Fast and Flexible Multivariate Time Series Subsequence Search

Kanishka Bhaduri
MCT Inc. at NASA Ames
Kanishka.Bhaduri-1@nasa.gov

Qiang Zhu*
CSE Dept, UCR
qzhu@cs.ucr.edu

Nikunj C. Oza
NASA Ames, IDU
Nikunj.C.Oza@nasa.gov

Ashok N. Srivastava
NASA Ames, IDU
Ashok.N.Srivastava@nasa.gov

ABSTRACT

Multivariate Time-Series (MTS) are ubiquitous, and are generated in areas as disparate as sensor recordings in aerospace systems, music and video streams, medical monitoring, and financial systems. Domain experts are often interested in searching for *interesting* multivariate patterns from these MTS databases which often contain several gigabytes of data. Surprisingly, research on MTS search is very limited. Most of the existing work only supports queries with the same length of data, or queries on a fixed set of variables. In this paper, we propose an efficient and flexible subsequence search framework for massive MTS databases, that, for the first time, enables querying on any subset of variables with arbitrary time delays between them. We propose two algorithms to solve this problem — (1) a *L*ist *B*ased *S*earch (*LBS*) algorithm which uses sorted lists for indexing, and (2) a *R**-tree *B*ased *S*earch (*RBS*) which uses Minimum Bounding Rectangles (MBR) to organize the subsequences. Both algorithms guarantee that all matching patterns within the specified thresholds will be returned (no false dismissals). The very few false alarms can be removed by a post-processing step. Since our framework is also capable of Univariate Time-Series (UTS) subsequence search, we first demonstrate the efficiency of our algorithms on several UTS datasets previously used in the literature. We follow this up with experiments using two large MTS databases from the aviation domain, each containing several millions of observations. Both these tests show that our algorithms have very high prune rates (>99%) thus needing actual disk access for only less than 1% of the observations. To the best of our knowledge, MTS subsequence search has never been attempted on datasets of the size we have used in this paper.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications—*Data Mining*

General Terms

Algorithms, Experimentation, Performance

Keywords

Similarity Search, Multivariate Analysis, Large Scale Mining

1. INTRODUCTION

Many data mining application domains generate large multivariate time series (MTS) databases. Examples of such domains include Earth sciences, music, video, medical monitoring, and aeronautical and aerospace systems, and financial systems. Domain experts are often interested in searching for particular patterns—waveforms over subsets of variables which may occur within some window of time of each other.

The motivation for this research comes from applications in Aviation Safety. Consider a typical problem that a safety analyst at an airline might want to address. Suppose that the airline has a large database of one million flights of multivariate time series that show the settings of the control surfaces (usually discrete signals), the pilot inputs (discrete), as well as the heading, speed, and readings from the propulsion systems (all usually continuous). In many such databases, the number of recorded parameters from a modern aircraft is nearly 1000. The safety analyst may want to find all situations in the database that correspond to a "go-around" which means that a landing has been aborted and the aircraft is directed to circle back for another landing.

Such a situation would correspond to a query on a subset of the fields in the time series database where the event **LANDING GEAR RETRACTED** occurs just after **ALTITUDE** descends below 2000 feet. This event is typical of what happens when a landing attempt must be aborted and the plane has to circle back to an appropriate point and attempt to land again. Another search for indicators of an "unstable approach" may include searching on parameters including speed, descent rate, vertical flight path, and several cockpit configuration parameters. Again, this search would be done on about a dozen parameters out of the 1000 parameters that may be recorded on the aircraft. The events

[*] This work was done when the author was vising NASA Ames.

KDD '10 Washington DC, USA

would be separated in time and may or may not occur on a particular flight.

Figure 1 shows an MTS from a real aviation dataset of CarrierX [1]. Each MTS contains the data collected from multiple sensors of an aircraft during a flight. In the figure, the x-axis refers to the different parameters while the y-axis refers to time of sampling the values. Typically, an analyst may be interested in only searching a subset of all the variables available. Queries by the analyst may look like:

1. Return all the flights (a subset of the MTS) where the altitude monotonically changes from 10000 ft to 5000 ft, speed varies between 300 knots to 200 knots, and landing gear is down. Such combination of parameter values may be precursors to unstable approaches while landing.

2. Return all the flights where the aircraft is climbing at 100 ft/s with flaps not withdrawn. There may be a time delay between these two sequences.

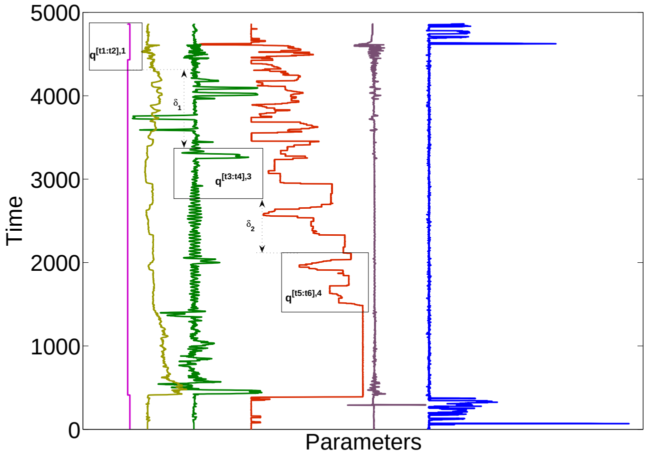


Figure 1: Sample MTS dataset and query Q. x-axis refers to different parameters and y-axis refers to time. Components of query and time delays are also shown.

Current research in MTS search [16][19][15][7] does not support the types of queries described here. Current algorithms in this area require that the query be of the same length as all the MTS in the database and that all queries be on a fixed set of variables (usually all the variables). Additionally, current algorithms do not allow for any time lag between the variables in the query.

Our primary application of interest is in the area of aviation. Given a large database of flight recorded data we wish to provide a search technology that allows analysts to rapidly identify flights with particular characteristics (as defined across a set of events on a subset of the multivariate time series). Thus, user supplies a query consisting of waveforms over several variables — typically substantially fewer than the total number of variables present in the database. The user may choose how many and which variables to query over every time, *i.e.*, this need not be fixed in advance. Also the query may cover any desired length of time up to the maximum length of the available time series. The waveforms have some (possibly zero) time shifts between them. The user also supplies a threshold for each variable describing the maximum allowable difference between the query variable and the corresponding variable in any matches that are returned. This threshold is in the same units as the corresponding variable to make threshold selection easier for domain experts. The MTS search algorithm must return all matches (with no false dismissals or false positives), consisting of the matching MTS in the database and where within the MTS the matching pattern was found (offset from the beginning of the MTS), such that the time shift constraints and the threshold constraints are satisfied.

There has been substantial research in making Univariate Time Series (UTS) search very fast on very large databases [5][14]. Therefore, one obvious approach to the MTS search problem is to search for each query variable separately within the database and then join the results while taking into account the time shift constraints. However, this may lead to much more searching than is required, leading to a substantial amount of processing time. For example, if the query consists of five variables, but searching on two variables leads to a small set of candidate matches, then a brute force search on the remaining three variables within the small candidate set would be much faster than a UTS search on the remaining three variables in the entire database. We exploit this fact in the novel algorithm that we present in this paper. However we still leverage UTS search—by doing so, we utilize existing work and advance the area of fast UTS search and also retain the flexibility of allowing queries over any desired subset of variables and with any desired time shifts among the variables, unlike existing MTS search algorithms. The specific contributions of this paper are as follows:

- We propose two algorithms a list based search algorithm (LBS) and R-tree based search algorithm RBS for efficient searching of UTS subsequences. Compared with state-of-the-art existing method on UTS subsequence search, we have a higher prune rate for our algorithms.

- Using these algorithms as the building blocks, we propose two novel MTS search algorithms which can search for arbitrary multidimensional patterns (subsequences) defined on a small subset of variables in massive MTS databases.

- To the best of our knowledge, the datasets that we have used for testing the performance of our MTS algorithms are the bigger that those reported in the literature.

The rest of the paper is organized as follows. In Section 2, we discuss in more detail related work in the areas of MTS and UTS search. In Section 3, we describe the notations and give a more precise definition of the MTS search problem. In Section 4 we describe the UTS search algorithm that we use as the core of our MTS search algorithm. This leads into Section 5 where we explain our MTS search algorithm. Section 7 describes our experiments with this algorithm and comparisons with some existing work. We provide conclusions and descriptions of future work in Section 8.

[1] We cannot release the name of the carrier due to the data sharing agreement.

2. RELATED WORK

We divide this section into related work on UTS search and MTS search.

UTS search: The topic of subsequence matching of time series has been an active area of research in the database/data mining community. Depending on the application, time series matching can be of the following categories: (1) full time series matching in which the queries are entire time series sequences, and (2) time series subsequence matching in which the queries can be of any size. Popular techniques for performing entire length time series search include the ones proposed by Keogh and Ratanamahatana [6], Sakurai *et al.* [12], Shou *et al.* [13] and the references therein. Since these techniques cannot be adopted to perform subsequence search easily, we do not consider them further in our discussion.

One of the early works of subsequence matching is by Faloutsos *et al.* [2] in which the authors have proposed a DFT/R-tree based indexing scheme. Input time series is first broken into overlapping window sequences of fixed length w and then six DFT coefficients are extracted from each sequence. These 6-dimensional representations are then packed into a minimum bounding rectangle (MBR) and indexed using an R-Tree data structure. On receiving a query, the same process is applied (extracting DFT coefficients) and then the search is performed on R-Tree. Candidate MBRs are then checked with the actual database to remove the false alarms. A dual approach to this one, proposed by Moon *et al.* [8], is to decompose the input time sequence into disjoint sequences and the query sequence into sliding windows. As a result, this technique can index data points directly instead of MBRs and thereby reduce false alarms. However, as the size of the time series increases to millions of points, storing all the points in the index may still be challenging.

To alleviate this problem, Traina *et al.* [14] recently proposed a technique of using multiple reference points to speed up the search. The idea is to randomly select multiple global reference points from the dataset, find the distances of all points from this reference point and index these distances in a tree or other index method. It has been verified that using multiple reference points, the candidate set of the search process can be significantly reduced. While our algorithm resembles this philosophy, it has the following significant differences: (1) [14] only talks about nearest neighbor and range query on the database, we show how it can be used for arbitrary subsequence matching, and (2) unlike [14] which only works for univariate time series, we adopt it for multivariate subsequence search with arbitrary number of variables and arbitrary time delays among those variables.

Several other techniques exist for subsequence matching in univariate time series databases. Due to shortage of space we only present the references here — ranked subsequence matching by Han *et al.* [4], disk resident pattern discovery by Mueen *et al.* [9], subsequence retrieval under DTW [11], and approximate embedding-based matching [1].

MTS search: There does not exist much work on multivariate time series (MTS) search. Yang and Shahabi [16] present a PCA-based similarity technique for comparing two MTS's. Given a database of MTS's this technique first computes the covariance matrix between two MTS. Then eigenvectors and eigenvalues of the covariance matrix are used as a measure of similarity between the MTSs. This work was extended in [18] in which the authors proposed the use of kernel PCA instead of traditional PCA which suffers from the curse of dimensionality. The *kernel trick* helps to solve this problem by not requiring one to explicitly compute the dot product among feature vectors.

Distance-based index structure for MTS has been discussed by Yang and Shahabi [17]. The proposed indexing scheme, known as *Muse*, builds a multi-level index structure. Unlike the PCA-based similarity index, *Muse* does not use any weights (*e.g.* eigenvalues) while constructing the index, obviating the need for changing the index whenever the weight changes. At query time, the levels are combined with the weights to generate the lower bound on the query distance to the candidates.

The work by Lee *et al.* [7] addresses the problem of searching in multi-dimensional sequences. The multi-dimensional sequence is partitioned into sequences, packed into MBR and then indexed using the R-tree scheme. The query is processed in a similar fashion to find the intersecting MBR's after which exact calculation is done. Vlachos *et al.* [15] proposes an index structure for multi-dimensional time series (2-D trajectory data) which can handle multiple distance functions such as LCSS and DTW. Similar to our proposed technique, the index is built using R-tree and queried using the minimum bounding envelope. This indexing and querying scheme can, however, only address the nearest neighbor query and not subsequence query which is the main focus of our work.

To the best of our knowledge, there does not exist any multi-dimensional search technique which (1) can perform search on any arbitrarily chosen subset of variables, and (2) take into consideration time delay between the variables in the query, both important to our particular application.

3. BACKGROUND

3.1 Notations

Let D be a database consisting of multivariate time series datasets $MT_1, \ldots MT_{|D|}$ where each MT_i can be represented as a matrix of row sequences $MT_i = \left[y_i^{(1,\cdot)} y_i^{(2,\cdot)} \ldots\right]^{\mathrm{T}}$ whose rows correspond to time instances and columns correspond to attributes or features. Each $y_i^{(j,\cdot)} = \left[y_i^{(j,1)} \ldots y_i^{(j,n)}\right]$, assuming there are n features $v_1, \ldots, v_n$ consistent across all the MT's. It is also assumed that each $y_i^{(j,\ell)} \in \mathbb{R}$ or $\{0, 1\}$. For consistency, the set of values across the ℓ-th column $y_i^{(\cdot,\ell)} = \left[y_i^{(1,\ell)} \quad y_i^{(2,\ell)} \ldots\right]$ is referred to as the ℓ-th UTS in the i-th MTS. Whenever appropriate we will drop the index i. Let $y^{(\cdot,\ell)}$ and $x^{(\cdot,\ell)}$ be two UTS sequences. Then,

- $L\left(y^{(\cdot,\ell)}\right)$ denotes the length of $y^{(\cdot,\ell)}$
- $y^{([a:b],j)}$ denotes the subsequence that includes entries in positions a through b
- $d(y^{(\cdot,\ell)}, x^{(\cdot,\ell)})$ denotes the distance between two univariate sequences (when they are of the same length).

Let w be the size of a sliding window containing w consecutive samples of a UTS. We now define ϵ-nearest neighbors (ϵ-NN).

Definition 3.1 (ϵ-NN). *Given a user defined threshold ϵ, and a univariate sequence q of length w, (which we call the query), ϵ-NN returns all the subsequences s of length w from the dataset, such that, $d(s, q) < \epsilon$.*

3.2 Problem definition

Before we present the formal problem definition, we present the definition of a query.

Definition 3.2 (Multi-variate Query). *A multivariate query Q consists of the following components:*

- *Values specified for a subset of attributes $V_q = \{v_1, v_3, v_4, \ldots, \}$ i.e. $q^{[t_1:t_2],1}, q^{[t_3:t_4],3}, q^{[t_5:t_6],4}, \ldots,$ and*
- *time delays $\delta_1, \delta_2, \delta_3, \ldots$ such that $t_3 - t_2 = \delta_1$, $t_5 - t_4 = \delta_2$, and so on.*

Definition 3.3 (Multi-variate Search (MTS)). *Given a database of multi-variate time series D, a query Q and a user-defined threshold ϵ, a MTS returns all the MT's such that for all $j \in V_q$,*

- $d(y_i^{[a:b],j}, q^{[t_j:t_k],j}) < \epsilon$, $b - a = t_k - t_j$
- *the variables are delayed by $\delta_1, \delta_2, \delta_3, \ldots$*

4. UNIVARIATE TIME SERIES SEARCH

When a query Q defined in Section 3.2 contains only one variable, it becomes a univariate time series search. For clarity and ease of exposition, we will start with solving this problem. We assume there is a minimal length for all queries and it is set to w. This value depends on different applications as we discuss in the experimental section. We first discuss the *L*ist *B*ased *S*earch (*LBS*) algorithm in details and then discuss the salient differences with our R^*-tree algorithm (*RBS*).

4.1 Algorithm approach: basic idea

For a univariate query q^v on the v-th variable, the brute-force method to find all its ϵ-NN is to compare it with all subsequences of length $L(q^v)$ for every offset of time series $y_i^{(.,v)}$ ($\forall i = 1, 2, \ldots, |D|$), which is time consuming and impractical.

A classic data mining solution to speed up this process is to find a lower bound of distance measure and use this bound to prune unpromising candidates. This lower bound should be: (1) cheaper to compute than computing the distances between all subsequences, otherwise we would spend more time; (2) tight with respect to the original distance measure, otherwise we cannot prune enough.

One such technique for deriving a lower bound, also used in the literature [14][10], is by using a reference subsequence and the triangle inequality. **We will show later that our framework to find the ϵ-NN even does not require calculating the lower bound one by one**. Figure 2 illustrates the basic idea of the pruning. First, we randomly pick a subsequence R (of the same length as w), and calculate its distance to all the remaining subsequences. Then, we order them by their distance to R. S_1 and S_2 are only shown for clarity in the figure. Note that these two steps are done before the query q^v comes and only need to be done once. When a query q^v comes, we calculate the distance $d(q^v, R)$. All candidates whose distances are not in the range $[d(q^v, R) - \epsilon, d(q^v, R) + \epsilon]$ (*e.g.* S_2 in Figure 2) can be pruned. This is due to the triangular inequality:

$$d(q^v, S_2) \geq |d(q^v, R) - d(S_2, R)| > \epsilon.$$

Finally, for all candidates in this range (*e.g.* S_1 in Figure 2), we do an exact calculation to remove the false alarms. In order to reduce the number of false alarms, we use multiple reference points to build several indices and then join the candidates from these indices to get the final set of candidates. We discuss this in details in the next section.

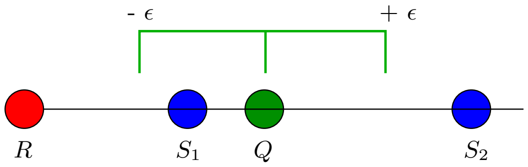


Figure 2: Candidate subsequences (S_1, S_2) ordered by their distance to a reference subsequence R. When a query Q comes, a range based on $d(Q, R)$ can be used to prune candidates.

4.2 Algorithm details

We first discuss the index building algorithm followed by the search algorithm. Alg. 1 presents the pseudo-code of *LBS* build index. The inputs are *UTS_Database* and length of the sliding window w. The output is a set of sorted lists. In the first step, we select r subsequences $R_1, \ldots, R_r$ of size w from *UTS_Database* which we call *reference points*. Then, for each overlapping subsequence S of length w from the i-th UTS in *UTS_Database*, we find the Euclidean distance of S from the ri-th reference point R_{ri}. We store these distances (as the key) along with the offset and UTS-id which generated this distance into a list called $Index_{ri}$. Thus at the end of this process, we build $|r|$ number of lists $Index_1, \ldots, Index_r$, one corresponding to each reference point. In the next step we simply sort these lists and store them in the disks either as one long list or in parts, depending on the size of the index.

Algorithm 1: Build Index for *L*ist *B*ased *S*earch (*LBS*)

```
Input: $UTS\_Database$, $w$
Output: Sorted lists $Index_1$, ..., $Index_r$
Initialization: Select $r$ reference points $R_1, \ldots, R_r$;
begin
    for $ri$ = 1 to $r$ do
        for $uts\_i$ in $UTS\_Database$ do
            for $j$ = 1 to ($L(uts\_i)$ - $w$ + 1) do
                $Dist = d(R_{ri}, uts\_i(j, j + w - 1))$;
                $New\_Entry = [uts\_i, j, Dist]$;
                $Index_{ri} \leftarrow Index_{ri} \bigcup New\_Entry$;
        Sort and save to disk $Index_{ri}$;
end
```

When a query Q of length w is provided, we use the search code shown in Alg. 2. The input in this case are the query Q, the *UTS_Database*, the set of indices, the set of reference points, w, and ϵ. The output of ϵ-NN search returns all subsequences of length w such that the distance of this with Q is less than ϵ. First, for each reference point R_i, we find the distance $Dist_i$ of the query from it. Then we collect those candidates from $Index_i$ whose key (distance) lies in the range $Dist_i \pm \epsilon$. We call this step the *first level* of pruning since we apply the triangle inequality directly here. Still many false alarms may be generated because the triangle inequality is essentially a one-sided test *i.e.* if the distance of any subsequence to any reference point is greater than ϵ, we can discard the former, but not otherwise, irrespective of the actual distance of the subsequence to the query. In the *second level* of pruning, we intersect the candidates found similarly using different reference points. This reduces the number of false alarms dramatically as we show in out experiments. Once a compact candidate set is found, we do a disk access to retrieve those candidates and remove false alarms. Note that we obtain a different candidate set if we use a different reference sequences. The size of candidate set is crucial to the running time, since we have to access the disk and perform the exact calculation to remove false alarms and return all matching candidates.

Algorithm 2: *LBS* ϵ-NN Search on UTS

```
Input: $UTS\_Dataset$, $Q$, $Index_1, \dots, Index_r$,
       $R_1, \dots, R_r$, $w$, $\epsilon$
Output: Set of $\epsilon$ nearest neighbors $\epsilon$-NN of $Q$
begin
    $\epsilon$-NN $\leftarrow \emptyset$;
    for $ri = 1$ to $r$ do
        $Dist_{ri} = d(Q, R_{ri})$;
        $Cand_{ri} = \{x \in Index_{ri} | Dist_{ri} - \epsilon \leq x \leq$
        $Dist_{ri} + \epsilon\}$;
    $Candidates \leftarrow \{\bigcap_{ri=1}^{r} Cand_{ri}\}$;
    for $c \in Candidates$ do
        Fetch $c$ from disk ;        // Actual disk access
        $Dist = d(c, Q)$;
        if $Dist \leq \epsilon$ then  $\epsilon$-NN $\leftarrow$ $\epsilon$-NN $\bigcup\{c\}$;
end
```

We now discuss now *LBS* handles queries longer than w in the following two cases:

$L(Q) = nw \quad (n > 1)$: We first divide Q into n disjoint subsequences of length w, and search the indices set for each of them with the threshold $\epsilon/\sqrt(n)$. Finally we do an exact calculation of full length candidates (over all n parts) to remove false alarms. The correctness of this approach relies on the following theorem [2].

THEOREM 4.1. *If $d(Q, S) < \epsilon$, then for at least one pair of disjoint sequences Q_i and S_i of length w, we have $d(Q_i, S_i) < \epsilon/\sqrt(n)$.*

$L(Q) = nw + v \quad (0 < v < w)$: Since we have solved the previous case, this one becomes easy. We can ignore the last subsequence of length v while searching in the index, and only consider it when we perform the exact calculation.

4.3 R^*-tree search algorithm (*RBS*)

Our detailed experimental results demonstrate that the proposed *LBS* algorithm offers a high prune rate even with a moderate number of reference points (*e.g.* 3). However the index, being a sorted list of time series points, is often huge (of the order of the number of points in the time series). This increases the storage costs. Our R^*-based search algorithm solves this problem by avoiding the need to store and index each point separately. Once a set of distances to a reference point are computed as before, we store them together into a Minimum Bounding Rectangle (MBR) and index the minimum and maximum bounds of this rectangle using a spatial indexing scheme such as R^*-tree. We have used two packing methods proposed in [2]: (1) the I-fixed method which combines a fixed number of points, and (2) the I-adaptive method which optimizes a cost function to find the optimal number of points per MBR. These resulting trees using multiple reference points become the *Index*'s for the *RBS* algorithm. When searching on Q, we perform the same transformation as *LBS* and search for $Dist_i \pm \epsilon$ in the R^*-tree. This returns a set of candidate MBRs (for each tree) which then needs to be joined to reduce false positives. Each element from the joined candidate set is retrieved from the disk to remove the false alarms. We do not present the pseudo-code here due to shortage of space.

5. MULTIVARIATE TIME SERIES SEARCH

Algorithm 3: MTS Build Index using *LBS*

```
Input: $MTS\_Database(D)$, $w$
Output: $Index$ for MTS search
Initialization: Select $R_1^{(\ell)}, \dots, R_r^{(\ell)}$ for $uts\_\ell$;
begin
    Convert entire $MTS\_Database$ into $uts_1, \dots, uts_d$;
    for $f = 1$ to $d$ do                          // all features
        for $ri = 1$ to $r$ do                     // all ref pts
            for $uts\_i$=1 to $|D|$ do // across all files
                for $j = 1$ to ($L(uts\_i)$ - $w$ + 1) do
                    $Dist = d(R_{ri}, uts\_i(j, j + w - 1))$;
                    $New\_Entry = [uts\_i, j, Dist]$;
                    $Index_{ri}^{(f)} \leftarrow Index_{ri}^{(f)} \bigcup New\_Entry$;
                Sort and save to disk $Index_{ri}^{(f)}$;
end
```

Algorithm 4: MTS ϵ-NN Search using *LBS*

```
Input: $D$, $Q$, $Index$, $R_1, \dots, R_r$, $w$, $[\epsilon_1, \dots,]$
Output: Set of $\epsilon$ nearest neighbors $\epsilon$-NN of $Q$
begin
    $\epsilon$-NN $\leftarrow \emptyset$;
    $Cand1 \leftarrow$ FindCandidates($Q^{(1)}$);
    $Cand2 \leftarrow$ FindCandidates($Q^{(2)}$);
    $Cand12 \leftarrow$ JoinCand($Cand1, Cand2, \delta_1$);
    for $c \in Cand12$ do
        Fetch $c$ from disk ;        // Actual disk access
        $Dist_1 = d(c^{(1)}, Q^{(1)})$,    $Dist_2 = d(c^{(2)}, Q^{(2)})$;
        if $Dist_1 \leq \epsilon_1$ and $Dist_2 \leq \epsilon_2$ then
            $CandA \leftarrow CandA \bigcup\{c\}$;
    $\epsilon$-NN $\leftarrow$ $\epsilon$-NN $\bigcup$ {disk-based search of remaining
    variables in $Q$ using $CandA$ and $\delta_2, \dots$};
end
```

6. ANALYSIS OF ALGORITHMS

State the diff between this method and FRM (FRM uses DFT for 6-d indexing while we use in 2-d..so it is cheaper and scales well)

We combine ref pt with R-tree for better performance and accuracy...also increasing the no of ref pts increases the prune rate which they cannot do

7. EXPERIMENTS

To validate the performance of the LBS and RBS algorithms, we ran a variety of tests on different datasets, both univariate and multivariate. All experiments were run on a 64-bit 2.33 GHz quad core dell precision 690 desktop running red hat enterprise linux version 5.4 having 2GB of physical memory. The algorithms were implemented in Matlab and run on version R2007b. In all our experiments we have measured the following four quantities:

- ω = set of nearest neighbors within radius ϵ of query Q, derived from the actual database
- C = candidate set returned by FRM, LBS and RBS
- L = length (# samples) of any UTS
- T = total number of sliding window sequences of any UTS

Using these, we derive and report the following quantities as done in the literature [2][8]:

- Selectivity $S = \frac{|\omega|}{T}$
- Prune rate $\rho = 1 - \frac{|C|}{T}$

Intuitively, selectivity refers to the true fraction of nearest neighbors for the chosen query Q and threshold ϵ, while prune rate ρ refers to the fraction of candidates that can be ignored as having distance greater than ϵ even without accessing the database. Note that for an MTS of size $|D|$ (files), the following two relations hold between any UTS of total length L (over all MTSs) and total number of sliding window sequences T:

$$L = \sum_{i=1}^{n} L_i, \quad T = \sum_{i=1}^{|D|}(L_i - w + 1) = L - |D|(w - 1)$$

where L_i is the length of any UTS in the i-th MTS.

In order to keep the comparison independent of the implementation across different platforms, we have not measured the actual running time of these algorithms. Note that all of these algorithms guarantee no false dismissals (but false alerts). Thus, actual running time is going to be proportional to the number of candidates returned, since each of these candidates need to be retrieved from the actual time series databases, and checked if their actual distance to the query (Q) is less than ϵ.

We first present results on univariate datasets, followed by results on multivariate datasets.

7.1 Univariate dataset experiments

Datasets	Length (time points)
Stock market data	329,112
Random walk data	500,000
Periodic data	1,000,000

Table 1: Description of the univariate datasets used for comparing the performance of FRM, LBS and RBS.

7.1.1 Dataset description and experimental setup

We have used three univariate datasets for testing our algorithms shown in Table 1. These datasets have been used in the literature ([2] and [8]) for finding subsequences from time series databases. Figure 3 shows a plot of these datasets.

The first dataset is the stock market dataset having 329,112 entries[2]. The second dataset is the random walk dataset generated synthetically. The first value is set to 1.5 and the subsequent values are obtained by adding a random value in the range (-0.001, 0.001) to the previous one. The last dataset is a pseudo periodic time series dataset[3] in which each value is between -0.5 and +0.5. This dataset appears highly periodic, but never repeats itself.

For all the univariate experiments, we have used the length of sliding window w=512 and length of query sequences the same as the length of sliding window. We perform experiments with several selectivities ranging from $10^{-6} \sim 10^{-1}$ [2]. The desired selectivities were achieved by modifying the threshold ϵ of each query. We tested three algorithms on these datasets: (1) the FRM algorithm using the adaptive MBR approach, details of which can be found in [2], (2) list based method (LBS), and (3) the R*-tree based method (RBS), the last two introduced in this paper. The number of reference points used is given as an argument for LBS and RBS *e.g.* RBS(5). To avoid the effects of noise and generate statistically significant results, we experimented with ten randomly generated queries each having length of w. Unless otherwise stated, we have used three to five reference points for the LBS and RBS methods. In the next section we present the thresholds, selectivities, and results on these three datasets.

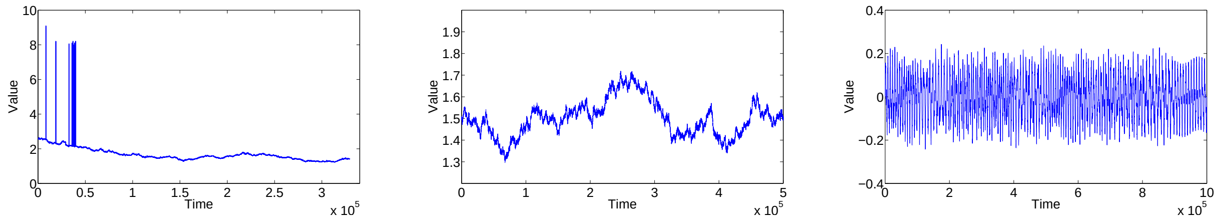
7.1.2 Results

We summarize the results of FRM, LBS and RBS in Table 2 for the stock dataset. We varied ϵ from 0.01 to 1.0 to generate selectivities in the range of $10^{-6} \sim 10^{-2}$. The table shows the prune rates and the number of nearest neighbors found in the datasets for each of these selectivities averaged over ten queries. Also shown in this table are the number of MBRs and average points per MBR for the $RBS(5)$ algorithm. We have used the *I-adaptive* MBR creation heuristic as discussed in [2], in which more points can be packed in a single MBR with larger ϵ, thereby reducing the total number of MBRs. For all the thresholds, we see that the prune rate of $LBS(3)$ is the best for all the thresholds. Also, the prune rates of $RBS(5)$ tend to be very close to the FRM algorithm.

We have similar results for the random walk dataset in Table 3. In this case also, the selectivity ranges from 10^{-6}

[2] Available from `ftp://ftp.santafe.edu/`

[3] Available from `http://archive.ics.uci.edu/ml/datasets/Pseudo+Periodic+Synthetic+Time+Series`



(a) Stock market dataset (b) Random walk dataset (c) Periodic dataset

Figure 3: Plots of univariate time series datasets used in our experiments.

ϵ	Prune rate (ρ)				#MBR	Pts/MBR	ω	S
	FRM	LBS(3)	RBS(3)	RBS(5)				
0.01	0.9762 ± 0.02	**0.9995** ± 0.001	0.994 ± 0.001	**0.9948** ± 0.002	2136	154	1	3.04E-06
0.1	0.9607 ± 0.03	**0.9901** ± 0.007	0.9727 ± 0.005	**0.9753** ± 0.015	2732	601	213	6.52E-04
0.5	0.92 ± 0.059	**0.9508** ± 0.036	0.9195 ± 0.016	**0.9213** ± 0.051	1253	1311	14094	4.29E-02
0.75	0.8969 ± 0.075	**0.9246** ± 0.055	0.8957 ± 0.021	**0.8974** ± 0.066	1007	1631	23510	7.16E-02
1.0	0.871 ± 0.083	**0.9017** ± 0.067	0.8683 ± 0.026	**0.8689** ± 0.084	861	1908	31433	9.57E-02

Table 2: Results of LBS and RBS on stock data (in bold). Shown are the mean and standard deviation of ρ over ten queries. In all cases, LBS shows the highest prune rate while the prune rates of RBS are comparable to FRM. Note that RBS does not require analysis in the DFT domain which cuts down index building time. Also RBS can produce better prune rates by increasing the number of reference points.

to 10^{-2}. As before, $LBS(3)$ performs the best, for all the thresholds while $RBS(5)$ performs better than FRM as the threshold is increased. For the RBS algorithm, we note that prune rate for threshold 0.01 is very close to the maximum value of 1.0 and differs from the FRM prune rate only in the third place of decimal.

The results on the periodic dataset present an interesting phenomenon. As before, the snapshots of the results are presented in Table 4. The LBS approach has the highest prune rate for all the thresholds. However, the $RBS(5)$ technique in this case performs poorly compared to the FRM technique. This can be explained noting that FRM builds MBR's in the DCT domain while RBS directly works in the input space. It is well-known that for periodic signals, DCT/DFT can extract most of the energy in the first few coefficients. Thus, the MBR's constructed by the FRM algorithm using only the first three DFT coefficients are highly condensed and informative. On the other hand, as pointed out by Moon *et al.* [8], adjacent values of the periodic dataset are relatively large. Hence adjacent windows have large distance to the reference points. When these are combined to form MBR, many windows far apart can be included in the same MBR, thereby increasing the number of candidates and false alarms. Note that the number of candidates can be reduced by increasing the number of reference points. Using around eight reference points, $RBS(8)$ has lesser number of candidates compared to FRM for this dataset. However, for fairness of comparison, we have used three reference points for both LBS and RBS when experimenting with the multivariate datasets.

To sum up, both the LBS algorithm and the RBS algorithms offer an excellent prune rate for univariate time series search. LBS offers the best prune rate of all the three algorithms compared here, but as discussed before, suffers from large storage cost — $O(n)$, where n is the number of elements in the timeseries. For a large n, this may be very expensive. On the other hand, RBS uses MBR's to group similar points and hence can reduce the storage cost dramatically. For example, the number of MBR's for the periodic dataset having 1 million data points is only about 6000, thereby reducing the search space by several orders of magnitude. However, since the unit of search is an MBR (containing several points) and not individual points, all the points in the selected MBR's need to be visited. Hence, the prune rate of RBS is lower than LBS. Nevertheless, both these algorithms have a better prune rate compared to FRM.

7.2 Multivariate dataset experiments

7.2.1 Dataset description

We have used two large multivariate datasets for demonstrating the search capabilities of LBS and RBS in the multivariate domain. These datasets that are relevant to the NASA Integrated Vehicle Health Management (IVHM) project. To the best of our knowledge, these multivariate datasets are by far much bigger compared to the datasets used in the literature for multi-dimensional time series search. The datasets are described next.

C-MAPSS dataset: The first dataset is simulated commercial aircraft engine data. This data was generated using the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) [3]. C-MAPSS is a high-fidelity system level engine simulator designed to simulate nominal and fault engine degradation over a series of flights. The dataset contains 6875 full flight recordings sampled at 1 Hz with 29 engine and flight condition parameters recorded over a 90 minute flight that includes ascent to cruise at 35000 feet and descent back to sea level. This dataset has over 32 mil-

ϵ	Prune rate (ρ)				#MBR	Pts/MBR	ω	S
	FRM	LBS(3)	RBS(3)	RBS(5)				
0.01	0.9913 ± 0.008	**0.9992** ± 0.001	0.9915 ± 0.007	**0.9928** ± 0.007	14920	167	1	2.00E-06
0.05	0.9772 ± 0.021	**0.9931** ± 0.007	0.9737 ± 0.016	**0.9765** ± 0.01	5258	475	31	6.19E-05
0.1	0.9603 ± 0.028	**0.9832** ± 0.012	0.956 ± 0.024	**0.9613** ± 0.027	3575	699	539	1.07E-03
0.2	0.934 ± 0.042	**0.9623** ± 0.024	0.9151 ± 0.04	**0.9323** ± 0.041	2401	1040	9756	1.95E-02
0.4	0.8919 ± 0.066	**0.8978** ± 0.057	0.8135 ± 0.068	**0.9054** ± 0.068	1670	1496	37326	7.47E-02

Table 3: Results of LBS and RBS on randomwalk data (in bold). Shown are the mean and standard deviation of ρ over ten queries. In all cases, LBS shows the highest prune rate while the prune rates of RBS are comparable to FRM. Note that RBS does not require analysis in the DFT domain which cuts down index building time. Also RBS can produce better prune rates by increasing the number of reference points.

ϵ	Prune rate (ρ)				#MBR	Pts/MBR	ω	S
	FRM	LBS(3)	RBS(3)	RBS(5)				
0.05	0.9749 ± 0.012	**0.9983** ± 0.001	0.889 ± 0.024	**0.9119** ± 0.017	6441	775	56	5.57E-05
0.1	0.9599 ± 0.017	**0.9928** ± 0.003	0.8639 ± 0.032	**0.8918** ± 0.019	5080	984	347	3.47E-04
0.2	0.928 ± 0.025	**0.9772** ± 0.009	0.8141 ± 0.048	**0.8506** ± 0.029	4098	1220	4131	4.14E-03
0.6	0.8068 ± 0.046	**0.8956** ± 0.023	0.6694 ± 0.106	**0.7266** ± 0.084	2971	1682	60191	6.02E-02

Table 4: Results of LBS and RBS on periodic data (in bold). Shown are the mean and standard deviation of ρ over ten queries. In all cases, LBS shows the highest prune rate while the prune rates of RBS are comparable to FRM. Note that RBS does not require analysis in the DFT domain which cuts down index building time. Also RBS can produce better prune rates by increasing the number of reference points.

lion tuples. Since some of the variables do not show much variability, we have tested our algorithm on a subset of 16 variables only. Table 5 presents the salient features of this dataset. Interested readers can download this dataset from: Dashlink[4].

US Regional carrier dataset (CarrierX): The second dataset is a real life commercial aviation dataset of a US regional carrier consisting of 3573 flights. Each flight contains 46 variables. Domain experts identified a subset of 9 variables combination of which are critical to assess the health of such aircraft systems. The entire dataset contains more than 22 million tuples.

For all the multivariate experiments, we have used sliding window size and length of query equal to 256. For LBS we have used three reference points, while for RBS we have used five reference points for building the indices. These choices are based on the prune rates of these algorithms on univariate datasets.

Datasets	# MTS $\|D\|$	Features	L
CMAPSS	6875	16	32,640,967
CarrierX	3573	9	22,207,852

Table 5: Description of the multivariate datasets used for demonstrating performance of LBS and RBS.

7.2.2 Results

Table 6 presents three sets of thresholds for each of the variables of the CMAPSS and CarrierX dataset. The choice

Data set	Variable Number	Thresholds		
		ϵ_1	ϵ_2	ϵ_3
CMAPSS	2	100	300	500
	4	1	5	10
	5	0.5	1	2
	6	50	100	200
	8	0.02	0.04	2
	15	0.1	0.5	1
	18	0.1	1	5
	20	0.01	0.02	0.03
	22	0.2	0.5	5
	23	0.3	0.5	0.8
	24	5	20	40
	25	0.2	0.5	1
	26	0.0001	0.0005	0.001
	27	2	5	10
	28	5	10	20
	29	0.5	1	2
CarrierX	6	10	15	20
	7	10	30	50
	8	1500	2500	3500
	23	2	4	6
	27	100	500	1000
	28	1000	1500	4000
	29	100	300	500
	30	10	50	100
	38	2	3	3.5

Table 6: Thresholds for the variables of CMAPSS and CarrierX dataset.

[4] `https://dashlink.arc.nasa.gov/data/c-mapss-aircraft-engine-simulator-data/`

of these thresholds is such that the selectivities of each variable independently ranges from $10^{-6} \sim 10^{-2}$.

The performance results of LBS and RBS on CMAPSS and CarrierX are presented in Table 7. The first column denotes the dataset. The second column refers to the five different queries we have run along with the variables for each query. We have run each query with three different thresholds (hence the three rows for each query) presented in the table in increasing order. For example, using Table 6, it can be concluded that $\epsilon_1 = (0.2, 2, 1)$ for the first query of CMAPSS. The next three columns show the number of candidates generated for the first variable (C_1), the second variable (C_2), and after joining these two candidate sets C_{12} both for LBS and RBS. The join on the candidate set is performed based on two criterion: (1) the time delay between any two candidates must conform to the ones specified in the query, and (2) they should be generated from the same MTS. Generating C_1 and C_2 is the first level of filtering while generating C_{12} refers to the second level of filtering. Column C_e is the actual number of these candidates which are found to be less than the threshold after doing the exact calculation. So smaller the size of C_{12}, the lesser the number of actual disk elements that need to be accessed. ω column refers to the actual number of nearest neighbors of the query after taking all the variables and time delays into consideration. The last two columns show the prune rate $\rho = C_{12}/T$ and selectivity $S = \omega/T$ respectively.

These results show that for the two large multivariate datasets, querying with different queries and thresholds, the prune rates are very high $\sim 99\%$ implying that only less than 1% of the candidates need to be retrieved from the database for exact calculation. Also, we notice that the sizes of the candidate sets are smaller for LBS than RBS for all the queries thereby raising lesser number of false positives. However, the storage requirements of LBS is non-trivial. For example, for CarrierX, we need to index approximately 22 million distances using each reference point per UTS. The total storage requirement for the index will be

$$22,000,000 \times (4 + 4 + 4)/(1024 \times 1024) \approx 250 \text{ MBytes},$$

for each UTS, assuming we store {distance, MTS_id, Offset} for each window sequence as a float of (4+4+4) bytes. For RBS, let's assume that (1) we have M MBR's on average for each reference point, and (2) we store {min_MBR, max_MBR, MTS_id, Offset} for each MBR. In our experiments we have $M = 5174619$. Then the total storage requirements (assuming 4 bytes for each) will be:

$$5,174,619 \times (4 + 4 + 4 + 4)/(1024 \times 1024) \approx 78 \text{ MBytes}.$$

Hence the storage requirements of RBS is much less than LBS. From these results we conclude that:

- query execution time of LBS is expected to be much faster than RBS due to higher prune rate
- RBS has relatively higher rate of false positives compared to LBS
- the index storage requirements of LBS may be significantly higher compared to RBS

8. CONCLUSION

In this paper we present two algorithms LBS and RBS for finding multivariate subsequences from large MTS datasets. Both these algorithms guarantee no false dismissals. To demonstrate the prune rate, first we have run experiments on several UTS datasets used in the literature for subsequence search. The results show that LBS offers the best prune rate of all the three algorithms compared in this paper. RBS has a lower prune rate due to the search unit being an MBRs and not individual points, but builds smaller indices. Experiments on two commercial aviation related MTS datasets each having millions of tuples show that both these algorithms offer excellent prune rates (greater than 0.9)> The latter implies that we need to retrieve only a small percentage (< 1%) of all the candidates for post processing in order to eliminate false positives. To the best of our knowledge, this is the first algorithm which allows extremely fast and flexible pattern/subsequence search in massive multivariate time series datasets on any subset of variables with time delays between them. Also the CMAPSS and CarrierX datasets that we have tested are the much bigger than any of the MTS datasets used in the literature for multivariate subsequence search. As a future work, we plan to develop a parallel and fully decentralized implementation of this MTS search technique on a Map-Reduce framework for better scalability.

Acknowledgements

This work was supported by the NASA Aviation Safety Program, Integrated Vehicle Health Management Project and a Google Annex project.

9. REFERENCES

[1] V. Athitsos, P. Papapetrou, M. Potamias, G. Kollios, and D. Gunopulos. Approximate Embedding-Based Subsequence Matching of Time Series. In *Proceedings of SIGMOD'08*, pages 365–378, 2008.

[2] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast Subsequence Matching in Time-series Databases. *SIGMOD Rec.*, 23(2):419–429, 1994.

[3] D. K. Frederick, J. A. DeCastro, and J. S. Litt. User's Guide for the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS). *NASA Technical Manuscript*, 2007-215026, 2007.

[4] W. Han, J. Lee, Y. Moon, and H. Jiang. Ranked Subsequence Matching in Time-Series Databases. In *Proceedings of VLDB'07*, pages 423–434, 2007.

[5] J. J. Shieh and E. Keogh. iSAX: Disk-aware Mining and Indexing of Massive Time Series Datasets. *Data Min. Knowl. Discov.*, 19(1):24–57, 2009.

[6] E. Keogh and C. A. Ratanamahatana. Exact Indexing of Dynamic Time Warping. *Knowl. Inf. Syst.*, 7(3):358–386, 2005.

[7] S. Lee, S. Chun, D. Kim, J. Lee, and C. Chung. Similarity Search for Multidimensional Data Sequences. In *Proceedings of ICDE'00*, pages 599–608, 2000.

[8] Y. Moon, K. Whang, and W. Loh. Duality-Based Subsequence Matching in Time-Series Databases. In *Proceedings of ICDE'01*, pages 263–272, Washington, DC, USA, 2001.

[9] A. Mueen, E. Keogh, and N. Bigdely-Shamlo. Finding Time Series Motifs in Disk-Resident Data. In *Proceedings of ICDM'09*, pages 367–376, Miami, USA, 2009.

Data set	Queryid	C_1		C_2		C_{12}		C_e	ω	ρ		S
		LBS	RBS	LBS	RBS	LBS	RBS			LBS	RBS	
CMAPSS	1: (25, 27, 4)	18409	3007594	738	2477549	52	801400	6	6	0.9999	0.9741	1.94E-07
		81409	3263815	7567	2565309	2668	1003839	17	10	0.9999	0.9675	3.24E-07
		251981	3841664	81330	2702600	23694	1454776	540	297	0.9992	0.9529	9.62E-06
	2: (20, 29, 5)	53585	870835	14969	2390063	1411	266022	252	6	0.9999	0.9914	1.94E-07
		179850	1295644	50502	2454707	13862	481096	1187	17	0.9995	0.9844	5.5E-07
		317793	1587719	141444	2633060	58905	633137	20124	259	0.9981	0.9795	8.38E-06
	3: (5, 15, 28)	528470	4753958	14725	306706	6171	290593	453	8	0.9998	0.9906	2.59E-07
		1137522	4861533	87236	425813	63690	399972	16289	121	0.9979	0.9871	3.92E-06
		2115994		177992		174391		79332	1445	0.9944		4.68E-05
	4: (26, 5, 27)	1311	2013861	57144	3655449	344	86193	5	3	0.9999	0.9972	9.71E-08
		34492	2143905	193974	3894274	8034	194616	2060	337	0.9997	0.9937	1.09E-05
		115350		501207		38648		22034	6471	0.9987		2.1E-04
	5: (5, 23, 2)	101344		74609		12945		18	9	0.9996		2.91E-07
		316085		164881		49908		332	49	0.9983		1.59E-06
		771259		337201		150020		4925	479	0.9951		1.55E-05
CarrierX	1: (29, 23, 28)	26235	469928	55610	530788	96	10226	3	3	0.9999	0.9995	1.41E-07
		79606	523225	204310	716418	952	14391	15	15	0.9999	0.9993	7.04E-07
		133451	583050	374437	896063	2640	20771	27	27	0.9998	0.999	1.27E-06
	2: (8, 28, 27)	17338	1120516	16541	74930	450	26361	3	1	0.9999	0.9987	4.7E-08
		48149	1174920	62316	267710	3595	92246	7	3	0.9998	0.9957	1.41E-07
		83177	1218440	1577348	3028623	54214	754404	885	9	0.9974	0.9645	4.22E-07
	3: (38, 8, 29)	935844	870535	223138	391564	71342	94594	12318	7	0.9966	0.9955	3.29E-07
		1500995	1369274	379346	555599	175800	213822	48395	64	0.9917	0.9899	3.01E-06
		1760160	1564834	527712	705614	277017	313020	102401	269	0.9869	0.9853	1.26E-05
	4: (6, 27, 30)	22039	2164753	13866	901583	71	402047	10	10	0.9999	0.9811	4.69E-07
		103096	2289089	156448	1033504	2204	477704	30	30	0.9998	0.9775	1.41E-06
		213954	2429383	351061	1196446	9408	568003	48	48	0.9995	0.9733	2.25E-06
	5: (28, 8, 29)	1298247	2671533	184660	1649628	76445	476399	47559	2	0.9964	0.9776	9.39E-08
		1947774	3368141	205164	129643	105286	29617	78467	125	0.9951	0.9986	5.87E-06
		5161965	6417365	227501	1735525	168155	972349	136137	882	0.9921	0.9543	4.14E-05

Table 7: Results of *LBS* and *RBS* CMAPSS and CarrierX dataset for five different queries and three different thresholds per query. For both *LBS* and *RBS*, the prune rates are always greater than 0.9, signifying that less than 1% of the candidates need to be retrieved from the MTS database for exact calculations. *LBS* has significantly lesser number of candidates *i.e.* false alarms C_{12} compared to *RBS*.

[10] A. Mueen, E. Keogh, Q. Zhu, S. Cash, and M. Westover. Exact Discovery of Time Series Motifs. In *Proceedings of SDM'09*, pages 473–484, 2009.

[11] S. Park, S. Kim, and W. Chu. Segment-based Approach for Subsequence Searches in Sequence Databases. In *Proceedings of SAC'01*, pages 248–252, 2001.

[12] Y. Sakurai, M. Yoshikawa, and C. Faloutsos. FTW: Fast Similarity Search Under the Time Warping Distance. In *Proceedings of PODS'05*, pages 326–337, 2005.

[13] Y. Shou, N. Mamoulis, and D. Cheung. Fast and Exact Warping of Time Series Using Adaptive Segmental Approximations. *Mach. Learn.*, 58(2-3):231–267, 2005.

[14] C. Traina, R. Filho, A. Traina, M. Vieira, and C. Faloutsos. The Omni Family of All-purpose Access Methods: A Simple and Effective Way to Make Similarity Search More Efficient. *The VLDB Journal*, 16:483–505, 2007.

[15] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures. In *Proceedings of KDD'03*, pages 216–225, New York, NY, USA, 2003.

[16] K. Yang and C. Shahabi. A PCA-based Similarity Measure for Multivariate Time Series. In *Proceedings of MMDB'04*, pages 65–74, 2004.

[17] K. Yang and C. Shahabi. A Multilevel Distance-Based Index Structure for Multivariate Time Series. In *Proceedings of TIME'05*, pages 65–73, Washington, DC, USA, 2005.

[18] K. Yang and C. Shahabi. A PCA-based Kernel for Kernel PCA on Multivariate Time Series. In *Proceedings of ICDM'05 Workshops*, pages 149–156, 2005.

[19] K. Yang and C. Shahabi. An Efficient k Nearest Neighbor Search for Multivariate Time Series. *Inf. Comput.*, 205(1):65–98, 2007.