphere is not penetrated by the SE-Line, the ϵ-MBR will not be penetrated by the SE-Line, too. On the contrary, if the inner sphere is penetrated by the SE-Line, the ϵ-MBR will also be penetrated by the SE-Line. The previous entering/exiting points method will be used only if the outer sphere is penetrated, but the inner sphere is not penetrated by the SE-line.

We have implemented our algorithm and the methods for penetration checking. The results show that our algorithm improves the searching performance significantly. The experiments are performed on a Sun SPARCcenter2000 workstation running Solaris 2.5.1 with 512MBytes of main memory. In the experiments, R^*-tree [16] is used to index the data subsequences and the dimension reduction technique is also used. According to the work in [2], three Fourier coefficients are sufficient to index time series data efficiently. Since each Fourier coefficient requires 2 numbers, the dimension of our R^*-tree is set to 6. The page size is 4KBytes and each page stores one internal node only. The number of maximum entries, M, of each internal node is 20. The other settings of the R^*-tree are set as suggested in [16]. For example, the minimum number of entries of each internal node, m, is set to 40% of M, i.e. 8. The re-insert parameter p is set to 30% of M, i.e. 6.

Experiments are performed on real stock data. The stock prices of one thousand companies in Hong Kong are collected from July 1995 to October 1996. Over six hundred and fifty thousand values are obtained. In each experiment, 100 queries are performed. The average CPU time used and the average number of page accesses are collected to evaluate the algorithm.

Because the space is limited, only three sets of experiments are discussed. In set 1, we study the performance of sequential search method. The time series data are read sequentially and the distance from the query sequence is computed by Lemma 2. We investigate our proposed tree-based algorithm in set 2 and set 3. In particular, set 2 uses only the Entering/Exiting Points method to check the penetration of the ϵ-MBR. Set 3 includes the heuristic inner and outer spheres.

First, we evaluate the average CPU time of the three sets by varying ϵ. The result is shown in Figure 4. It can be observed that our proposed tree-based indexing method outperforms the sequential search method. Since the sequential search method has to compare all data values for each query, it has a constant cpu time usage over the whole range of the error bound. For our proposed method, the cpu time increases as the error bound increases because more subsequences are qualified and so more branches of the R^*-tree are needed to be traversed.

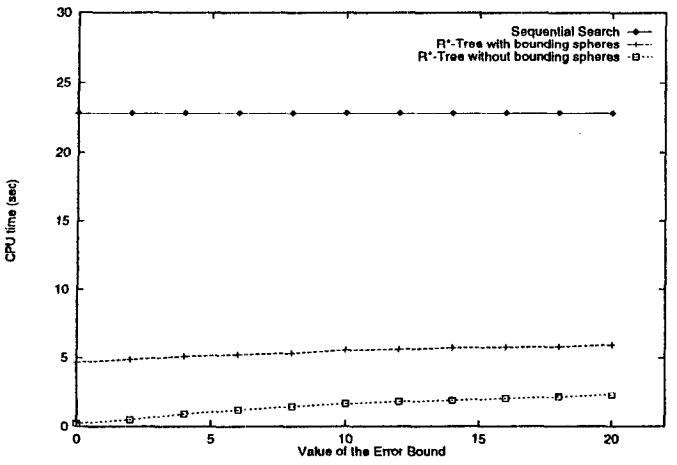

Figure 4: CPU Time vs Error Value of the 3 sets of experiments

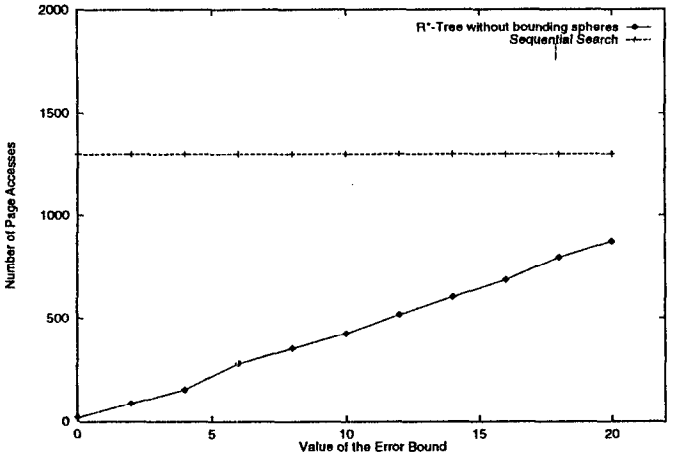

Figure 5: Number of Page Accesses vs Error Value of the 3 sets of experiments

However, what surprising here is that the performance of the searching method using bounding spheres is worse than that of the one without using it. We believe it can be explained by the work in [26]. The authors in [26] find that the bounding rectangles of R^*-tree usually have long diagonal, but small volume. That means the length of one of the dimensions of the bounding rectangles is usually much longer than the length of other dimensions. Thus, the outer sphere will be so large that the probability that a SE-Line penetrates the outer sphere but not penetrates the bounding rectangle is high. Moreover, the inner sphere will be so small that the probability that a SE-Line penetrates the bounding rectangles but not penetrates the inner sphere is also high. As a result, a lot of cpu time spent on computing the penetration of the bounding spheres cannot help to speed up the algorithm. Though the bounding spheres are good heuristic method for ray tracing in computer graphics, they cannot be applied to our problem.

In another experiment, we study the disk access behavior of the searching methods. By varying ϵ, we measure the average number of page accesses and plot it in Figure 5. For the sequential search method, the number of page accesses is constantly equal to $(0.65M \times 8Bytes)/4\text{KBytes}$ ($\approx$ 1300) pages because it has to access all pages for every query. From Figure 5, we notice that the number of page accesses of our proposed method is less than that of the sequential search method over the whole range of the error bound. When $\epsilon = 0$ (exact search), the number of page accesses of the sequential search method is one thousand times larger than that of our method.

References

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient Similarity Search in Sequence Databases. In *International Conference on Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[2] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 419–429, 1994.

[3] 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. of the 21st VLDB Conference*, pages 490–501, 1995.

[4] H. V. Jagadish, A. O. Mendelzon, and T. Milo. Similarity-Based Queries. In *Symposium on Principles of Database Systems*, pages 36–45, 1995.

[5] D. Rafiei and A. Mendelzon. Similarity-Based Queries for Time Series Data. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 13–25, 1997.

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

[7] H. Shatkay and S. B. Zdonik. Approximate Queries and Representations for Large Data Sequences. In *International Conference on Data Engineering*, pages 536–545, 1996.

[8] C.-S. Li, P. S. Yu, and V. Castelli. Similarity Search Algorithm for Databases of Long Sequences. In *International Conference on Data Engineering*, pages 546–553, 1996.

[9] T. Bozkaya, N. Yazdani, and Z. M. Ozsoyoglu. Matching and Indexing Sequences of Different Lengths. In *Proc. 1997 ACM CIKM, Sixth International Conference on Information and Knowledge Management*, 1997.

[10] B. Yi, H. V. Jagadish, and C. Faloutsos. Efficient Retrieval of Similar Time Sequences Under Time Warping. In *International Conference on Data Engineering*, 1998.

[11] S. K. Lam and M. H. Wong. A Fast Signature Algorithm for Sequence Data Searching. In *The Third International Workshop on Next Generation Information Technologies and Systems*, pages 172–181, 1997.

[12] S.K. Lam and M.H. Wong. A Fast Projection Algorithm for Sequence Data Searching. *Data and Knowledge Engineering*, 28:321–339, 1998.

[13] K.W. Chu, S.K. Lam, and M.H. Wong. An Efficient Hash-based Algorithm for Sequence Data Searching. *The Computer Journal*, 41:402–415, 1998.

[14] K. P. Chan and W. C. Fu. Efficient Time Series Matching by Wavelets . In *International Conference on Data Engineering*, 1999.

[15] B. Bollobás, G. Das, D. Gunopulos, and H. Mannila. Time-Series Similarity Problems and Well-Separated Geometric Sets. In *13th Annual ACM Symposium on Computational Geometry*, pages 454–456, 1997.

[16] 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. of the ACM SIGMOD Conference on Management of Data*, pages 322–331, 1990.

[17] K. Shim, R. Srikant, and R. Agrawal. A Fast Algorithm for high-dimensional Similarity Joins. Technical report, IBM Almaden Research Center, 1996.

[18] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+ tree: a dynamic index for multi-dimensional objects. In *Proc. of the 13th VLDB Conference*, 1987.

[19] G. Das, D. Gunopulos, and H. Mannila. Finding similar time series. In *1st European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 88–100, 1997.

[20] K. V. R. Kanth, D. Agrawal, and A. K. Singh. Dimensionality-Reduction for Similarity Searching in Dynamic Databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 166–176, 1998.

[21] H. F. Davis and A. D. Snider. *Introduction to Vector Analysis.* Wm. C. Brown Publishers, 1995.

[22] A. Guttman. R-tree: a dynamic index structure for spatial searching. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 47–57, 1984.

[23] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree : An Index Structure for High-Dimensional Data. In *Proc. of the 22th VLDB Conference*, pages 28–39, 1996.

[24] J. B. Fraleigh and R. A. Beauregard. *Linear Algebra.* Addison Wesley, 1995.

[25] A. Watt. *3D Computer Graphics.* Addison Wesley, 1993.

[26] N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 369–380, 1997.