Similarity Searching for Multi-attribute Sequences*

Tamer Kahveci Ambuj Singh Aliekber Gürel
Department of Computer Science Department of Mathematics
University of California, Santa Barbara, CA 93106
{tamer,ambuj}@cs.ucsb.edu, aliekber@math.ucsb.edu

Abstract

We investigate the problem of searching similar multi-attribute time sequences. Such sequences arise naturally in a number of medical, financial, video, weather forecast, and stock market databases where more than one attribute is of interest at a time instant. We first solve the simple case in which the distance is defined as the Euclidean distance. Later, we extend it to shift and scale invariance. We formulate a new symmetric scale and shift invariant notion of distance for such sequences. We also propose a new index structure that transforms the data sequences and clusters them according to their shiftings and scalings. This clustering improves the efficiency considerably. According to our experiments with real and synthetic datasets, the index structure's performance is 5 to 45 times better than competing techniques, the exact speedup based on other optimizations such as caching and replication.

1 Introduction

Time series or sequence data sets arise naturally in many real world applications like stock market, weather forecasts, video databases, sensor-based controls, and medicine. There is a frequent need to understand the information content of this data in order to respond better to common trends, to provide corrective emergency steps, or to predict the future evolution based on past records. Some examples of queries on such datasets include finding the companies which have similar profit/loss patterns, finding similar motion patterns in a video database, finding similar patterns in medical sensor data in order to respond to patient health problems, to predict infrastructure usage by comparison with past trends, or to predict common genetic functionality by study of gene expression patterns over time.

Time series data is said to have d attributes if d values are stored for each time point. Stock market data, which is formed by storing the closing prices of a company is an example of a 1-attribute sequence. If we include the P/E ratio and the number of shares sold, this becomes a 3-attribute sequence. The trajectory of an object moving on a plane is a 2-attribute sequence, because two values (i.e. X and Y coordinates) are stored for each discrete time point. Medical data is usually multi-attribute since a single sensor is seldom sufficient to record the health of a patient: such a sequence can be obtained by using the blood pressure values, the heart beat rate, the amount of calcium, and other parameters recorded periodically from a patient.

There are many ways to compare the similarity of two time sequences. One approach is to define the distance between two sequences to be the Euclidean distance, by viewing a sequence as a point in an appropriate multi-dimensional space [1, 4, 7, 11, 19].

Range searches and nearest neighbor searches for *whole matching* and *subsequence matching* [1] have been the principal queries of interest for time series data. *Whole matching* corresponds to the case when the query sequence and the sequences in the database have the same length. Agrawal et al. [1] developed one of the first solutions to this problem. The authors transformed the time sequence to the frequency domain by using DFT. Later, they reduced the number of dimensions to a feasible size by storing the first few frequency coefficients. Chan and Fu [4] used Haar wavelet transform to reduce the number of dimensions and compared this method to DFT. The authors found that Haar wavelet transform performs better than DFT. However, the performance of DFT could be improved using the symmetry of Fourier Transforms [17]. In this case, both methods gave similar results.

1.1 Adopting new metrics for distance

Non-Euclidean metrics have also been used to compute the similarity for time sequences. Agrawal, Lin, Sawhney, and Shim [2] use L_∞ as the distance metric. Another distance metric for multi-attribute time sequences is D_{norm} [13]. Although this metric has a high recall, it allows false dismissals.

Defining the distance as some norm of the difference between two time sequences may be insufficient if the sequences can be made closer by linear transformations. The most important transformations are scaling and shifting. Scaling is needed because of the need to compare time sequences recorded on devices with different calibrations or different units.

One emerging area of applications for shift and scale invariant comparison of time sequences is data from genome microarrays. By choosing cells from an organism under different stages of development, or under different physical conditions, valuable information can be obtained about the expression of genes, their relationship to one another, and genetic pathways. Using the absolute values of the measurements may be misleading because of variation in the physical conditions like data quality and quantity, scan-

*Work supported partially by NSF under grants EIA-0080134, EIA-9986057, IIS-9877142, ANI-0123985, and NSFIIS98-17432.

ner quality, glass quality, hybridization conditions and post-hybridization washing. In order to allow comparisons under different experimental conditions, shift and scale invariant comparisons of the resulting sequences can be useful.

Das, Gunopulos and Mannila [6] showed that the similarity between two sequences after eliminating the outliers, and scaling and shifting can be determined in O(n^6) time using *longest common subsequence* (LCSS) technique, where n is the length of the strings. However, the LCSS can be found in O($(n+m)^3\delta^3$) time [20] if only shift invariance is involved, where m and n are length of sequences and δ is the error.

Rafiei and Mendelzon [16] developed algorithms for answering similarity queries under a set of user-specified linear transformations. When these transformations are *safe*, the queries can be answered efficiently by applying the specified transformation to an index MBR. Multiple transformations can also be applied collectively to the database sequences [15]. The problem we study here is different in that we consider all possible scalings and shiftings, not just a set of user-specified transformations.

Goldin and Kanellakis [8] propose a technique based on *normalization* for the comparison of the time sequences. A *normalized* time sequence has a zero mean and a unit standard deviation. Given a query Q, the authors present a search algorithm that finds all database sequences S such that the *normalized* Euclidean distance [1] $D_N(Q, aS+b) \leq \epsilon$ for some $a > 0$ and b. Chu and Wong [5] considered the asymmetric formulation $D_2(aQ + b, S) \leq \epsilon$. They use a transformation to map the data sequences onto the *Shift Eliminated Plane*. Both of these formulations of distance are inherently asymmetric in its treatment of query and database sequences. We focus on the symmetric notion of the distance in this paper. The restriction to only positive scalings ($a > 0$) also appears artificial.

The Landmark model [14] stores the turning points of the time sequences. The distance between time sequences here is invariant with respect to 6 transformations: shifting, uniform amplitude scaling, uniform time scaling, uniform bi-scaling, time warping, and non uniform amplitude scaling. However, the landmarks of different time sequences may correspond to different time points.

1.2 Our contribution

In this paper, we consider the similarity search problem for multi-attribute sequences. We first solve the simple case in which the distance is basically defined as the Euclidean distance. Later, we extend it to handle shift and scale invariance. We point out the problems with current methods for scale and shift invariant distance computations, and propose a new symmetric notion of distance: the distance between two time sequences is defined to be the smallest Euclidean distance after scaling and shifting either one of the sequences to be as close to the other. We define two models for comparing multi-attribute time sequences: in the first model, the scalings and shiftings of the component sequences are dependent, and in the second model they are independent.

We propose a novel index structure called *CS-Index* (Cone Slice) for shift and scale invariant comparison of time sequences. As a part of this technique, the sequences in the database are first mapped to the shift eliminated plane [5]. The transformed points are then clustered in hierarchical cone slices. These slices are stored on disk according to an *in-order* traversal, and a pointer to each slice along with angle and spatial extent information is maintained in memory. Given any query, it is first mapped onto shift eliminated plane. The shift and scale invariant distance between the query and the slices are computed in memory to obtain a set of candidate slices. The hierarchical construction of the index structure allows early pruning. Finally, the candidate slices are read from disk in a single seek, and false hits are eliminated.

Experimental results show that the CS-Index structure performs 50 to 100 times faster than the R*-tree index structure and 5 to 10 times faster than sequential scan. The efficiency of the index structure can be further improved by selectively replicating or caching parts of the index structure.

The rest of the paper is organized as follows. We discuss the simple case in which the Euclidean distance is used as the dissimilarity measure in Section 2. We define the problem of shift and scale invariant search of multi-attribute time sequences in Section 3. In Section 4, we propose the *CS-index* structure for range queries and nearest neighbor queries. We present experimental results on a number of synthetic and real datasets in Section 5. We end with a brief discussion in Section 6.

2 Multi-attribute time sequences

A *d-attribute* time sequence is formed by storing d values at each time point. If v is a d-attribute time sequence of length l, then it can be represented as a vector $v = (v_1, v_2, ..., v_l)$, where $v_i = (v_{i,1}, v_{i,2}, ..., v_{i,d})$ for $1 \leq i \leq l$ ($v_{i,j}$ are scalars.). Figure 1(a) depicts a two-attribute sequence of length four.

The Euclidean distance, D_2, between d-attribute time sequences u and v of length l is defined as

$$D_2(u, v) = \sqrt{\sum_{1 \leq i \leq l} \sum_{1 \leq j \leq d} (u_{i,j} - v_{i,j})^2}.$$

2.1 Whole matching for multi-attribute sequences

If the length of the sequences in the database and the length of the query sequence is equal, similarity searching is called *whole matching*. Whole matching is a well studied problem for 1-attribute sequences (e.g. [1, 4, 7, 17]). Whole matching problem for multi-attribute sequences can be solved using any of the existing techniques after transforming multi-attribute sequences to 1-attribute sequences.

[1] This corresponds to an unbounded query in the authors' terminology.

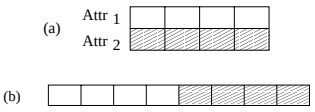


Figure 1. (a) Two-attribute sequence of length four. (b) One-attribute representation of the same sequence.

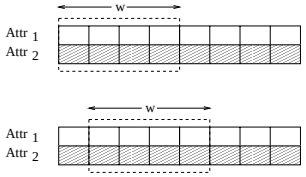


Figure 2. A sliding window of length four on a two-attribute sequence.

This transformation is performed by appending the attributes consecutively. This idea is depicted in Figure 1. Figure 1(b) shows the 1-attribute representation of the 2-attribute sequence given in Figure 1(a).

Once the multi-attribute time sequences are transformed to 1-attribute sequences, they can be viewed as points in a multi-dimensional space. For example, the sequence in Figure 1 is considered as a point in 8-dimensional space. The dimensionality of this data can be reduced using any energy preserving dimensionality reduction technique (e.g. DFT, SVD, or wavelets). These points are then indexed using any multi-dimensional index structure (e.g. R-tree [3, 9]). The distance is defined as the Euclidean distance in this multi-dimensional space.

2.2 Subsequence matching for multi-attribute sequences

Searching similar subsequences of database sequence to query sequence is called *subsequence matching*. Subsequence matching is more difficult than whole matching since the similar subsequences can be located at any location of the database sequences. Current subsequence matching techniques use sliding window based schemes to construct an index structure prior to search [7, 10, 11]. All of these techniques consider 1-attribute sequences.

Multi-attribute sequences can be handled by sliding the window over the multi-attribute sequence as in Figure 2. Each of the subsequences in these windows is then transformed into 1-attribute sequences as shown in Figure 1. Hence, each window maps to a point in a multi-dimensional space. For example, each window in Figure 2 maps to an 8-dimensional point. Later, the dimensionality of these points is reduced using a dimensionality reduction technique (e.g. DFT). The points are then indexed using the MR index structure [11].

3 Shift and scale invariant search

Simple Euclidean metric may not be sufficient for multi-attribute time sequences. In this section, we discuss the idea of shift and scale invariant distance metric. We begin with 1-attribute time sequences and later generalize to multi-attribute time sequences.

3.1 1-attribute time sequences

Consider the 1-attribute time sequences $v_1 = (2, 6, 4, 10)$ and $v_2 = (5, 7, 6, 9)$. Although the Euclidean distance between these two sequences is large, they can be made identical by scaling and shifting: $v_1 = 2 \cdot v_2 - 8 \cdot N$, where $N = (1, 1, 1, 1)$ is the normal vector. We begin by considering the existing notion of distance [5, 8].

A time sequence x can be normalized (also called *z-normalization*) as $x' = (x - mean(x))/std(x)$ [8]. That z-normalization does not minimize the Euclidean distance under all scalings and shiftings can easily be seen from the following example: Let $u = (0, 0, 1, 1)$ and $v = (3, 2, 1, 0)$, then $D_2(u', v') = 1.94$. On the other hand, $D_2(-0.5 \cdot v + 1.25, u) = 0.5$. is much less than the normalized distance.

Let v be a 1-attribute time sequence of length l, then the set of all possible scalings of v is defined as $SC(v) = \{c \cdot v | c \in \mathcal{R}\}$. The set of all scalings of v forms a line in l-dimensional space that passes through origin and v. Similarly, the set of all possible shiftings of v is defined as $SH(v) = \{v + c \cdot N | c \in \mathcal{R}\}$. The set of all shiftings forms a line in l-dimensional space which passes through v and which is parallel to the vector N. The plane which passes through the origin and is perpendicular to the normal vector N is called the *shift eliminated plane* (SE-plane). The projection of a sequence v onto the SE-plane is a point represented by $TSE(v)$. The minimum distance $d(u, v)$ between v and all scalings and shiftings of u can be computed as the distance between the projections of $SC(u)$ and $SH(v)$ onto the SE-plane. We establish the following lemma.

Lemma 1 *Given two 1-attribute time sequences u and v of the same length,*

$d(u, v) = ||TSE(v)||_2 \cdot \sqrt{1 - (\frac{TSE(v).TSE(u)}{||TSE(v)||_2 \cdot ||TSE(u)||_2})^2}$, *where $u.v$ is the dot product of the vectors u and v, and $||v||_2$ is the second norm of v.*

Though efficiently computable, the above distance formulation by Chu and Wong [5] and Goldin and Kanellakis [8] is not symmetric. That $d(u, v) \neq d(v, u)$, for some u, v, can be seen from the following example. Let $u = (0, 0, 1, 1)$ and $v = (3, 2, 1, 0)$, then
$d(u, v) = D_2(-2 \cdot u + 2.5, v) = 1$. On the other hand,
$d(v, u) = D_2(-0.5 \cdot v + 1.25, u) = 0.5$.
The scaling and shifting coefficients for this counter example are computed as in [5]. The absence of symmetry in the definition of $d(u, v)$ can lead to counter-intuitive results. For example, consider two sequences u and v from the database. It is possible that u is in the result set when

we perform a range query using v as the query sequence, but not vice-versa. It follows from Lemma 1 that the distance function $d(u, v)$ is symmetric, i.e. $d(u, v) = d(v, u)$, if and only if $||TSE(u)||_2 = ||TSE(v)||_2$. Since this condition is quite restrictive, we modify the definition of distance in order to make it symmetric.

Definition 1 *Given two 1-attribute time sequences u and v of the same length, the distance between these sequences is defined as*

$dist(u, v) = min\{d(u, v), d(v, u)\}$.

We could have used other functions such as *max*, or *average* in the above definition, but it is the *min* function that captures the notion of distance more adequately. We will see later that our index structure works as well for other functions.

3.2 Multi-attribute time sequences

The attributes of a multi-attribute time sequence can be *independent*, i.e., allowing independent scalings and shiftings, or *dependent*.

Let $N_{k,l}$ be the $k \times l$ matrix which is composed of all 1's. Define I_k to be the identity matrix of k dimensions. Let v be a k-attribute time sequence of length l.

Definition 2 *If v is a k-attribute dependent time sequence of length l, then the set of all possible scalings of v is defined as*

$SC(v) = \{c \cdot v | c \in \mathcal{R}\}$,

and the set of all possible shiftings of v is defined as

$SH(v) = \{v + c \cdot N_{k,l} | c \in \mathcal{R}\}$.

Definition 3 *If v is a k-attribute independent time sequence of length l, then the set of all possible scalings of v is defined as*

$SC(v) = \{C_k I_k v | C_k \in \mathcal{R}^k\}$,

and the set of all possible shiftings of v is defined as

$SH(v) = \{v + C_k I_k N_{k,l} | C_k \in \mathcal{R}^k\}$.

Given the above definitions of scalings and shiftings of multi-attribute time sequences, the distance function given in Definition 1 can be used for both dependent and independent multi-attribute time sequences. For example, let $u = ((0, 0), (1, 1))$ and $v = ((3, 2), (1, 0))$ be two attribute time sequences of length two. If u and v are dependent sequences, then $dist(u, v)$ = 0.25 (scale v with -0.5, and shift v by 1.25.). If u and v are independent sequences, then $dist(u, v)$ = 0 (scale v with 0, and shift the first attribute of v by 0 and the second attribute of v by 1.).

4 The CS-index structure

In this section, we propose a new index structure which clusters the data sequences according to both their scaling and shifting lines. We call this index structure *CS-index* (Cone Slice) for it resembles slices of hierarchically ordered cones. The idea is to project both the scaling lines and the shifting lines of the data sequences on to the SE-plane, and cluster the sequences whose projected shifting lines are close and for whom the angles between the projected scaling lines are small. Consideration of both the shifting line and the scaling line of data sequences is prompted by Definition 1. If we consider only the shifting lines of the data sequences, and use an R*-tree or any other similar index structure merely based on spatial closure as in [5], then there are several disadvantages: a)The angular distance between scaling lines of the data sequences clustered within a disk page may be larger than the angular distance between scaling lines of the data sequences in different pages. This may result in a large number of false hits. b) Disk I/O's for reading the candidate sequences involve random seeks, resulting in a high I/O overhead.

/*Let f be the fanout and p be the page capacity.*/
Algorithm CS-INDEX-BULKLOAD(S)
/*Let S be the set of time sequences in the database. */

1. For all $v \in S$, $v := TSE(v)$.
2. Choose a random sequence v.
3. Sort all sequences in S in ascending order of angular distance to v. Let A_S be this order.
4. Sort all sequences in S in ascending order of their distance to the origin. Let D_S be this order.
5. *SPLIT*(S, A_S, D_S).

Figure 3. CS-index bulk-loading algorithm

We will first describe the construction of the *CS-index* structure for the 1-attribute time sequences. Later we will extend the idea to the multi-attribute case.

4.1 CS-index for 1-attribute time sequences

Let q be a query sequence and let v be a time sequence in the database. Let $\theta_{q,u}$ be the angle between the vectors $TSE(q)$ and $TSE(u)$. Using Lemma 1 and Definition 1, we conclude that $dist(q, u) = min\{||TSE(q)||_2, ||TSE(u)||_2\} \cdot sin\theta_{q,u}$. This means that the distance between a query and a database sequence is based on the lengths of the projections of the two sequences on the SE-plane and the angle between the projections. Therefore, a good index structure must cluster radially, i.e., based on the distance from the origin as well as the angular distance.

The CS-index structure consists of a set of data pages stored on disk, and summary information stored in memory about the data pages. The data pages are organized in a tree structure defined by two parameters: page capacity, denoted p, and fanout, denoted f. The tree structure is virtual in the sense that it is used only for clustering of data; there are no physical index pages. The rationale for choosing the appropriate fanout will be explained later.

The data sequences are bulk-loaded into the index struc-

Algorithm SPLIT(S, A_S, D_S)

- if $|S| > p$ /* The number of points are more than page capacity */
  1. if $|S| < p \cdot f$ then $f := \frac{|S|}{p}$. /* Reduce fanout if there is not sufficient points left for current fanout */
  2. Using A_S, partition the sorted set S into subsets $S_1, S_2, ..., S_f$ of size $|S|/f$.
  3. Obtain A_{S_i} and D_{S_i}, for each i, i = 1, ..., f, by restricting A_S and D_S to S_i.
  4. for i:= 1 to f
     (a) $P := p$ sequences closest to the origin in S_i.
     (b) store P as the next page.
     (c) $S_i := S_i - P$.
     (d) update A_{S_i} and D_{S_i}.
     (e) *SPLIT*(S_i, A_{S_i}, D_{S_i}).
  5. end for
- end if

Figure 4. Split Algorithm. This algorithm recursively splits the data points first according to the angles and then according to their distances from the origin.

ture as shown by algorithms in Figures 3 and 4. First, the shifting lines of all the sequences are projected onto the SE-plane (Step 1 of procedure *CS-INDEX-BULKLOAD*). Later, these sequences are sorted based on angular distance to a randomly chosen sequence (Step 3), and based on distance to the origin (Step 4). After that, procedure *SPLIT* is invoked. This procedure carries out a clustering based on angles and distances from origin. Using the angular distances, the set of sequences is partitioned into f subsets $S_1, S_2, \ldots, S_f$ (Step 2). Later, the angular and spatial orderings for these subsets are obtained by restricting the original orderings A_S and D_S (Step 3). In the second splitting phase (Step 4), a piece of size p (Steps 4.a, 4.b, and 4.c). is chopped from each cone by intersecting it with a sphere centered at the origin. Each of these pieces is called a *slice*. Later, the angular and spatial orderings are updated for the remainder of the subset (Step 4.d). Rest of the points are then recursively split (Step 4.d). Steps 1, 2, 3, 4.a-4.d require $O(n)$ time, where n is the size of the input set S. As a result, the time complexity, $T(n)$ of the *SPLIT* algorithm satisfies the recurrence $T(n) = fT(n/f) + O(n)$. This leads to a time complexity of $O(nlogn)$ for the *SPLIT* algorithm. Since the two sorting steps in the *CS-INDEX-BULKLOAD* algorithm also have $O(nlogn)$ complexity, the entire index construction requires $O(nlogn)$ time.

Figure 5 shows different steps of the index construction algorithm on a sample dataset when f = 2. The data points are projected to the SE-plane. The dataset is partitioned into f equal sized sets based on their angular distance in Figure 5(a) (Step 2 of *SPLIT*). Later, a data page is determined by clustering p closest points in one of these sets, based on Euclidean distance to the origin in Figure 5(b) (Step 4.a-4.b of *SPLIT*). The *SPLIT* algorithm is then invoked recursively for the rest of the points in the reduced set. The reduced set is again partitioned into f sets, based on angles in Figure 5(c). Figure 5(d) presents the final index structure.

Each slice s can be viewed as the intersection of a cone with a ring that is determined by two radii, r_s and R_s, where r_s is the minimum Euclidean distance between the origin and a point on the slice and R_s is the maximum Euclidean distance between the origin and a point on the slice.

The index structure in Figure 5(d) has three levels. The root level contains two slices 4, and 11. There is a total of 14 slices, each containing at most p points. A slice s_i is said to be the *ancestor* of a slice s_j (and similarly, slice s_j is said to be a *descendant* of s_i) if the cone corresponding to s_j is contained entirely in the cone corresponding to s_i. For example, in Figure 5(d), slice 4 is the ancestor of slices 1, 2, 3, 5, 6 and 7.

The distance between a query sequence q and a slice is defined as the minimum scale and shift invariant distance between q and any point on the slice. The formal definition is as follows:

Definition 4 *Let s be a slice in the CS-index and q be a query sequence on the SE-plane. Let $\theta_{q,s}$ be the angle between q and a point in s for which $sin\theta_{q,s}$ is the minimum. The distance between q and s is defined as:*

$dist(q, s) = min\{||q||_2, r_s\} \cdot sin\theta_{q,s}$.

Using this definition, we have the following theorems:

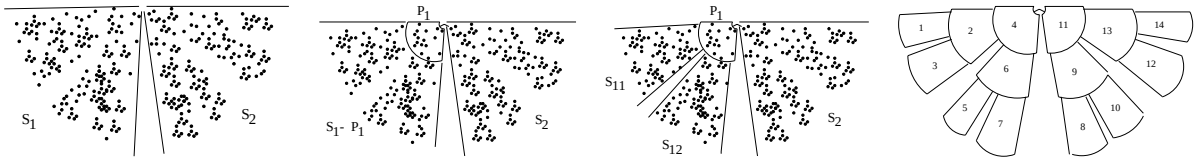
Theorem 1 *Let s be a slice in the CS-index, v be a data sequence contained in s, and q be a query sequence on the SE-plane, then*

$dist(q, s) \leq dist(q, v)$.

Theorem 2 *Let s_1 and s_2 be two slices in the CS-index such that s_1 is the parent of s_2. Let q be a query sequence on the SE-plane, then*

$dist(q, s_1) \leq dist(q, s_2)$.

Some important observations about the CS-index are as follows. 1) If the distance between a slice and a query sequence is greater than ϵ (Theorem 1), then the distance between the query sequence and any data sequence contained in that slice is greater than ϵ. In other words, given a range query, a slice may contain candidate sequences only if the distance between the query sequence and that slice is less than the search range. 2) If the distance between a slice and a query sequence is greater than ϵ, then the distance between that query sequence and all children of that slice is greater than ϵ (Theorem 2). 3) If the distance between a slice and a query sequence is less than ϵ, then the distance between that query sequence and all the ancestors of that slice is less than ϵ (Theorem 2).



(a) (b) (c) (d)

Figure 5. Construction of the CS-index when f = 2. The data points are transformed onto shift eliminated plane. (a) Angular partitioning, (b) spatial partitioning, (c) recursive invocation of angular partitioning, (d) final index structure.

Algorithm RANGE-QUERY(q, ϵ)

1. $C := \emptyset$ /* the set of candidate pages*/
2. $S := \emptyset$ /* stack for navigation*/
3. Push all the root slices s_1, s_2, ..., s_f on to S.
4. While $S \neq \emptyset$
   (a) $s := Pop(S)$
   (b) if $dist(q, s) \leq \epsilon$
      i. $C := C \cup \{s\}$
      ii. Push all the children of s on to S
5. Read the candidate pages in set C and perform postprocessing to eliminate false retrievals.

Figure 6. Range search algorithm

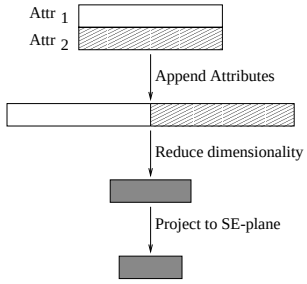


Figure 7. Transformation of a 2-attribute dependent time sequence to the SE-plane.

4.2 Similarity search on the CS-index structure

The range query algorithm on the CS-index is presented in Figure 6. A range query is performed in two phases: an in-memory candidate generating phase followed by a disk-based post-processing step. In the in-memory phase, the search starts from the root pages and proceeds downwards. If the distance between the query sequence and a slice is less than the query range (Step 4.b), then the method marks this slice as a candidate and expands the query to all the children of that slice. If the distance between the query sequence and a slice is greater than the search range, then the algorithm prunes that slice and all its children. Once the candidate sets are determined, the disk-based processing step begins. Using the disk placement information, a sequential scan is used to read all the candidate pages. Data points that are not in the range are pruned to ensure no false retrievals. Since the set of pages read from disk during the post-processing step is a subset of all the pages, our method performs no worse than sequential scan. In fact, it performs much better since clustering reduces the range of the sequential scan considerably. As a consequence of Theorem 1 and Theorem 2, we have the following corollary:

Corollary 1 *The range query algorithm in Figure 6 does not incur any false drops.*

We present a detailed discussion of the nearest neighbor queries on the CS-index structure in the technical report version of this paper [12].

4.3 Multi-attribute CS-index

The CS-index structure, so far defined for 1-attribute sequences, can be extended easily to multi-attribute sequences. We consider the multi-attribute extension for dependent and independent time sequences separately.

4.3.1 Case 1: dependent attributes

If the multi-attribute time sequences in the database are k-attribute dependent time sequences of length l, then the problem reduces to the 1-attribute case by simply transforming the sequences into 1-attribute time sequences of length $k \cdot l$ as in Figure 1. This is justified because all the entries are scaled and shifted by the same amount. Figure 7 depicts how the dependent attributes are handled. These sequences are considered as points in a $(k \cdot l)$-dimensional space. The dimensionality of these points are then reduced using a dimensionality reduction technique (e.g. DFT). Later, the CS-index structure is constructed on these points as explained in Figures 3, 4, and 5. Range queries are performed as in Figure 6.

4.3.2 Case 2: independent attributes

If the database consists of k-attribute independent time sequences of length l, then all the attributes must be consid-

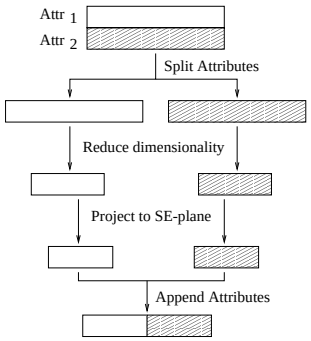


Figure 8. Transformation of a 2-attribute independent time sequence to the SE-plane.

ered separately. This is because different attributes may be scaled or shifted by different amounts. Figure 8 presents how the CS-index structure handles independent attribute sequences. We first split each time sequence into k 1-attribute time sequences of length l. This is like splitting a $k \cdot l$-dimensional space into k non-overlapping l-dimensional spaces. As a result, each k-attribute time sequence corresponds to k points in l-dimensional subspaces. We determine the SE-planes of these l-dimensional subspaces. We project the 1-attribute time sequences onto their corresponding SE-planes, and concatenate the vectors corresponding to these k projections to construct a 1-attribute $k \cdot (l-1)$-dimensional point. We construct the CS-index on these points as described in Section 4.1. Every slice of the constructed index can be projected into k different subspaces; these projections are called *subslices*. Another choice would be to maintain k separate CS-index structures for 1-attribute $(l-1)$-dimensional points. However, this would require additional post-processing.

For a given range query or a nearest neighbor query of k attributes, we split the query into k 1-attribute subqueries, one for each attribute. For each subquery, we obtain its distance to a slice by considering the subslice corresponding to that attribute. The distance between a query and a slice is defined to be the square root of the sum of squares of the k different subquery subslice distances. Once these distances are obtained, pruning and post-processing proceeds as in the single-attribute case. Range queries are performed similar to Figure 6. The only difference is that the distance function is computed for each attribute separately, and the results are accumulated to find the distance for the independent case.

4.4 Improving post-processing performance

The candidate slices for a range query or a NN-query are determined using an in-memory search. The postprocessing step uses one sequential scan to read the candidate slices. The performance of the index structure is therefore determined by how closely clustered the candidate slices are on disk [2]. The number of non-candidate slices placed between the first and the last candidate slice on disk should be minimized. A clustering of candidate slices can be achieved by three techniques: carrying out a more effective pruning in the in-memory phase, optimizing the placement of pages on disk, and caching/replication of disk pages. We elaborate on these ideas next.

4.4.1 Fanout selection

We noted earlier that the fanout f is an independent parameter of the bulk-loading algorithm that is not affected by the size of disk pages. For a given dataset, a large fanout leads to thick slices that span a smaller angle, whereas a small fanout leads to thin slices that span a larger angle. The success of the pruning procedure depends on both the thickness and the angular span of the slices: a thick or a wide slice is less likely to be pruned. The right choice of fanout ensures that slices are not too thick and not too wide. This can be determined either experimentally or theoretically if the data distribution is known. In our experiments, the optimal value for fanout varied between 5 and 7.

4.4.2 Disk placement

The second parameter that improves the post-processing performance is the placement of pages on disk. Note that if the distance between a query sequence q and a slice s is less than the given search range ϵ, then the distance between q and the parent of s is also less than ϵ. Therefore, if a slice s is in the candidate set, then all its ancestors are also in the candidate set. For example, if slice 3 of the CS-index in Figure 5 is a candidate, then slices 2 and 4 are also guaranteed to be in the candidate set. In order to reduce I/O cost, slices 1, 2, and 4 should be stored contiguously on disk. In general, the slices should be placed on disk in a manner that minimizes the distance between a slice and all its descendants. This means that the slices belonging to a subtree should be stored contiguously; it does not help to interleave a subtree with slices from a sibling subtree. The second condition that minimizes the parent-child distance is that a root node should be linearized in the middle of its subtree. These two conditions imply an *in-order* placement: a tree with $2k$ subtrees is linearized by an *in-order* traversal of its first k subtrees, placement of the root slice, and an *in-order* traversal of the remaining k subtrees. This linearization is used to place the slices on the disk.

4.4.3 Replication and caching of pages

The final parameter that improves the post-processing performance is the degree of caching and replication of disk pages. Both caching and replication can reduce the number of redundant pages that are read.

Replicating a page means that we maintain a copy of the page with all its subtrees on disk. Replicating k levels of the index structure means that we replicate the pages at the first k levels of the CS-index at their children. The advantage of

[2] For simplicity we assume a 1-d disk model.

Dataset	Size	R-tree Size	CS-index Size
stock market dataset	2.5M	160K	25K
motion dataset	8M	393K	41K

Table 1. The size of the R-tree and the CS-index structure for stock market dataset and motion dataset.

replication is that it can reduce the distance between a page and its ancestors. Replication works best if the queries are sufficiently narrow so that all candidate pages belong to a subtree and its ancestors. Otherwise, it can lead to redundant pages being read.

Caching k levels of the index structure means to keep the pages at the first k levels of the CS-index in memory and to place the in-order linearizations of the subtrees at level k on disk. Unlike replication, caching can *only* improve the performance of our index structure.

4.5 Extending the CS-index structure to other symmetric distance functions

The CS-index structure can also be used when the distance function is obtained by using *max* or *avg* functions instead of *min* function as follows:

Case 1. $dist(q,u) = max\{d(q,u), d(u,q)\}$.
In this case, one can prove that

$dist(q,u) = max\{||TSE(q)||_2, ||TSE(u)||_2\} \cdot sin\theta_{q,u}$.
Similar to the *min* function, this distance function also uses a Euclidean distance and an angular distance in its computation. The CS-index structure will work well for the *max* function since it clusters time sequences based on the distance from the origin as well as the angular distance.

Case 2. $dist(q,u) = avg\{d(q,u), d(u,q)\}$.

$dist(q,u) = \frac{(min\{d(q,u),d(u,q)\}+max\{d(q,u),d(u,q)\})}{2}$.
Hence, $dist(q,u) = \frac{||TSE(q)||_2+||TSE(u)||_2}{2} \cdot sin\theta_{q,u}$. Similar to *min* and *max* functions, *avg* function is also based on the distance from the origin and the angular distance. As a result of this, the CS-index structure will work well for the *avg* function too.

5 Experimental results

We carried out several experiments to test the performance of the CS-index structure. We used three different datasets in our experiments:

1) The first dataset is a stock market dataset, obtained again from *chart.yahoo.com*. The time sequences in this dataset consist of 2 attributes. The first attribute is the closing price, and the second attribute is the volume. There are 20,000 time sequences of length 32 in this dataset.

2) The second dataset is obtained synthetically by considering four different kinds of object trails in a 2-attribute sequence. The motions that we consider are: bouncing ball, circular motion, billiard ball moving within the confines of a rectangular table with perfect carom and elasticity, and a periodic motion along a sine curve. This dataset contains 2^{15} time sequences, distributed evenly among the four different motion types. The length of the time sequences is 32.

3) The third dataset is a multi-attribute dataset of 1, 2, 4, and 8 attributes. The time sequences are obtained synthetically by adding four sine signals of random frequencies and amplitudes and some amount of random noise. Corresponding to each choice of attributes, we have 2^{15} time sequences of length 32.

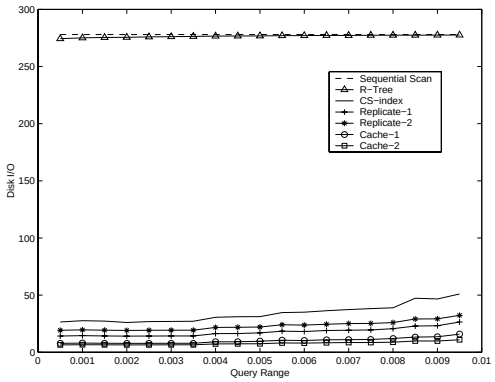


Figure 9. I/O overhead of CS-index versus R-Tree and sequential scan for the 2-attribute dependent stock market dataset.

We compressed the time sequences in these datasets to 4 dimensions using DFT, and then built the CS-index structure on the compressed data. Since both DFT and TSE are distance preserving transformations, there are no false drops in the resulting index structure. We also built R-Tree index as proposed by Chu and Wong [5] for comparison. Since the CS-index structure uses bulk loading, we used the VAM-Split implementation of R-Trees [21]. Table 1 displays the size of the CS-index structure and R-tree for the first two datasets. The size of the CS-index structure is much smaller than R-tree. This is because the CS-index structure disk does not store pointers to individual time sequences in the database since slices of the CS-index structure corresponds to continuous pages on the disk. In our query model, we considered range queries with 20 different values of ϵ in the range $(0.001, 0.01)$. For every value of ϵ, 1000 sequences in the database were chosen at random for querying. We assume that the page size is 4K in our experiments.

The first experiment considers the dependent attributes. 2-attribute stock market dataset. Figure 9 shows the I/O overhead for sequential scan, R-tree, and the CS-index structure. For the CS-index structure, we also present results for replicating one level, replicating two levels, caching one level, and caching two levels. The results show that the R-Tree index structure accesses almost all the pages. This can be explained as follows. The angular distance between two points may be large even if the Euclidean distance between them is small. Since R-Tree index

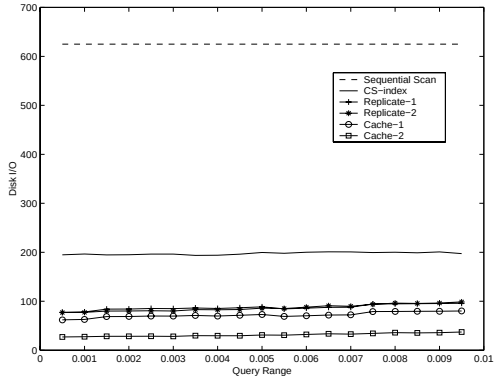


Figure 10. I/O overhead of CS-index versus sequential scan for the 2-attribute stock market dataset.

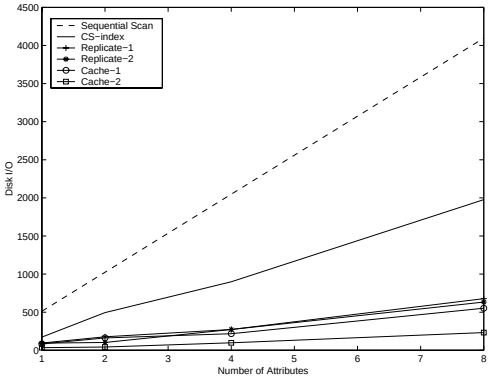


Figure 12. I/O overhead of CS-index versus sequential scan for the sine curve dataset of an increasing number of attributes.

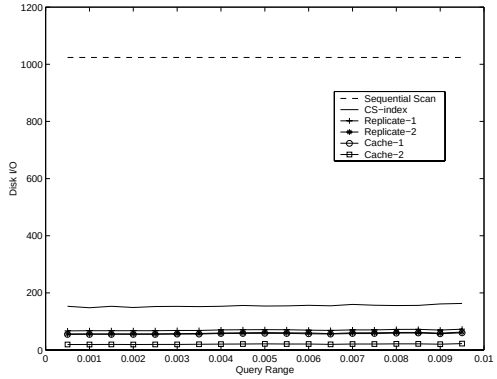


Figure 11. I/O overhead of CS-index versus sequential scan for the 2-attribute motion dataset.

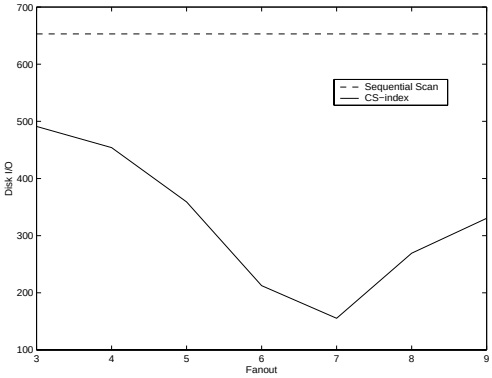


Figure 13. I/O overhead of CS-index versus sequential scan for the 2-attribute sine curve dataset for different fanout values.

structure clusters based on Euclidean distance, the resulting boxes may span a large angular distance. This may lead to a large number of candidate MBRs. Though R-tree index and sequential scan access a similar number of pages, R-tree index accesses pages randomly. Since a random read can cost approximately 10 times a sequential read [18], the performance of the R-tree index is about 10 times worse than sequential scan. Therefore, we conclude that the R-Tree index structure is not appropriate for scale and shift invariant searches. On the other hand, the CS-index structure accesses less than 20% of all the pages. Since, the CS-index structure performs a single sequential disk read, it incurs a seek cost only once (like sequential scan). CS-index is 5 to 10 times faster than sequential scan. Replicating only one one level of the CS-index structure almost doubles the speedup. Replicating two or more levels degrades the performance. If only one level of the CS-index is cached, then the speedup is 15 to 35.

In remaining experiments, we assume that the time sequences in the dataset have independent attributes. We compare the performance of our method to sequential scan. R-Tree results are not presented, because its performance was much worse than sequential scan. Figures 10 and 11 show the experimental results for the stock market dataset and the motion dataset compressed to 4 dimensions. We obtained speedups up to 8 for CS-index with no replication and caching. The speedup doubled when one level is replicated. The speedup increased to 40 when 2 levels of our index structure are cached.

The next experiment reports the scalability of the CS-index structure for an increasing number of attributes. In this experiment, we used the third dataset. Figure 12 plots the number of disk reads for sequential scan and CS-index structure. As the number of attributes increases, the size of the dataset increases linearly. This is reflected in the number of disk reads for sequential scan. The experimental results show that the number of disk reads for the CS-index structure also increases linearly with the number of attributes. Therefore, the speedup of the CS-index structure remains invariant under an increasing number of attributes.

Figure 13 plots the number of disk read for the CS-index structure for different values of fanout for the stock market data set compressed to 4 dimensions. The CS-index performs the best when the fanout is 7 for this data set. We obtained a similar *U-shaped* graph for the other datasets too. The optimal fanout for other datasets varies between 5 and 7. Reason for this is explained in Section 4.4.1.

More comprehensive experimental results for additional datasets and a discussion for subsequence searches are available in the technical report version of this paper [12].

6 Discussion

We considered the problem of similarity search for multi-attribute sequences. First, we considered the simple Euclidean distance metric. Later, we considered more challenging metrics of shift and scale invariance. We formulated a new notion of similarity that is symmetric in allowing transformations on both query and data sequences. Furthermore, we do not impose any restriction on the shifting and scaling constants. We considered both the cases of when the scalings and the shiftings of the attributes are dependent and when they are independent.

We proposed a new index structure called CS-index that clusters time sequences according to their scalings and shiftings. This index structure recursively splits the search space into hierarchical cones and selects a slice of each cone as a disk page. This index structure allows early pruning in the search phase. Later, we considered techniques to improve the performance of the index structure: in-order based placement on disk, choice of right fanout, and caching/replication. Finally, we showed that the method can be extended to multi-attribute time sequences.

According to experimental results with both real and synthetic datasets, our method performs 5 to 10 times faster than sequential scan. We also evaluated the effects of replicating higher levels of the index structure. Replicating only the root level of the CS-index almost doubled the performance of our method. Further replication eventually degraded the performance. We also experimented with caching. According to our experiments, if only the pages at the root level are cached, our method performs 10 to 25 times faster than sequential scan. We obtained speedup up to 45 when we cached one more level.

The techniques presented in this paper can be easily extended to perform shift and scale invariant subsequence searches. According to our experiments, our technique is 3 times faster than sequential scan for subsequence searches.

Multi-attribute time sequences are an important emerging class of applications. They arise ubiquitously and naturally: in medical applications, in control applications, in video and event sequences, and in history-based applications such as the stock market. The ability to query such data under different distance metrics is necessary for understanding and analyzing the characteristics of such datasets. The index structures presented in this paper are an important first step toward this, and should be widely applicable.

References

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *FODO*, Evanston, Illinois, October 1993.

[2] R. Agrawal, K. Lin, H.S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *VLDB*, Zürich, Switzerland, September 1995.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, Atlantic City, NJ, 1990.

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

[5] K.K.W. Chu and M.H. Wong. Fast time-series searching with scaling and shifting. In *PODS*, Philadelphia, PA, 1999.

[6] G. Das, D. Gunopulos, and H. Mannila. Finding similar time series. In *PKDD*, pages 88–100, 1997.

[7] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, Minneapolis MN, May 1994.

[8] D. Q. Goldin and P. C. Kanellakis. On similarity queries for time-series data: Constraint specification and implementation. In *CP*, pages 137–153, France, September 1995.

[9] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[10] T. Kahveci and A. Singh. An efficient index structure for string databases. In *VLDB*, pages 351–360, Roma, Italy, September 2001.

[11] T. Kahveci and A. Singh. Variable length queries for time series data. In *ICDE*, Heidelberg, Germany, 2001.

[12] T. Kahveci, A.K. Singh, and A. Gürel. Shift and scale invariant search of multi-attribute time sequences. Technical report, UCSB, 2001.

[13] S.-L. Lee, S.-J. Chun, D.-H. Kim, J.-H. Lee, and C.-W. Chung. Similarity search for multidimensional data sequences. In *ICDE*, San Diego, CA, 2000.

[14] C.-S. Perng, H. Wang, S.R. Zhang, and D.S. Parker. Landmarks: a new model for similarity-based pattern querying in time series databases. In *ICDE*, San Diego, USA, 2000.

[15] D. Rafiei. On similarity-based queries for time series data. In *ICDE*, Sydney, Australia, March 1999.

[16] D. Rafiei and A.O. Mendelzon. Similarity-based queries for time series data. In *SIGMOD*, pages 13–25, Tucson, AZ, 1997.

[17] D. Rafiei and A.O. Mendelzon. Efficient retrieval of similar time sequences using DFT. In *FODO*, Kobe, Japan, 1998.

[18] B. Seeger, P.-A. Larson, and R. McFayden. Reading a set of disk pages. In *VLDB*, pages 592–603, 1993.

[19] C. Shahabi, X. Tian, and W. Zhao. TSA-tree: A wavelet-based approach to improve the efficieny of multi-level surprise and trend queries. In *SSDBM*, 2000.

[20] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, San Jose, CA, 2002.

[21] D. White and R. Jain. Similarity indexing: Algorithms and performance. In *SPIE Storage and Retrieval for Image and Video Databases*, 1996.