Embedding-based Subsequence Matching in Time Series Databases

PANAGIOTIS PAPAPETROU, Aalto University, Finland
VASSILIS ATHITSOS, University of Texas at Arlington, TX
MICHALIS POTAMIAS, Boston University, MA
GEORGE KOLLIOS, Boston University, MA
DIMITRIOS GUNOPULOS, University of Athens, Greece

We propose an embedding-based framework for subsequence matching in time series databases that improves the efficiency of processing subsequence matching queries under the Dynamic Time Warping (DTW) distance measure. This framework partially reduces subsequence matching to vector matching, using an embedding that maps each query sequence to a vector and each database time series into a sequence of vectors. The database embedding is computed off-line, as a preprocessing step. At runtime, given a query object, an embedding of that object is computed online. Relatively few areas of interest are efficiently identified in the database sequences by comparing the embedding of the query with the database vectors. Those areas of interest are then fully explored using the exact DTW-based subsequence matching algorithm. We apply the proposed framework to define two specific methods. The first method focuses on time series subsequence matching under unconstrained Dynamic Time Warping. The second method targets subsequence matching under constrained Dynamic Time Warping (cDTW), where warping paths are not allowed to stray too much off the diagonal. In our experiments, good trade-offs between retrieval accuracy and retrieval efficiency are obtained for both methods, and the results are competitive with respect to current state-of-the-art methods.

Categories and Subject Descriptors: H.3.1 [**Content Analysis and Indexing**]: Indexing methods; H.2.8 [**Database Applications**]: Data Mining; H.2.4 [**Systems**]: Multimedia Databases

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: embedding methods, similarity matching, nearest neighbor retrieval, non-Euclidean spaces, non-metric spaces.

ACM Reference Format:
Papapetrou, P., Athitsos, V., Potamias, M., Kollios, G., and Gunopulos, D 2011. Embedding-based Subsequence Matching in Time Series Databases. ACM Trans. Datab. Syst. 0, 0, Article 40 (2011), 40 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

Author's addresses: P. Papapetrou, Department of Information and Computer Science, Aalto University, Finland; V. Athitsos, Computer Science and Engineering Department, University of Texas at Arlington, USA; M. Potamias and G. Kollios, Computer Science Department, Boston University, USA; D. Gunopulos, Department of Informatics and Telecommunications, University of Athens, Greece.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Sequential data occur in a wide range of real-world applications. For example, time series are used for representing data in diverse areas, including scientific measurements, financial data, audio, video and human activity. Biological sequences, such as proteins and DNA, are crucial building blocks of living organisms; analyzing and understanding such sequences is a topic of enormous scientific and social interest. Consequently, in multiple domains, large databases of sequences are used as repositories of knowledge about those domains. At the same time, retrieving information of interest in such repositories becomes a challenging task, due to the large amounts of data that needs to be searched.

Subsequence matching is the problem of identifying, given a query sequence and a database of sequences, the database *subsequence* that best matches the query sequence. Achieving efficient subsequence matching is an important problem in domains where the database sequences are much longer than the queries, and where the best subsequence match for a query can start and end at any position of any database sequence. Motivating applications include keyword-based search in handwritten documents, DNA and protein matching, query-by-humming, etc.

Time series data naturally appear in a wide variety of domains, including financial data (e.g. stock values), scientific measurements (e.g. temperature, humidity, earthquakes), medical data (e.g. electrocardiograms), audio, video and human activity. Improved algorithms for time series subsequence matching can make a big difference in real-world applications such as query by humming [Zhu and Shasha 2003], word spotting in handwritten documents, and content-based retrieval in large video databases and motion capture databases. One commonly used similarity measure is the Euclidean Distance and generally the L_p measures. However, these measures fail to identify misalignments and warps in the time axis. Typically, similarity between time series is measured using dynamic time warping (DTW) [Kruskal and Liberman 1983], which is indeed robust to misalignments and time warps, and has given very good experimental results for applications such as time series mining and classification [Keogh 2002].

The classical DTW algorithm can be applied for full sequence matching, so as to compute the distance between two time series. With small modifications, the DTW algorithm can also be used for subsequence matching, so as to find, for one time series, the best matching subsequence in another time series [Lee and Kim 1999; Morguet and Lang 1998; Oka 1998; Sakurai et al. 2007]. Constrained Dynamic Time Warping (cDTW) is a modification of DTW that places constraints on the possible alignment between two sequences [Keogh 2002; Sakurai et al. 2005]. It has been shown that in many cases of interest these constraints improve both accuracy and efficiency [Keogh 2002; Ratanamahatana and Keogh 2005].

DTW can be used both for full sequence and for subsequence matching, and can identify the globally optimal subsequence match for a query in time linear to the length of the database [Lee and Kim 1999; Morguet and Lang 1998; Oka 1998; Sakurai et al. 2007]. While this complexity is definitely attractive compared to exhaustively matching the query with every possible database subsequence, in practice, subsequence matching is still a computationally expensive operation in many real-world applications, especially in the presence of large database sizes.

This paper proposes an embedding-based framework for improving the efficiency of processing subsequence matching queries in time series databases under the Dynamic Time Warping distance measure. The key idea is that the subsequence matching problem can be partially converted to the much more manageable problem of nearest neighbor retrieval in a real-valued vector space. This conversion is achieved by defining an

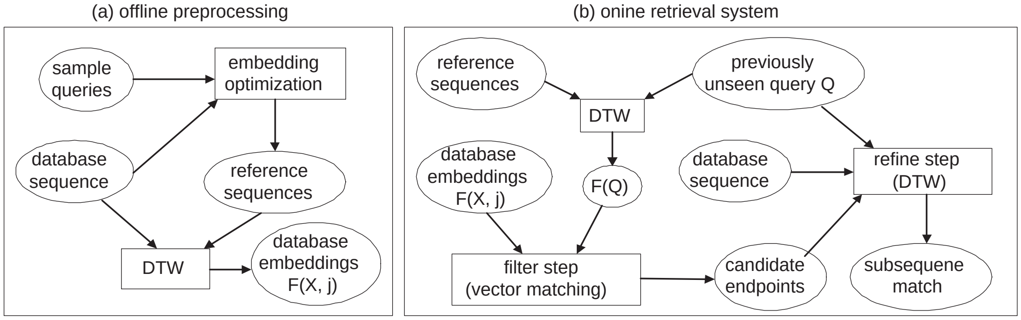

Fig. 1. Flowchart of the offline and the online stages of the proposed method. System modules are shown as rectangles, and input/output arguments are shown as ellipses. The goal of the online stage is to identify, given a query time series Q, its optimal subsequence match in the database.

embedding that maps each database sequence into a sequence of vectors. There is a one-to-one correspondence between each such vector and a position in the database sequence. The embedding also maps each query series into a vector, in such a way that if the query is very similar to a subsequence, the embedding of the query is likely to be similar to the vector corresponding to the endpoint of that subsequence.

These embeddings are computed by matching queries and database sequences with so-called *reference sequences*, i.e., a relatively small number of preselected sequences. The expensive operation of matching database and reference sequences is performed offline. At runtime, the query time series is mapped to a vector by matching the query with the reference sequences, which is typically orders of magnitude faster than matching the query with all database sequences. Then, promising candidates for the best subsequence match are identified by finding the nearest neighbors of the query vector among the database vectors. An additional refinement step is performed, where subsequences corresponding to the top vector-based matches are evaluated using the DTW algorithm. Figure 1 illustrates the flowchart of the offline and the online stages of the proposed method.

Converting subsequence matching to vector retrieval is computationally advantageous for the following reasons:

- Identifying candidate subsequence matches using the proposed framework involves vector comparisons. Under certain conditions (i.e., if the dimensionality of the embedding is small compared to the length of the query sequences), performing the required vector comparisons can be done significantly faster than running Dynamic Time Warping to match the query sequence with the entire database.
- Sampling and dimensionality reduction methods can easily be applied to further reduce the amount of time per query required for vector matching, and the amount of storage required for the database vectors.
- Numerous internal-memory and external-memory indexing methods exist for speeding up nearest neighbor retrieval in vector and metric spaces [Böhm et al. 2001; Hjaltason and Samet 2003b; White and Jain 1996]. Converting subsequence matching to a vector retrieval problem allows us to use such methods for additional computational savings.

We apply the proposed framework to define two specific methods:

- Embedding-based Subsequence Matching (EBSM) is the simplest among the two methods. It is formulated for unconstrained DTW, and it can also be applied for constrained DTW (cDTW).
- Bidirectional Subsequence Embedding (BSE) builds on top of EBSM, and is formulated to take advantage of the additional constraints available in cDTW.

Both EBSM and BSE are approximate, meaning that they do not guarantee retrieving the correct subsequence match for every query. Performance can be easily tuned to provide different trade-offs between accuracy and efficiency. In the experiments, both methods produce good accuracy/efficiency trade-offs, by significantly speeding up subsequence match retrieval, even when only small losses in retrieval accuracy (incorrect results for less than 1% of the queries) are allowed.

2. RELATED WORK

A large body of literature addresses the problem of efficient sequence matching. Several methods assume that sequence similarity is measured using the Euclidean distance [Faloutsos et al. 1994; Chan and Fu 1999; Moon et al. 2002; Moon et al. 2001] or variants [Argyros and Ermopoulos 2003; Rafiei and Mendelzon 1997; Wu et al. 2005]. However, such methods cannot handle even the smallest misalignment caused by time warps, insertions, or deletions. Robustness to misalignments is achieved using distance measures based on dynamic programming (DP), such as dynamic time warping (DTW) [Kruskal and Liberman 1983] and edit distance based approaches [Levenshtein 1966; Vlachos et al. 2002; Chen and Ng 2004; Latecki et al. 2005; Chen et al. 2005; Morse and Patel 2007]. In the remaining discussion we restrict our attention to the dynamic time warping distance measure, which is the most popular measure for time series.

Sequence matching methods can be divided into two categories: 1). methods for full sequence matching, where the best matches for a query are constrained to be entire database sequences, and 2). methods for subsequence matching, where the best matches for a query can be arbitrary subsequences of database sequences. Several well-known methods only address full sequence matching [Keogh 2002; Sakurai et al. 2005; Vlachos et al. 2003; Yi et al. 1998; Assent et al. 2009], and cannot be easily used for efficient retrieval of subsequences.

Some methods reduce subsequence matching to full sequence matching, by cutting database sequences into small pieces, and requiring each query to correspond to an entire such piece. One example is the query-by-humming system described in [Zhu and Shasha 2003], where each database song is cut into smaller, disjoint pieces. Another example is the method for word search in handwritten documents described in [Rath and Manmatha 2003], where, as preprocessing, the documents are segmented automatically into words, and full sequence matching is performed between query words and database words. Such approaches fail when the query corresponds to a database subsequence that is not stored as a single piece.

Indexing methods for sequence matching can be further subdivided based on the underlying distance or similarity measure that they target. For strings, popular distance/similarity measures are the edit distance [Levenshtein 1966] and the Smith-Waterman measure [Smith and Waterman 1981]. Strings are sequences of discrete symbols, sampled from an oftentimes small alphabet, e.g., an alphabet of 20 letters for protein sequences, and an alphabet of four letters for DNA sequences. OASIS [Meek et al. 2003] employs a best first search technique over a suffix tree for string alignment under the Smith-Waterman measure, and achieves significant speed-ups over brute-force application of Smith-Waterman. However, such methods are not directly applicable to the problem of subsequence matching of time series, which typically uses symbols obtained from a continuous space, such as the space of real numbers or higher-dimensional vector spaces. One way of retrofitting these methods for time series subsequence matching is to transform both the time series database and the query into strings [Keogh and Lin 2005; Lin et al. 2007] and use these methods directly for retrieving the best match. Such approach, however, will produce results different from those given by subsequence matching under DTW. In this paper, we focus on speeding up subsequence matching in databases of time series under DTW-based distances. Moreover, the limited number of symbols in strings has been exploited to design indexing methods based on q-grams, e.g., [Burkhardt et al. 1999; Li et al. 2007] and suffix-trees, e.g., [Meek et al. 2003; Navarro and Baeza-Yates 1999], while other embedding-based methods have been developed for subsequence matching in large string databases [Papapetrou et al. 2009]. Applying such methods to our problem setting requires discretizing time series (i.e., real values), which is not a very trivial task.

An indexing structure for unconstrained DP-based subsequence matching of time series is proposed in [Park et al. 2003]. However, as database sequences get longer, the time complexity for that method becomes similar to that of unoptimized DP-based matching. The method in [Park et al. 2001] can handle such long database sequences; the key idea is to speed up DTW by reducing the length of both query and database sequences. The length is reduced by representing sequences as ordered lists of monotonically increasing or decreasing segments. By using monotonicity, that method is only applicable to 1D time series. A related method that can be used for multidimensional timeseries is proposed in [Keogh and Pazzani 2000]. In that method, time series are approximated by shorter sequences, obtained by replacing each constant-length part of the original sequence with the average value over that part.

A method for improving the efficiency of subsequence matching under DTW is described in [Zhou and Wong 2008]. In that method, it is assumed that the length of the optimal subsequence is known, and equal to the length of the query. For unconstrained DTW, the method proposed in [Zhou and Wong 2008] has a best-case complexity of $O(mn)$, where m is the size of the query and n is the size of the long sequence that we search for subsequence matches.

The SPRING method for efficient subsequence matching under unconstrained DTW is proposed in [Sakurai et al. 2007]. In that method, optimal subsequence matches are identified by performing full sequence matching between the query and each database sequence. Subsequences are identified by prepending to each query a “null” symbol that matches any sequence prefix with zero cost. The time complexity of that method is linear to both database size and query size, which matches the best-case complexity of [Zhou and Wong 2008]. Consequently, the SPRING method is as fast as or faster than the method of [Zhou and Wong 2008]. Furthermore, unlike [Zhou and Wong 2008], SPRING does not place any constraint on the length of the subsequence match. The EBSM method described in this paper also does not place any constraint on the length of the subsequence match. In our experiments EBSM is significantly faster than SPRING, at the cost of some loss in retrieval accuracy.

A method to process motion capture data using extensions of the time warping distance is discussed in [Chen et al. 2009]. In particular, they present an approach to find efficiently (non trivial) subsequences that match between two time series based on motion capture data.

A method to handle shifting and scaling in both temporal and amplitude dimensions based on time warping distance, that is called Spatial Assembling Distance (SpADe), is presented in [Chen et al. 2007].

The powerful lower-bounding method LB_Keogh for efficient time series matching is described in [Keogh 2002]. The main idea is to use the warping constraint to create an envelope around the query sequence. Then, using a sliding window of size equal to the query, we can estimate a lower bound of the matching cost between the query and each possible subsequence. Since LB_Keogh gives a lower bound on the actual distance, this approach can be used to prune a large number of subsequences. For the subsequences that cannot be pruned, the exact dynamic programming algorithm is used to compute the distances and ultimately find the best match. However, as shown in our experiments, performance of LB_Keogh is highly dependent on the warping width parameter w and the query size; performance deteriorates as warping width and query size increase. Our proposed method, achieves significant speedups even for high warping widths and long query sizes (1000). Furthermore, computing the LB_Keogh for each possible subsequence can be time consuming for large databases. Note that, although some improvements to the LB_Keogh have been proposed (e.g. [Shou et al. 2005; Vlachos et al. 2004]), these improvements achieve not more than a small constant factor in terms of both the tightness of the lower bound and the query time performance. Therefore, we can use this approach (LB_Keogh) as a good yardstick to evaluate the performance of our method.

The DTK method [Han et al. 2007] is a method for subsequence matching under cDTW. DTK breaks the database into small non-overlapping sequences and further employs the piece-wise approximation method (PAA), described in [Keogh and Pazzani 2000], for efficient indexing. This approach however, does not scale well as the query size increases, as shown in our experiments. A similar approach is used to index time series for sequence and subsequence matching under scaling and dynamic time warping [Fu et al. 2008]. Actually, when the scaling factor is 1 (no scaling at all), the index and the query algorithm in [Han et al. 2007] are the same as the ones in [Fu et al. 2008]. Therefore, since here we do not consider scaling, we just use the DTK as a competitor to our method.

The framework proposed in this paper is embedding-based. Several embedding methods exist in the literature for speeding up distance computations and nearest neighbor retrieval. Examples of such methods include include Lipschitz embeddings [Hjaltason and Samet 2003a], FastMap [Faloutsos and Lin 1995], MetricMap [Wang et al. 2000], SparseMap [Hristescu and Farach-Colton 1999], and BoostMap [Athitsos et al. 2004; Athitsos et al. 2005]. Such embeddings can be used for speeding up full sequence matching, as done for example in [Athitsos et al. 2004; Athitsos et al. 2005; Hristescu and Farach-Colton 1999]. However, the above-mentioned embedding methods can only be used for full sequence matching, not subsequence matching.

Using the proposed framework we define two specific methods: embedding-based subsequence matching (EBSM) for unconstrained DTW, and Bidirectional Subsequence Embedding (BSE) for cDTW. EBSM was first introduced in [Athitsos et al. 2008]. As mentioned earlier, EBSM uses embeddings to define an efficient filtering process which, given a query, quickly identifies a relatively small number of promising candidate matches. Compared to the method described in [Athitsos et al. 2008], in this paper we substantially improve the efficiency of this filtering process, using two distinct and complementary approaches. The first improvement is described in Section 5.2.2, and utilizes the fact that embeddings of nearby database positions tend to have similar values. We use that fact to produce a compressed reprentation of those embeddings, that leads to faster identification of the most similar matches with the embedding of the query. The second improvement is described in Section 5.2.3, and uses the PDTW method described in [Keogh and Pazzani 2000] as a second filtering approach, that further narrows down the list of candidate matches obtained using embeddings. Also, in this paper we include experimental evaluations on a much larger dataset compared to [Athitsos et al. 2008].

The second method introduced in this paper, Bidirectional Subsequence Embedding (BSE), is novel, and has not been published before. BSE is useful when the underlying distance measure is constrained DTW (cDTW). EBSM can be used both with cDTW and unconstrained DTW. However, in the case of cDTW, BSE exploits the additional constraints (compared to unconstrained DTW) to provide additional gains in performance over EBSM. Sections 3.3 and 3.4 describe the definitions of the unconstrained and constrained versions of DTW, and highlight the differences between these two versions. In Section 8 we describe how the BSE method exploits the additional constraints of cDTW, so as to achieve better performance than EBSM.

3. BACKGROUND

In this section we define dynamic time warping (DTW), both as a distance measure between time series, and as an algorithm for evaluating similarity between time series. We follow to a large extent the descriptions in [Keogh 2002] and [Sakurai et al. 2007]. We use the following notation:

- Q, X, R, and S are sequences (i.e., time series). Q is typically a query sequence, X is typically a database sequence, R is typically a reference sequence, and S can be any sequence whatsoever.
- $|S|$ denotes the length of any sequence S.
- S_t denotes the t^{th} step of sequence S. In other words, $S = (S_1, \ldots, S_{|S|})$.
- $S^{i:j}$ denotes the subsequence of S starting at position i and ending at position j. In other words, $S^{i:j} = (S_i, \ldots, S_j)$, $S^{i:j}_t$ is the t^{th} step of $S^{i:j}$, and $S^{i:j}_t = S_{i+t-1}$.
- $D_{\text{full}}(Q, X)$ denotes the full sequence matching cost between Q and X. In full matching, Q_1 is constrained to match with X_1, and $Q_{|Q|}$ is constrained to match with $X_{|X|}$.
- $D(Q, X)$ denotes the subsequence matching cost between sequences Q and X. This cost is **a**symmetric: we find the subsequence $X^{i:j}$ of X (where X is typically a large database sequence) that minimizes $D_{\text{full}}(Q, X^{i:j})$ (where Q is typically a query).
- $D_{i,j}(Q, X)$ denotes the smallest possible cost of matching $(Q_1, \ldots, Q_i)$ to any suffix of $(X_1, \ldots, X_j)$ (i.e., Q_1 does not have to match X_1, but Q_i has to match with X_j). $D_{i,j}(Q, X)$ is also defined for $i = 0$ and $j = 0$, as specified below.
- $D_j(Q, X)$ denotes the smallest possible cost of matching Q to any suffix of $(X_1, \ldots, X_j)$ (i.e., Q_1 does not have to match X_1, but $Q_{|Q|}$ has to match with X_j). Obviously, $D_j(Q, X) = D_{|Q|,j}(Q, X)$.
- $\|X_i - Y_j\|$ denotes the distance between X_i and Y_j.

Given a query sequence Q and a database sequence X, the subsequence matching problem is the problem of finding the subsequence $X^{i:j}$ of X that is the best match for the entire Q, i.e., that minimizes $D_{\text{full}}(Q, X^{i:j})$. In the next paragraphs we formally define what the best match is, and we specify how it can be computed.

3.1. Legal Warping Paths

A warping path $W = ((w_{1,1}, w_{1,2}), \ldots, (w_{|W|,1}, w_{|W|,2}))$ defines an alignment between two sequences Q and X. The i^{th} element of W is a pair $(w_{i,1}, w_{i,2})$ that specifies a correspondence between element $Q_{w_{i,1}}$ of Q and element $X_{w_{i,2}}$ of X. The cost $C(Q, X, W)$ of warping path W for Q and X is the L_p distance (for any choice of p) between vectors $(Q_{w_{1,1}}, \ldots, Q_{w_{|W|,1}})$ and $(X_{w_{1,2}}, \ldots, X_{w_{|W|,2}})$:

$$C(Q, X, W) = \sqrt[p]{\sum_{i=1}^{|W|} \|Q_{w_{i,1}} - X_{w_{i,2}}\|^p} \ . \qquad (1)$$

In the remainder of this section, to simplify the notation, we will assume that $p = 1$. However, the formulation we propose can be similarly applied to any choice of p.

For W to be a legal warping path, in the context of subsequence matching under DTW, W must satisfy the following constraints:

— **Boundary conditions:** $w_{1,1} = 1$ and $w_{|W|,1} = |Q|$. This requires the warping path to start by matching the first element of the query with some element of X, and end by matching the last element of the query with some element of X.

— **Monotonicity:** $w_{i+1,1} - w_{i,1} \geq 0, w_{i+1,2} - w_{i,2} \geq 0$. This forces the warping path indices $w_{i,1}$ and $w_{i,2}$ to increase monotonically with i.

— **Continuity:** $w_{i+1,1} - w_{i,1} \leq 1, w_{i+1,2} - w_{i,2} \leq 1$. This restricts the warping path indices $w_{i,1}$ and $w_{i,2}$ to never increase by more than 1, so that the warping path does not skip any elements of Q, and also does not skip any elements of X between positions $X_{w_{1,2}}$ and $X_{w_{|W|,2}}$.

— **(cDTW only) Diagonality:** $w_{|W|,2} - w_{1,2} = |Q| - 1, w_{i,2} - w_{1,2} \in [w_{i,1} - \Theta(Q, w_{i,1}), w_{i,1} + \Theta(Q, w_{i,1})]$, where $\Theta(Q, t)$ is some suitably chosen function (e.g., $\Theta(Q, t) = \rho|Q|$, for some constant ρ such that $\rho|Q|$ is relatively small compared to $|Q|$). The diagonality constraint makes the difference between (unconstrained) DTW and cDTW, and imposes that the subsequence $X^{w_{1,2}:w_{|W|,2}}$ be of the same length as Q.

3.2. Optimal Warping Paths and Distances

The optimal warping path $W^*(Q, X)$ between Q and X is the warping path that minimizes the cost $C(Q, X, W)$:

$$W^*(Q, X) = \text{argmin}_W C(Q, X, W). \tag{2}$$

We define the optimal subsequence match $M(Q, X)$ of Q in X to be the subsequence of X specified by the optimal warping path $W^*(Q, X)$. In other words, if $W^*(Q, X) = ((w^*_{1,1}, w^*_{1,2}), \ldots, (w^*_{m,1}, w^*_{m,2}))$, then $M(Q, X)$ is the subsequence $X^{w^*_{1,2}:w^*_{m,2}}$. We define the partial dynamic time warping (DTW) distance $D(Q, X)$ to be the cost of the optimal warping path between Q and X:

$$D(Q, X) = C(Q, X, W^*(Q, X)). \tag{3}$$

Clearly, partial DTW is an asymmetric distance measure.

To facilitate the description of our method, we will define two additional types of optimal warping paths and associated distance measures. First, we define $W^*_{\text{full}}(Q, X)$ to be the optimal *full warping path*, i.e., the path $W = ((w_{1,1}, w_{1,2}), \ldots, (w_{|W|,1}, w_{|W|,2}))$ minimizing $C(Q, X, W)$ under the additional boundary constraints that $w_{1,2} = 1$ and $w_{|W|,2} = |X|$. Then, we can define the full DTW distance measure $D_{\text{full}}(Q, X)$ as:

$$D_{\text{full}}(Q, X) = C(Q, X, W^*_{\text{full}}(Q, X)). \tag{4}$$

Distance $D_{\text{full}}(Q, X)$ measures the cost of full sequence matching, i.e., the cost of matching the entire Q with the entire X. In contrast, $D(Q, X)$ from Equation 3 corresponds to matching the entire Q with a *subsequence* of X.

We define $W^*(Q, X, j)$ to be the optimal warping path matching Q to a subsequence of X ending at X_j, i.e., the path $W = ((w_{1,1}, w_{1,2}), \ldots, (w_{|W|,1}, w_{|W|,2}))$ minimizing $C(Q, X, W)$ under the additional boundary constraint that $w_{|W|,2} = j$. Then, we can define $D_j(Q, X)$ as:

$$D_j(Q, X) = C(Q, X, W^*(Q, X, j)). \tag{5}$$

We define $M(Q, X, j)$ to be the optimal subsequence match for Q in X under the constraint that the last element of this match is X_j:

$$M(Q, X, j) = \text{argmin}_{X^{i:j}} D_{\text{full}}(Q, X^{i:j}). \tag{6}$$

Essentially, to identify $M(Q, X, j)$ we simply need to identify the start point i that minimizes the full distance D_{full} between R and $X^{i:j}$. For cDTW, the length of $M(Q, X, j)$ is constrained to be equal to the length of Q.

3.3. The DTW Algorithm

Dynamic time warping (DTW) is a term that refers both to the distance measures that we have just defined, and to the standard algorithm for computing these distance measure and the corresponding optimal warping paths. First, we describe the algorithm for computing the subsequence match under uncostrained DTW.

We define an operation $\oplus$ that takes as inputs a warping path $W = ((w_{1,1}, w_{1,2}), \ldots, (w_{|W|,1}, w_{|W|,2}))$ and a pair (w', w'') and returns a new warping path that is the result of appending (w', w'') to the end of W:

$$W \oplus (w', w'') = ((w_{1,1}, w_{1,2}), \ldots, (w_{|W|,1}, w_{|W|,2}), (w', w'')). \tag{7}$$

The DTW algorithm uses the following recursive definitions:

$$D_{0,0}(Q, X) = 0, D_{i,0}(Q, X) = \infty, D_{0,j}(Q, X) = 0 \tag{8}$$

$$W_{0,0}(Q, X) = (), W_{0,j}(Q, X) = () \tag{9}$$

$$\mathrm{A}(i, j) = \{(i, j-1), (i-1, j), (i-1, j-1)\} \tag{10}$$

$$(\mathrm{pi}(Q, X), \mathrm{pj}(Q, X)) = \mathrm{argmin}_{(s,t)\in \mathrm{A}(i,j)} D_{s,t}(Q, X) \tag{11}$$

$$D_{i,j}(Q, X) = \|Q_i - X_j\| + D_{\mathrm{pi}(Q,X),\mathrm{pj}(Q,X)}(Q, X) \tag{12}$$

$$W_{i,j}(Q, X) = W_{\mathrm{pi}(Q,X),\mathrm{pj}(Q,X)} \oplus (i, j) \tag{13}$$

$$D(Q, X) = \min_{j=1,\ldots,|X|} \{D_{|Q|,j}(Q, X)\} \tag{14}$$

The DTW algorithm proceeds by employing the above equations at each step, as follows:

- **Inputs.** A short sequence Q, and a long sequence X.
- **Initialization.** Compute $D_{0,0}(Q, X), D_{i,0}(Q, X), D_{0,j}(Q, X)$.
- **Main loop.** For $i = 1, \ldots, |Q|, j = 1, \ldots, |X|$:
  (1) Compute $(\mathrm{pi}(Q, X), \mathrm{pj}(Q, X))$.
  (2) Compute $D_{i,j}(Q, X)$.
  (3) Compute $W_{i,j}(Q, X)$.
- **Output.** Compute and return $D(Q, X)$.

The DTW algorithm takes time $O(|Q||X|)$. By defining $D_{0,j} = 0$ we essentially allow arbitrary prefixes of X to be skipped (i.e., matched with zero cost) before matching Q with the optimal subsequence in X [Sakurai et al. 2007]. By defining $D(Q, X)$ to be the minimum $D_{|Q|,j}(Q, X)$, where $j = 1, \ldots, |X|$, we allow the best matching subsequence of X to end at any position j. Overall, this definition matches the entire Q with an optimal subsequence of X.

For each position j of sequence X, the optimal warping path $W^*(Q, X, j)$ is computed as value $W_{|Q|,j}(Q, X)$ by the DTW algorithm (step 3 of the main loop) . The globally optimal warping path $W^*(Q, X)$ is simply $W^*(Q, X, j_{\text{opt}})$, where j_{opt} is the endpoint of the optimal match: $j_{\text{opt}} = \mathrm{argmin}_{j=1,\ldots,|X|}\{D_{|Q|,j}(Q, X)\}$.

3.4. The cDTW Algorithm

Constrained DTW (cDTW) is obtained from DTW simply by placing an additional constraint, which narrows down the set of positions in one sequence that can be matched with a specific position in the other sequence. Given a warping width w, this constraint is defined as follows:

$$D_{i,j}(Q,X) = \infty \text{ if } |i-j| > w \; . \tag{15}$$

The term "Sakoe-Chiba band" is often used to characterize the set of (i,j) positions for which $D_{i,j}$ is not infinite. Notice that if $w = 0$, cDTW becomes the L_p distance.

While a simple modification of DTW, cDTW has been shown to be significantly more efficient than DTW for full sequence matching [Keogh 2002], and to also produce more meaningful matching scores, as measured for example based on nearest neighbor classification accuracy [Ratanamahatana and Keogh 2005]. The constraints of cDTW can improve accuracy by eliminating from consideration pathological cases, i.e., accidental alignments that are legal (in the absence of constraints) and produce optimal scores, but do not capture a meaningful correspondence between the two time series.

Given the above definitions, the subsequence match of Q in a database X is the subsequence $X_{\mathrm{opt}} = (X_j, \ldots, X_{j+|Q|-1})$ that minimizes $D(Q, X_{\mathrm{opt}})$. Similarly to other approaches for subsequence matching under cDTW, namely LB_Keogh [Keogh 2002] and DTK [Han et al. 2007], we require that the subsequence match have the same length as the query. A simple approach for finding the subsequence match of Q is the sliding-window approach: we simply compute the matching cost between Q and every subsequence of X that has length $|Q|$.

The LB_Keogh [Keogh 2002] method speeds up the sliding window approach, often by orders of magnitude, by computing an efficient lower bound of the matching cost, that can be used to reject many subsequences without computing the exact cDTW cost between Q and those subsequences. With respect to LB_Keogh, which is an exact method, the method proposed in this paper can be seen as an approximate alternative for quickly rejecting many candidate subsequences; in our method, accuracy can be easily traded for efficiency, so as to achieve significantly larger speedups than LB_Keogh.

4. EBSM: AN EMBEDDING FOR SUBSEQUENCE MATCHING

Let $X = (X_1, \ldots, X_{|X|})$ be a database sequence that is relatively long, containing for example millions of elements. Without loss of generality, we can assume that the database only contains this one sequence X (if the database contains multiple sequences, we can concatenate them to generate a single sequence). Given a query sequence Q, we want to find the subsequence of X that optimally matches Q under DTW. We can do that using brute-force search, i.e., using the DTW algorithm described in the previous section. This paper describes a more efficient method. Our method is based on defining a novel type of embedding function F, which maps every query Q into a d-dimensional vector and every element X_j of the database sequence also into a d-dimensional vector. In this section we describe how to define such an embedding, and then we provide some examples and intuition as to why we expect such an embedding to be useful.

Let R be a sequence, of relatively short length, that we shall call a *reference object* or *reference sequence*. We will use R to create a 1D embedding F^R, mapping each query sequence into a real number $F(Q)$, and also mapping each step j of sequence X into a real number $F(X,j)$:

$$F^R(Q) \;=\; D_{|R|,|Q|}(R,Q) \; . \tag{16}$$

$$F^R(X,j) \;=\; D_{|R|,j}(R,X) \; . \tag{17}$$

Naturally, instead of picking a single reference sequence R, we can pick multiple reference sequences to create a multidimensional embedding. For example, let $R_1, \ldots, R_d$

be d reference sequences. Then, we can define a d-dimensional embedding F as follows:

$$F(Q) = (F^{R_1}(Q), \ldots, F^{R_d}(Q)) . \tag{18}$$

$$F(X,j) = (F^{R_1}(X,j), \ldots, F^{R_d}(X,j)) . \tag{19}$$

Computing the set of all embeddings $F(X,j)$, for $j = 1, \ldots, |X|$ is an off-line preprocessing step that takes time $O(|X| \sum_{i=1}^{d} |R_i|)$. In particular, computing the i^{th} dimension F^{R_i} can be done simultaneously for all positions (X,j), with a single application of the DTW algorithm with inputs R_i (as the short sequence) and X (as the long sequence). We note that the DTW algorithm computes each $F^{R_i}(X,j)$, for $j = 1, \ldots, |X|$, as value $D_{|R_i|,j}(R_i, X)$ (see Section 3.3 for more details).

Given a query Q, its embedding $F(Q)$ is computed online, by applying the DTW algorithm d times, with inputs R_i (in the role of the short sequence) and Q (in the role of the long sequence). In total, these applications of DTW take time $O(|Q| \sum_{i=1}^{d} |R_i|)$. This time is typically negligible compared to running the DTW algorithm between Q and X, which takes $O(|Q||X|)$ time. We assume that the sum of lengths of the reference objects is orders of magnitude smaller than the length $|X|$ of the database sequence.

Consequently, a very simple way to speed up brute force search for the best subsequence match of Q is to:

- Compare $F(Q)$ to $F(X,j)$ for $j = 1, \ldots, |X|$.
- Choose some j's such that $F(Q)$ is very similar to $F(X,j)$.
- For each such j, and for some length parameter L, run dynamic time warping between Q and $(X^{j-L+1:j})$ to compute the best subsequence match for Q in $(X^{j-L+1:j})$.

As long as we can choose a small number of such promising areas $(X^{j-L+1:j})$, evaluating only those areas will be much faster than running DTW between Q and X. Retrieving the most similar vectors $F(X,j)$ for $F(Q)$ can be done efficiently by applying a multidimensional vector indexing method to these embeddings [Gionis et al. 1999; Weber et al. 1998; Sakurai et al. 2000; Chakrabarti and Mehrotra 2000; Li et al. 2002; Egecioglu and Ferhatosmanoglu 2000; Kanth et al. 1998; Weber and Böhm 2000; Koudas et al. 2004; Tuncel et al. 2002; Tao et al. 2009].

We claim that, under certain circumstances, if Q is similar to a subsequence of X ending at X_j, and if R is some reference sequence, then $F^R(Q)$ is likely to be similar to $F^R(X,j)$. Here we provide some intuitive arguments for supporting this claim.

Let's consider a very simple case, illustrated in Figure 2. In this case, the query Q is *identical* to a subsequence $X^{i':j}$. Consider a reference sequence R, and suppose that $M(R,X,j)$ (defined as in Equation 6) is $X^{i:j}$, and that $i \geq i'$. In other words, $M(R,X,j)$ is a suffix of $X^{i':j}$ and thus a suffix of Q (since $X^{i':j} = Q$). Note that the following holds:

$$F^R(Q) = D_{|R|,|Q|}(R,Q) = D_{|R|,j}(R,X) = F^R(X,j). \tag{20}$$

In other words, if Q appears exactly as a subsequence $X^{i':j}$ of X, it holds that $F^R(Q) = F^R(X,j)$, *under the condition* that the optimal warping path aligning R with $X^{1:j}$ does not start before position i', which is where the appearance of Q starts.

This simple example illustrates an ideal case, where the query Q has an exact match $X^{i':j}$ in the database. The next case to consider is when $X^{i':j}$ is a slightly perturbed version of Q, obtained, for example, by adding noise from the interval $[-\epsilon, \epsilon]$ to each Q_t. In that case, assuming always that $M(R,X,j) = X^{i:j}$ and $i \geq i'$, we can show that $|F^R(Q) - F^R(X,j)| \leq (2|Q| - 1)\epsilon$. This is obtained by taking into account that warping path $W^*(R,X,j)$ cannot be longer than $2|Q| - 1$ (as long as $i \geq i'$).

There are two cases we have not covered:

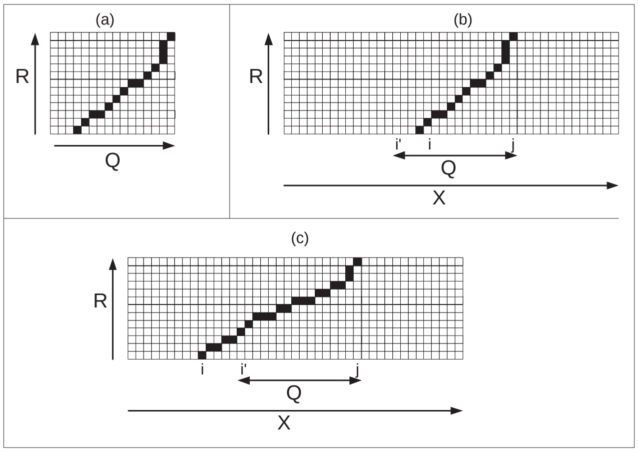

Fig. 2. (a) Example of an optimal warping path $W^*(R, Q, |Q|)$ aligning a reference object R to a suffix of Q. $F^R(Q)$ is the cost of $W^*(R, Q, |Q|)$. (b) Example of a warping path $W^*(R, X, j)$, aligning a reference object R to a subsequence $X^{i:j}$ of sequence X. $F^R(X, j)$ is the cost of $W^*(R, X, j)$. The query Q from (a) appears exactly in X, as subsequence $X^{i':j}$, and $i' < i$. Under these conditions, $F^R(Q) = F^R(X, j)$. (c) Similar to (b), except that $i' > i$. In this case, typically $F^R(Q) \neq F^R(X, j)$.

- Perturbations along the *temporal* axis, such as repetitions, insertions, or deletions. Unfortunately, for unconstrained DTW, due to the non-metric nature of the DTW distance measure, no existing approximation method can make any strong mathematical guarantees in the presence of such perturbations.
- The case where $i < i'$, i.e., the optimal path matching the reference sequence to a suffix of $X^{1:j}$ starts before the beginning of $M(Q, X, j)$. We address this issue in Section 7.

Given the lack of mathematical guarantees, in order for the proposed embeddings to be useful in practice, the following *statistical* property has to hold empirically: given position $j_{\text{opt}}(Q)$, such that the optimal subsequence match of Q in X ends at $j_{\text{opt}}(Q)$, and given some random position $j \neq j_{\text{opt}}(Q)$, it should be statistically very likely that $F(Q)$ is closer to $F(X, j_{\text{opt}}(Q))$ than to $F(X, j)$. If we have access to query samples during embedding construction, we can actually optimize embeddings so that $F(Q)$ is closer to $F(X, j_{\text{opt}}(Q))$ than to $F(X, j)$ as often as possible, over many random choices of Q and j. We do exactly that in Section 6.

5. FILTER-AND-REFINE RETRIEVAL

Our goal in this paper is to design a method for efficiently retrieving, given a query, its best matching subsequence from the database. In the previous sections we have defined embeddings that map each query object and each database position to a d-dimensional vector space. In this section we describe how to use such embeddings in an actual system.

5.1. General Framework

The retrieval framework that we use is filter-and-refine retrieval, where, given a query, the retrieval process consists of a filter step and a refine step [Hjaltason and Samet 2003a]. The filter step typically provides a quick way to identify a relatively small number of candidate matches. The refine step evaluates each of those candidates using the original matching algorithm (DTW in our case), in order to identify the candidate that best matches the query.

The goal in filter-and-refine retrieval is to improve retrieval efficiency with small, or zero loss in retrieval accuracy. Retrieval efficiency depends on the cost of the filter step (which is typically small) and the cost of evaluating candidates at the refine step. Evaluating a small number of candidates leads to significant savings compared to brute-force search (where brute-force search, in our setting, corresponds to running SPRING [Sakurai et al. 2007], i.e., running DTW between Q and X). Retrieval accuracy, given a query, depends on whether the best match is included among the candidates evaluated during the refine step. If the best match is among the candidates, the refine step will identify it and return the correct result.

Within this framework, embeddings can be used at the filter step, and provide a way to quickly select a relatively small number of candidates. Indeed, here lies the key contribution of our framework, in the fact that we provide a novel method for quick filtering, that can be applied in the context of subsequence matching. Our method relies on computationally cheap vector matching operations, as opposed to requiring computationally expensive applications of DTW. To be concrete, given a d-dimensional embedding F, defined as in the previous sections, F can be used in a filter-and-refine framework as follows:

Offline preprocessing step: Compute and store vector $F(X, j)$ for every position j of the database sequence X.

Online retrieval system: Given a previously unseen query object Q, we perform the following three steps:

- **Embedding step:** compute $F(Q)$, by measuring the distances between Q and the chosen reference sequences.
- **Filter step:** Select database positions (X, j) according to the distance between each $F(X, j)$ and $F(Q)$. These database positions are candidate *endpoints* of the best subsequence match for Q.
- **Refine step:** Evaluate selected candidate positions (X, j) by applying the DTW algorithm.

In the next subsections we specify the precise implementation of the filter step and the refine step.

5.2. The Filter Step

The simplest way to implement the filter step is by simply comparing $F(Q)$ to every single $F(X, j)$ stored in our database. The problem with doing that is that it may take too much time, especially with relatively high-dimensional embeddings (for example, 40-dimensional embeddings are used in our experiments). The cost of the filter step can be a significant part of the overall retrieval cost, as filtering involves comparisons between high-dimensional vectors. To improve the efficiency of the filter step, we propose three alternatives. The first one performs uniform sampling over the vector space. The second alternative uses a compressed version of the embedding space. The third alternative is piecewise Dynamic Time Warping (PDTW), a method described in [Keogh and Pazzani 2000], which can be easily integrated into our filter step.

5.2.1. Faster Filtering Using Sampling. In our implementation we use sampling, so as to avoid comparing $F(Q)$ to the embedding of every single database position. The way the embeddings are constructed, embeddings of nearby positions, such as $F_Q(X, j)$ and $F_Q(X, j + 1)$, tend to be very similar. A simple way to apply sampling is to choose a parameter δ, and sample uniformly one out of every δ vectors $F_Q(X, j)$. Given $F(Q)$, we only compare it with vectors $F_Q(X, 1), F_Q(X, 1 + \delta), F_Q(X, 1 + 2\delta), \ldots$. If, for a database position (X, j), its vector $F_Q(X, j)$ was not sampled, we simply assign to that position the distance between $F(Q)$ and the vector that was actually sampled among $\{F_Q(X, j - \lfloor\delta/2\rfloor), \ldots, F_Q(X, j + \lfloor\delta/2\rfloor)\}$.

5.2.2. Faster Filtering Using Segmentation. A significantly better speedup is achieved by compressing the vector space. Each $F_R(X, j)$, for $j = 1, \ldots, |X|$ can be considered as a time series. The values in each row change smoothly over time. Thus, we could use some common time series segmentation method that would identify non-overlapping segments with similar values. For our implementation we used the top-down time series segmentation algorithm described in [Keogh et al. 1993]. The segmentation error criterion is the one used by Bingham et al. [Bingham et al. 2006].

Given a time series sequence T and a maximum number of segments K, the main steps of the top-down segmentation algorithm are shown below:

— Iterate over positions $i = 2 : |T| - 1$ of time series T and find the position $i = k$ where some cost function is minimized.
— Report position $i = k$ as split point and run the algorithm recursively for sequences T[1:k] and T[k+1:—T—].
— Halt when the number of split points becomes K.

The cost function we used for the segmentation above is computed as follows: each segment is represented by the average value of the time series points that belong to the segment. The deviation of all points in the segments from the representative point defines the cost of the segmentation, i.e., the sum of the differences (L_1 norm) of each point in the segment from the representative point. The intuition behind this is that, at each iteration, we should choose the split point that causes the minimum deviation, i.e., the minimum sum.

For a given row in the embedding space, i.e., $F_{R_t}(X)$, with t being the t^{th} row, the final outcome of this process is a set of non-overlapping segments. Let $S_{R_t} = \{(f_{t_1}, b_{t_1}), \ldots, (f_{t_k}, b_{t_k})\}$ define a k-segmentation of R_t, with f_{t_i} being the **average value** of $F_{R_t}(X)$ in segment i and b_{t_i} the position in X where that segment ends. We also assume that segment (f_{t_1}, b_{t_1}) starts at position 1, segment (f_{t_k}, b_{t_k}) ends at position $|X|$ and $b_{t_{j+1}} = b_{t_j} + 1$, for $j = 2, \ldots, k-1$. The above segmentation is applied to each row in the embedding space, yielding a set of segmentations, $\mathcal{S} = \{S_{R_1}, \ldots, S_{R_{|R|}}\}$.

The above segmentation is used to speedup the filter step. At all times, a running vector V is stored and updated keeping the current pair-wise differences of each $F_Q(R_t)$ and each $F_{R_t(X,j)}$. $\mathcal{S}$ is scanned from position 1 to position $|\mathcal{S}|$, and whenever a segmentation border is detected, the corresponding dimension of V is updated accordingly. The sum of all values stored in V produce a score for each database position (X, j). Let V_j denote the j_{th} instance of V. Then, $V_1(t) = F_Q(R_t) - F_{R_t}(X, 1))^2$, for $t = 1, \ldots, |R|$ i.e., the first instance of V is initialized to the actual vector difference of the database embedding at position 1 and the query embedding. The, $\mathcal{S}$ is scanned progressively, and whenever a segment border is detected, V is updated accordingly. If, for a database position (X, j), the running vector has not been updated, we simply assign to that position the value of the previously updated instance of V.

5.2.3. Filter Step Using Piecewise Dynamic Time Warping. The filter step is further speeded up by adding one extra filtering operation before actually proceeding to refining with SPRING. In particular, we use piecewise Dynamic Time Warping (PDTW). PDTW was proposed in [Keogh and Pazzani 2000] as a standalone method for speeding up retrieval under DTW. At the same time, PDTW is orthogonal to our approach, and thus can be used to further prune our set of of candidate matches. In PDTW, query and database time series are approximated by shorter sequences, obtained by replacing each constant-length part of the original sequence with the average value over that part. The candidate matches obtained by the previously described filter steps can be evaluated using PDTW much faster than they would be evaluated under the standard DTW algorithm. Thus, we use PDTW to rank candidate matches and we finally pass the highest ranking candidates to the refine step for the final evaluation.

5.3. The Refine Step for Unconstrained DTW

The filter step ranks all database positions (X, j) in increasing order of the distance (or estimated distance, when we use approximations such as PCA, or sampling) between $F(X, j)$ and $F(Q)$. The task of the refine step is to evaluate the top p candidates, where p is a system parameter that provides a trade-off between retrieval accuracy and retrieval efficiency.

Algorithm 4.1 describes how this evaluation is performed. Since candidate positions (X, j) actually represent candidate *endpoints* of a subsequence match, we can evaluate each such candidate endpoint by starting the DTW algorithm from that endpoint and going backwards. In other words, the end of the query is aligned with the candidate endpoint, and DTW is used to find the optimal start (and corresponding matching cost) for that endpoint.

If we do not put any constraints, the DTW algorithm will go all the way back to the beginning of the database sequence. However, subsequences of X that are much longer than Q are very unlikely to be optimal matches for Q. In our experiments, 99.7% out of the 1000 queries used in performance evaluation have an optimal match no longer than twice the length of the query. Consequently, we consider that twice the length of the query is a pretty reasonable cut-off point, and we do not allow DTW to consider longer matches.

One complication is a case where, as the DTW algorithm moves backwards along the database sequence, the algorithm gets to another candidate endpoint that has not been evaluated yet. That endpoint will need to be evaluated at some point anyway, so we can save time by evaluating it now. In other words, while evaluating one endpoint, DTW can simultaneously evaluate all other endpoints that it finds along the way. The two adjustments that we make to allow for that are that:

- The "sink state" $Q_{|Q|+1}$ matches candidate endpoints (that have not already been checked) with cost 0 and all other database positions with cost ∞.
- If in the process of evaluating a candidate endpoint j we find another candidate endpoint j', we allow the DTW algorithm to look back further, up to position $j' - 2|Q| + 1$.

The $\mathrm{endpoint}$ array in Algorithm 4.1 keeps track, for every pair (i, j), of the endpoint that corresponds to the cost stored in $\mathrm{cost}[i][j]$. This is useful in the case where multiple candidate endpoints are encountered, so that when the optimal matching score is found (stored in variable $\mathrm{distance}$), we know what endpoint that matching score corresponds to.

The $\mathrm{columns}$ variable, which is an output of Algorithm 4.1, measures the number of database positions on which DTW is applied. These database positions include both each candidate endpoint and all other positions j for which $\mathrm{cost}[i][j]$ is computed. The $\mathrm{columns}$ output is a very good measure of how much time the refine step takes, compared to the time it would take for brute-force search, i.e., for applying the original DTW algorithm as described in Section 3. In the experiments, one of the main measures of EBSM efficiency (the DTW cell cost) is simply defined as the ratio between $\mathrm{columns}$ and the length $|X|$ of the database.

We note that each application of DTW in Algorithm 4.1 stops when the minimum $\mathrm{cost}[i][j]$ over all $i = 1, \ldots, |Q|$ is higher than the minimum distance found so far. We

```
input  : $Q$: query.
         $X$: database sequence.
         sorted: an array of candidate endpoints $j$, sorted in decreasing order of $j$.
         $p$: number of candidates to evaluate.
output: $j_{\text{start}}, j_{\text{end}}$: start and end point of estimated best subsequence match.
         distance: distance between query and estimated best subsequence match.
         columns: number of database positions evaluated by DTW.
for $i = 1$ to $|X|$ do
    unchecked[$i$] = 0;
end
for $i = 1$ to $p$ do
    unchecked[sorted[$i$]] = 1;
end
distance = $\infty$;columns = 0;
// main loop, check all candidates sorted[1], ..., sorted[p].
for $k = 1$ to $p$ do
    candidate = sorted[$k$];
    if (unchecked[candidate] == 0) then continue;
    $j$ = candidate + 1;
    for $i = |Q| + 1$ to $1$ do
        cost[$i$][$j$] = $\infty$;
    end
    while (true) do
        $j = j - 1$;
        if (candidate $- j \geq 2 * |Q|$) then break;
        if (unchecked[$j$] == 1) then
            unchecked[$j$] = 0; candidate = $j$;
            cost[$|Q| + 1$][$j$] = 0;endpoint[$|Q| + 1$][$j$] = $j$;
        else
            cost[$|Q| + 1$][$j$] = $\infty$; // $j$ is not a candidate endpoint.
        end
        for $i = |Q|$ to $1$ do
            previous = $\{(i+1, j), (i, j+1), (i+1, j+1)\}$;
            $(p_i, p_j) = \text{argmin}_{(a,b) \in \text{previous}} \text{cost}[a][b]$;
            cost[$i$][$j$] = $|Q_i - X_j| + \text{cost}[p_i][p_j]$; endpoint[$i$][$j$] = endpoint[$p_i$][$p_j$];
        end
        if (cost[1][j] < distance) then
            distance = cost[1][$j$];j$_{\text{start}}$ = $j$;$j_{\text{end}}$ = endpoint[1][$j$];
        end
        columns = columns + 1;
        if ($\min\{\text{cost[i][j]} | i = 1, \ldots, |Q|\} \geq$ distance) then break;
    end
end
//final alignment step
start = $j_{\text{end}} - 3|Q|$; end = $j_{\text{end}} + |Q|$;
Adjust $j_{\text{start}}$ and $j_{\text{end}}$ by running the DTW algorithm between $Q$ and $X^{\text{start:end}}$;
```

Algorithm 3.1. The refine step for unconstrained DTW.

do that because any $\text{cost}[i][j-1]$ will be at least as high as the minimum (over all i's) of $\text{cost}[i][j]$, except if $j-1$ is also a candidate endpoint (in which case, it will also be evaluated during the refine step).

```
input  : $X$: database sequence.
         $Q_S$: training query set.
         $d$: embedding dimensionality.
         $RSK$: initial set of $k$ reference subsequences.
output: $R$: set of $d$ reference subsequences.
// select d reference sequences with highest variance from RSK
$R = \{R_1, .., R_d \,|R_i \in RSK \quad with \quad maximum \quad variance\}$
CreateEmbedding($R, X$);
oldSEE = 0;
for $i = 1$ to $|Q_S|$ do
    oldSEE+ = $EE(Q_S[i])$;
end
j = 1;
repeat
    // consider replacing $R_j$ with another reference object
    $CandR = RSK - R$;
    for $i = 0$ to $|CandR|$ do
        CreateEmbedding($R - \{R_j\} + \{CandR[i]\}, X$);
        newSEE = 0;
        for $i = 1$ to $|Q_S|$ do
            newSEE+ = $EE(Q_S[i])$;
        end
        if (newSEE < oldSEE) then
            $R_j = CandR[i]$;
            oldSEE = newSEE;
        end
    end
    $j = (j \,\%\, d) + 1$;
until (true);
```

Algorithm 3.2. The training algorithm for selection of reference objects.

The refine step concludes with a final alignment/verification operation, that evaluates, using the original DTW algorithm, the area around the estimated optimal subsequence match. In particular, if j_{end} is the estimated endpoint of the optimal match, we run the DTW algorithm between Q and $X^{(j_{\text{end}}-3|Q|):(j_{\text{end}}+|Q|)}$. The purpose of this final alignment operation is to correctly handle cases where j_{start} and j_{end} are off by a small amount (a fraction of the size of Q) from the correct positions. This may arise when the optimal endpoint was not included in the original set of candidates obtained from the filter step, or when the length of the optimal match was longer than $2|Q|$.

6. SELECTION OF REFERENCE SEQUENCES

In this section, we present an approach for selecting reference objects in order to improve the quality of the embedding. The goal is to create an embedding where the rankings of different subsequences with respect to a query in the embedding space resemble the rankings of these subsequences in the original space. Our approach is largely an adaptation of the method proposed in [Venkateswaran et al. 2006].

The first step is based on the max variance heuristic, i.e., the idea that we should select subsequences that cover the domain space (as much as possible) and have distances to other subsequences with high variance. In particular, we select uniformly at random l subsequences with sizes between *(minimum query size)/2* and *maximum* *query size* from different locations in the database sequence. Then, we compute the DTW distances for each pair of them ($O(l^2)$ values) and we select the k subsequences with the highest variance in their distances to the other $l - 1$ subsequences. Thus we select an initial set of k reference objects.

The next step is to use a learning approach to select the final set of reference objects assuming that we have a set of samples that is representative of the query distribution. The input to this algorithm is a set of k reference objects RSK selected from the previous step, the number of final reference objects d (where $d < k$) and a set of sample queries $\mathbb{Q}_s$. The main idea is to select d out of the k reference objects so as to minimize the embedding error on the sample query set. The embedding error $\mathrm{EE}(Q)$ of a query Q is defined as the number of vectors $F(X, j)$ in the embedding space that the embedding of the query $F(Q)$ is closer to than it is to the embedding of $F(X, j_Q)$, where j_Q is the endpoint of the optimal subsequence match of Q in the database.

Initially, we select d initial reference objects $R_1, \dots, R_d$ and we create the embedding of the database and the query set $\mathbb{Q}_s$ using the selected R_i's. Then, for each query, we compute the embedding error and we compute the sum of these errors over all queries, i.e., $\mathrm{SEE} = \sum_{Q \in \mathbb{Q}_s} \mathrm{EE}(Q)$. The next step, is to consider a replacement of the i^{th} reference object with an object in $\mathrm{RSK}-\{R_1, \dots, R_d\}$, and re-estimate the SEE. If SEE is reduced, we make the replacement and we continue with the next $(i + 1)$-th reference object. This process starts from $i = 1$ until $i = d$. After we replace the d^{th} reference object we continue again with the first reference object. The loop continues until the improvement of the SEE over all reference objects falls below a threshold. The pseudocode of the algorithm is shown in Algorithm 4.2. To reduce the computation overhead of the technique we use a sample of the possible replacements in each step. Thus, instead of considering all objects in $\mathrm{RSK} - \{R_1, \dots, R_d\}$ for replacement, we consider only a sample of them. Furthermore, we use a sample of the database entries to estimate the SEE.

Note that the embedding optimization method described here largely follows the method described in [Venkateswaran et al. 2006]. However, the approach in [Venkateswaran et al. 2006] was based on the Edit distance, which is a metric, and therefore a different optimization criterion was used. In particular, in [Venkateswaran et al. 2006], reference objects are selected based on the pruning power of each reference object. Since DTW is not a metric, reference objects in our setting do not have pruning power, unless we allow some incorrect results. That is why we use the sum of errors as our optimization criterion.

7. HANDLING LARGE RANGES OF QUERY LENGTHS

In Section 4 and in Figure 2 we have illustrated that, intuitively, when the query Q has a very close match $X^{i:j}$ in the database, we expect $F^R(Q)$ and $F^R(X, j)$ to be similar, as long as $M(R, X, j)$ is a suffix of $M(Q, X, j)$. If we fix the length $|Q|$ of the query, as the length $|R|$ of the reference object increases, it becomes more and more likely that $M(R, X, j)$ will start before the beginning of $M(Q, X, j)$. In those cases, $F^R(Q)$ and $F^R(X, j)$ can be very different, even in the ideal case where Q is identical to $X^{i:j}$.

In our experiments, the minimum query length is 60 and the maximum query length is 637. Figure 3 shows a histogram of the lengths of the 40 reference objects that were chosen by the embedding optimization algorithm in our experiments. We note that smaller lengths have higher frequencies in that histogram. We interpret that as empirical evidence for the argument that long reference objects tend to be harmful when applied to short queries, and it is better to have short reference objects applied to long queries. Overall, as we shall see in the experiments section, this 40-dimensional embedding provides very good performance.

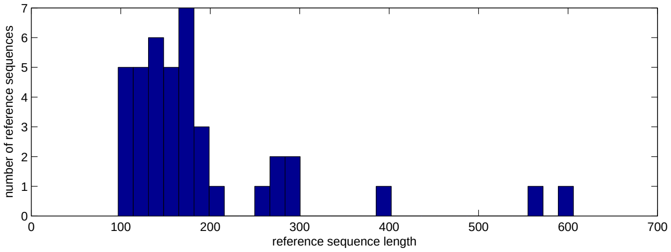

Fig. 3. Distribution of lengths of the 40 reference objects chosen by the embedding optimization algorithm in our experiments.

At the same time, in any situation where there is a large difference in scale between the shortest query length and the longest query length, we are presented with a dilemma. While long reference objects may hurt performance for short queries, using only short reference objects gives us very little information about the really long queries. To be exact, given a reference object R and a database position (X, j), $F^R(X, j)$ only gives us information about subsequence $M(R, X, j)$. If Q is a really long query and R is a really short reference object, proximity between $F(Q)$ and $F(X, j)$ cannot be interpreted as strong evidence of a good subsequence match for the entire Q ending at position j; it is simply strong evidence of a good subsequence match ending at position j for some small *suffix* of Q defined by $M(R, Q, |Q|)$.

The simple solution in such cases is to use, for each query, only embedding dimensions corresponding to a subset of the chosen reference objects. This subset of reference objects should have lengths that are not larger than the query length, and are not too much smaller than the query length either (e.g., no smaller than half the query length). To ensure that for any query length there is a sufficient number of reference objects, reference object lengths can be split into d ranges $[r, rs), [rs, rs^2), [rs^2, rs^3), \dots [rs^{d-1}, rs^d)$, where r is the minimum desired reference object length, rs^d is the highest desired reference object length, and s is determined given r, d and rs^d. Then, we can constrain the d-dimensional embedding so that for each range $[rs^i, rs^{i+1})$ there is only one reference object with length in that range.

We do not use this approach in our experiments, because the simple scheme of using all reference objects for all queries works well enough. However, it is important to have in mind the limitations of this simple scheme, and we believe that the remedy we have outlined here is a good starting point for addressing these limitations.

8. BIDIRECTIONAL SUBSEQUENCE EMBEDDINGS

In this section we introduce Bidirectional Subsequence Embeddings (BSE), a new embedding-based method for subsequence matching under cDTW, i.e., the variation of DTW that includes the diagonality constraint.

The motivation for BSE stems from an important difference between unconstrained DTW and cDTW. In particular, cDTW imposes the constraint that the length of the subsequence match should be equal to the length of the query sequence. If we want to identify subsequence matches with different lengths than that of the query, we have to perform repeated searches, and resize the query or the database accordingly for each search. Essentially, searching for the best subsequence match under cDTW can be decomposed into multiple independent searches, each of which identifies the best subsequence match of a specific length [Keogh 2002; Han et al. 2007]. In order to speed up subsequence matching under cDTW, we need to speed up each of these independent searches.

In each of the multiple independent searches that we need to perform, the length of the subsequence match that we are looking for is known. This additional information, i.e., the length of the match, is not used by EBSM. BSE is a modification of EBSM that exploits the knowledge of the length of the match, so as to achieve additional gains in performance.

Our starting point for BSE is similar to that of EBSM: we use reference sequences to define 1D embeddings. Given a reference sequence R, and given the definition in Section 3.4 of the matching cost D for cDTW, we define a 1D embedding H^R as follows:

$$H^R(Q) = \begin{cases} D(R, (Q_{|Q|-|R|+1}, \ldots, Q_{|Q|})) & \text{if } |R| \leq |Q| \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

$$H_Q^R(X, j) = \begin{cases} D(R, (X_{j-|R|+1}, \ldots, X_j)) & \text{if } |R| \leq |Q| \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

If we compare the above two equations with the corresponding equations 16 and 17 for EBSM, we notice two important differences: first, if the reference sequence is longer than the query, then the query is mapped to zero. Second, the embedding $H_Q^R(X, j)$ depends not only on X and j, but also on the length of the query: $H_Q^R(X, j) = 0$ when the reference sequence R is longer than Q. These changes have a simple interpretation: they effectively force us to ignore, given a query Q, any reference sequence longer than Q. This is motivated by the fact that, when we impose the diagonality constraint and R is longer than Q, there is no legal warping path matching R with any suffix of Q.

If $R_1, \ldots, R_d$ are d reference sequences, then a d-dimensional embedding H is defined as follows:

$$H(Q) = (H^{R_1}(Q), \ldots, H^{R_d}(Q)) \ . \tag{23}$$

$$H_Q(X, j) = (H_Q^{R_1}(X, j), \ldots, H_Q^{R_d}(X, j)) \ . \tag{24}$$

Again, we note that the embedding of the database position (X, j) also depends on the length of the query.

If Q is exactly identical to a database subsequence ending at position (X, j^*), then $H(Q) = H_Q(X, j^*)$. If we perturb that subsequence match $(X_{j^*-|Q|+1}, \ldots, X_{j^*})$ so that it is not identical to Q anymore, we expect that small perturbations will lead to small changes in $H_Q(X, j^*)$, so that $H(Q)$ will still be fairly similar to $H_Q(X, j^*)$. Therefore, embeddings H are useful for identifying candidate endpoints of subsequence matches. Because of that, we refer to embeddings H as *endpoint embeddings*. These endpoint embeddings, and the motivation behind them, are just a straightforward adaptation of EBSM from unconstrained DTW to cDTW.

We can easily adapt the definition of endpoint embeddings H to also define startpoint embeddings G, that can be used to identify candidate start points of subsequence matches. We define 1D startpoint embeddings G^R and multidimensional startpoint embeddings G as follows:

$$G^R(Q) = \begin{cases} D(R, (Q_1, \ldots, Q_{|R|})) & \text{if } |R| \leq |Q| \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

$$G_Q^R(X, j) = \begin{cases} D(R, (X_j, \ldots, X_{j+|R|-1})) & \text{if } |R| \leq |Q| \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

$$G(Q) = (G^{R_1}(Q), \ldots, G^{R_d}(Q)) \ . \tag{27}$$

$$G_Q(X, j) = (G_Q^{R_1}(X, j), \ldots, G_Q^{R_d}(X, j)) \ . \tag{28}$$

We note that startpoint embeddings could also, in principle, also be defined for unconstrained DTW. As a matter of fact, EBSM could be defined using startpoint embeddings instead of endpoint embeddings, and there is no fundamental reason for either of these two alternatives to perform better or worse than the other one. A natural question is whether by combining startpoint embeddings with endpoint embeddings we can get better performance than by using each of them in isolation. For unconstrained DTW, we have still not obtained a clear answer to that question. However, for cDTW, there is a clear affirmative answer, and BSE is the method that we have formulated for combining startpoint embeddings and endpoint embeddings.

The reason that, under cDTW, it is easy to combine startpoint embeddings and endpoint embeddings, is that, in cDTW we know the length of the subsequence match we are looking for. Given a query Q, if the best subsequence match ends at position (X, j), then that match has to be of length $|Q|$, and consequently that match has to start at position $(X, j - |Q| + 1)$. Consequently, if the best subsequence match ends at position (X, j), we expect the startpoint embedding $G(Q)$ to be similar to the startpoint embedding $G_Q(X, j-|Q|+1)$ of the first position of the match, and we also expect the endpoint embedding $H(Q)$ to be similar to the endpoint embedding $H_Q(X, j)$ of the final position of the match.

Based on the above observation, instead of identifying promising candidate matches using only endpoint embeddings, as EBSM does, we can use both types of embeddings together: to quickly evaluate subsequence $(X_{j-|Q|+1|}, \ldots, X_j)$ as a possible match for Q, we should compare $G(Q)$ with $G(X, j - |Q| + 1)$, and $H(Q)$ with $H(X, j)$.

To capture the correspondence, given Q, between startpoint embedding $G_Q(X, j - |Q| + 1)$ and endpoint embedding $H_Q(X, j)$, we define a unified embedding F, which we call a *bidirectional subsequence embedding* (BSE), that combines startpoint and endpoint embeddings. The BSE embedding F is simply a concatenation of the startpoint and endpoint embeddings:

$$F(Q) = (G(Q), H(Q)) \,. \tag{29}$$

$$F_Q(X, j) = (G_Q(X, j - |Q| + 1), H_Q(X, j)) \,. \tag{30}$$

Figure 4 illustrates the construction of a BSE embedding given a query Q and a reference object R.

To summarize, the key difference between EBSM and BSE is that EBSM uses only the equivalent of endpoint embeddings, whereas BSE combines information from startpoint embeddings and endpoint embeddings. The question of how to combine startpoint embeddings and endpoint embeddings in unconstrained DTW is nontrivial. On the other hand, using the constraints available in cDTW we can easily combine startpoint and endpoint embeddings, online, based on the length of the query. As we shall see in the experiments, this combination leads to improved performance over using only endpoint embeddings.

8.1. Computing Bidirectional Database Embeddings

Suppose that we have chosen d reference sequences $R_1, \ldots, R_d$. We note that applying Equations 26 and 22 to compute embeddings $G_Q(X, j)$ and $H_Q(X, j)$ requires knowing the query, or, at least, the length of the query. At the same time, computing $G_Q(X, j)$ and $H_Q(X, j)$ online, given a query, is too expensive (actually, even more expensive than using brute force to find the subsequence match of the query), unless we can make use of some precomputed information.

This precomputed information is in the form of query-independent embeddings $G(X, j)$ and $H(X, j)$, that are simply defined by dropping the dependency on the query

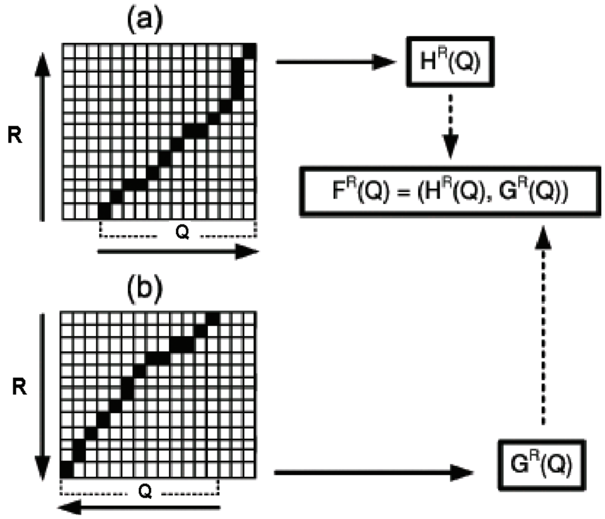

Fig. 4. An example that illustrates the construction of the bidirectional embedding given a query Q and a reference object R.

length. Given reference sequences $R_1, \ldots, R^d$, we define:

$$G^R(X,j) \;=\; D(R, (X_j, \ldots, X_{j+|R|-1})) \tag{31}$$

$$G(X,j) \;=\; (G^{R_1}(X,j), \ldots, G^{R_d}(X,j)) \tag{32}$$

$$H^R(X,j) \;=\; D(R, (X_{j-|R|+1}, \ldots, X_j)) \tag{33}$$

$$H(X,j) \;=\; (H^{R_1}(X,j), \ldots, H^{R_d}(X,j)) \tag{34}$$

Embeddings $G(X,j)$ and $H(X,j)$ do not depend on the query, and so they can be precomputed off-line and stored. Given a query Q, $G_Q(X,j)$ and $H_Q(X,j)$ can be obtained by putting a 0 to all embedding dimensions corresponding to reference sequences longer than Q. Even more simply, those embedding dimensions can be ignored when computing Euclidean distances.

It is also important to note that G^R and H^R are related as follows:

$$G^R(X,j) = H^R(X, j + |R| - 1)\ . \tag{35}$$

This means that, in practice, only startpoint embeddings $G(X,j)$ need to be precomputed and stored. Embeddings $H(X,j)$, and the query-sensitive embeddings $F_Q(X,j)$, can be easily obtained, online, from the precomputed embeddings $G(X,j)$. As we will see in the experiments, the total retrieval time, that includes these online computations, is still much faster than the retrieval time obtained using brute force or alternative exact methods such as LB_Keogh [Keogh 2002] and DTK [Han et al. 2007].

8.2. Using BSE for Filter-and-Refine Retrieval

The retrieval framework that we use is filter-and-refine retrieval and it is very similar to the one used by EBSM. In particular, given embeddings G, H, and $F = (G, H)$, defined as in the previous sections, F can be used in a filter-and-refine framework as follows:

Offline preprocessing step: Compute and store vector $G(X, j)$ for every position j of the database sequence X. Computing embeddings $G(X, j)$, for $j = 1, \ldots, |X|$, is an off-line preprocessing step that takes time $O(|X| \sum_{i=1}^{d} |R_i|^2)$.

Online retrieval system: Given a previously unseen query object Q, we perform the following three steps:

- **Embedding step:** compute $G(Q)$ and $H(Q)$, by measuring the cDTW matching cost between Q and the reference sequences. Concatenate $G(Q)$ and $H(Q)$ to form vector $F(Q)$. Also, given Q and the precomputed $G(X, j)$, form vectors $G_Q(X, j), H_Q(X, j)$, and $F_Q(X, j)$.
- **Filter step:** For some user-defined parameter p, select p database positions (X, j) according to the Euclidean distance between each $F_Q(X, j)$ and $F(Q)$. These database positions define candidate subsequence matches $(X_{j-|Q|+1}, \ldots, X_j)$ for Q.
- **Refine step:** Evaluate the selected candidate subsequence matches. Evaluation proceeds by first applying LB_Keogh [Keogh 2002] to establish a lower bound of the matching cost, and then evaluating the exact cDTW matching cost for enough candidates to assure that the best matching candidate has been found, as described in [Keogh 2002].

We note that the refine step, instead of simply measuring the cDTW matching cost between the query and all candidate subsequence matches, uses LB_Keogh to speed up computations. LB_Keogh is an exact method, so it guarantees that, if the correct subsequence match has been included in the candidates, the refine step will identify that match. At the same time, the correct subsequence match will not be retrieved unless it has been identified as a candidate during the filter step. Consequently, similar to EBSM, BSE is an approximate method, and it is possible that, for some queries, the correct subsequence match will be rejected during the filter step. The experiments demonstrate that BSE leads to good tradeoffs between accuracy and efficiency.

8.3. Embedding Optimization and Speedup

In our experiments, the max variance heuristic described earlier is sufficient for constructing embeddings that give state-of-the-art results. The learning method improves performance even further. The filter and refine steps are speeded up accordingly using the segmentation technique and the additional refinement step described for EBSM.

9. EXPERIMENTS

The proposed methods are evaluated on 20 time series datasets obtained from the UCR Time Series Data Mining Archive [Keogh 2006]. We note that the paper describing the original version of EBSM [Athitsos et al. 2008] only included experiments on three of those 20 datasets (50Words, Wafer, and Yoga), and thus the experimental evaluation of EBSM in this paper is significantly more comprehensive than in [Athitsos et al. 2008]. EBSM is compared to the two state-of-the-art methods for subsequence matching under unconstrained DTW:

- **SPRING:** the exact method proposed by Sakurai et al. [Sakurai et al. 2007], which applies the DTW algorithm as described in Section 3.3.
- **Modified PDTW:** a modification of the approximate method based on piecewise aggregate approximation that was proposed by Keogh et al. [Keogh and Pazzani 2000].

Actually, as formulated in [Keogh and Pazzani 2000], PDTW (given a sampling rate) yields a specific accuracy and efficiency, by applying DTW to smaller, subsampled versions of query Q and database sequence X. Even with the smallest possible sampling rate of 2, for which the original PDTW cost is 25% of the cost of brute-force search,

Table I. Dataset Description

Name	Length of each time series	Size of "training set" (queries)	Number of time series used for embedding optimization	Number of time series used for measuring performance	Size of "test set" (database)
50Words	270	450	192	258	455
Adiac	176	390	166	224	391
Beef	470	30	13	17	30
CBF	128	30	13	17	900
Coffee	286	28	12	16	28
ECG	96	100	43	57	100
FaceAll	131	560	239	321	1690
FaceFour	350	24	10	14	88
Fish	463	175	75	100	175
Gun-Point	150	50	21	29	150
Lightning-2	637	60	26	34	61
Lightning-7	319	70	30	40	73
OliveOil	570	30	13	17	30
OSU Leaf	427	200	85	115	242
Swedish Leaf	128	500	213	287	625
Synthetic Control	60	300	128	172	300
Trace	100	100	43	57	275
Two Patterns	128	1000	427	573	4000
Wafer	152	1000	428	572	6174
Yoga	426	300	130	170	3000

Description of the twenty UCR datasets we combined to generate our dataset. For each original UCR dataset we show the sizes of the original training and test sets. We note that, in our experiments, we use the original training sets to obtain queries for embedding optimization and for performance evaluation, and we use the original test sets to generate the long database sequence (of length 3,729,295)

the original PDTW method has an accuracy rate of less than 50%. We modify the original PDTW so as to significantly improve those results, as follows: in our modified PDTW, the original PDTW of [Keogh and Pazzani 2000] is used as a filter step, that quickly identifies candidate endpoint positions, exactly as the proposed embeddings do for EBSM. We then apply the refine step on top of the original PDTW rankings, using the exact same algorithm (Algorithm 4.1) for the refine step that we use in EBSM. We will see in the results that the modified PDTW works very well, but still not as well as EBSM.

BSE is evaluated on the same UCR time series data used for EBSM and also on an additional random walk synthetic dataset. BSE is compared to two state-of-the-art methods for subsequence matching under constrained DTW:

- **LB_Keogh with sliding window:** Given a query of length $|Q|$, a sliding window of size $|Q|$ scans the time series database, performing the LB_Keogh lower bounding technique at each step.
- **DTK:** the exact subsequence matching method proposed in [Han et al. 2007]. We note that this method has been designed to work for external memory, but here we evaluate it on main memory datasets. Therefore, first we buffer the complete index in main memory and then we run the queries. Thus, all the operations are executed in main memory.

9.1. Datasets

To create a large and diverse enough dataset, we combined twenty of the datasets from UCR Time Series Data Mining Archive [Keogh 2006]. The UCR datasets that we used are shown on Table I.

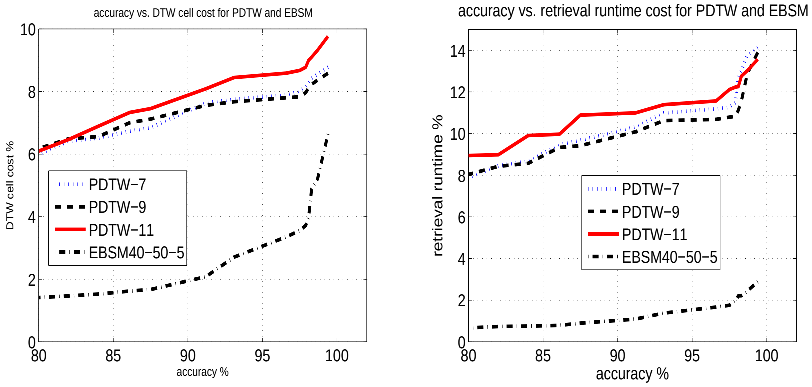

Fig. 5. Comparing the accuracy versus efficiency trade-offs achieved by EBSM with segmentation rate 50 and applying PDTW at the refine step, and by the modified version of PDTW with sampling rates 7, 9, 11, and 13. The left figure measures efficiency using the DTW cell cost, and the right figure measures efficiency using the retrieval runtime cost. The costs shown are average costs over our test set of 2897 queries. Note that SPRING, being an exact method, corresponds to a single point (not shown on these figures), with perfect accuracy 100%, maximal DTW cell cost 100%, and maximum retrieval runtime cost 100%.

Each of the twenty UCR datasets contains a test set and a training set. As can be seen on Table I, for all datasets, the original split into training and test sets created test sets that were greater than or equal in size to the corresponding training sets. In order to evaluate indexing performance, we wanted to create a sufficiently large database. Thus, we switched these “training set” and “test set” designations in our experiments. More specifically, our database is a single time series X, that was generated by concatenating all time series in the original test sets. The length $|X|$ of the database is obviously the sum of lengths of all these time series, which adds up to 3,729,295.

Our set of queries was the set of time series in the original training sets of the twenty UCR datasets. In total, this set includes 5397 time series. We randomly chose 2500 of those time series as a validation set of queries, that was used for embedding optimization using Algorithm 4.2. The remaining 2897 queries were used to evaluate indexing performance. Naturally, the set of 2897 queries used for performance evaluation was completely disjoint from the set of queries used during embedding optimization.

9.2. Performance Measures

Our methods are approximate, meaning that they do not guarantee finding the optimal subsequence match for each query. The two key measures of performance in this context are accuracy and efficiency. Accuracy is simply the percentage of queries in our evaluation set for which the optimal subsequence match was successfully retrieved. Efficiency can be evaluated using two measures: DTW cell cost, which is independent of the hardware and software used in the experiments, and retrieval runtime, which, although dependent on hardware and software, gives a better picture of what results to expect in a real system. Here we define the two measures of efficiency in more detail:

— **DTW cell cost:** For each query Q, the DTW cell cost is the ratio of number of cells $[i][j]$ visited by Algorithm 4.1 over number of cells $[i][j]$ using the SPRING method (for the SPRING method, this number is the product of query length and database length). For PDTW with sampling rate s, we add $\frac{1}{s^2}$ to this ratio, to reflect the cost of running the DTW algorithm between the subsampled query and the subsampled database. For the entire test set of 2897 queries, we report the average DTW cell cost over all queries.

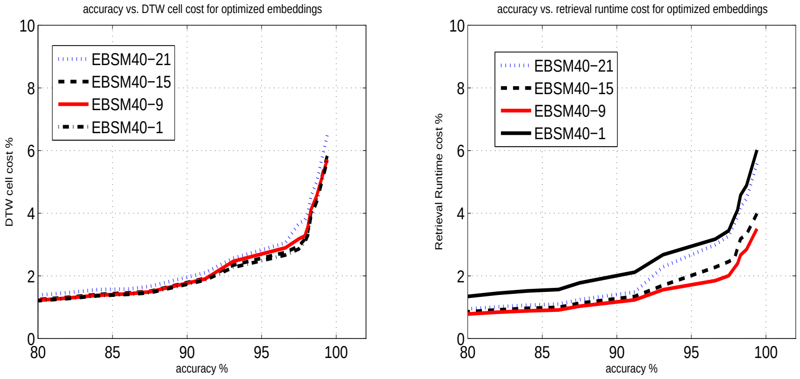

Fig. 6. Accuracy vs. efficiency for EBSM with sampling rates 1, 9, 15, and 21. The **left** figure measures efficiency using the DTW cell cost, and the **right** figure measures efficiency using the retrieval runtime cost. The costs shown are average costs over our test set of 2897 queries.

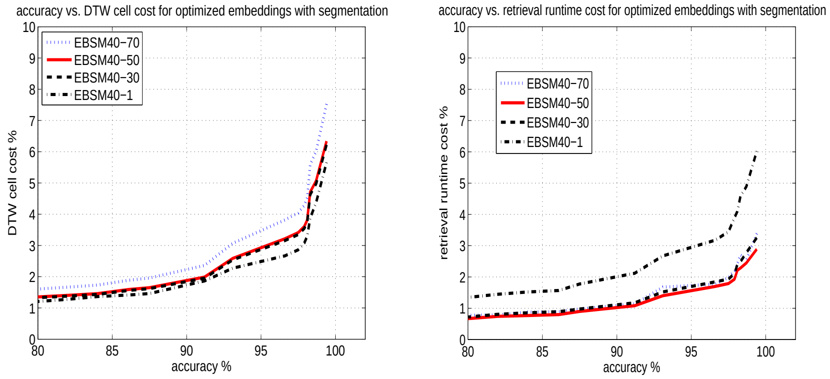

Fig. 7. Accuracy vs. efficiency for EBSM with segmentation rates 1, 30, 50, and 70. The left figure measures efficiency using the DTW cell cost, and the right figure measures efficiency using the retrieval runtime cost. The costs shown are average costs over our test set of 2897 queries.

— **Retrieval runtime cost:** For each query Q, given an indexing method, the retrieval runtime cost is the ratio of total retrieval time for that query using that indexing method over the total retrieval time attained for that query using brute force search. For the entire test set, we report the average retrieval runtime cost over all 2897 queries. While runtime is harder to analyze, as it depends on diverse things such as cache size, memory bus bandwidth, etc., runtime is also a more fair measure for comparing our methods to competitors, as it includes the costs of both the filter step and the refine step. The DTW cell cost ignores the cost of the filter step for our methods.

We remind the reader that the SPRING method simply uses the standard DTW algorithm of Section 3.3, and thus, for unconstrained DTW, SPRING is equivalent to brute-force search. Consequently, by definition, the DTW cell cost of SPRING is always 1, and the retrieval runtime cost of SPRING is always 1.

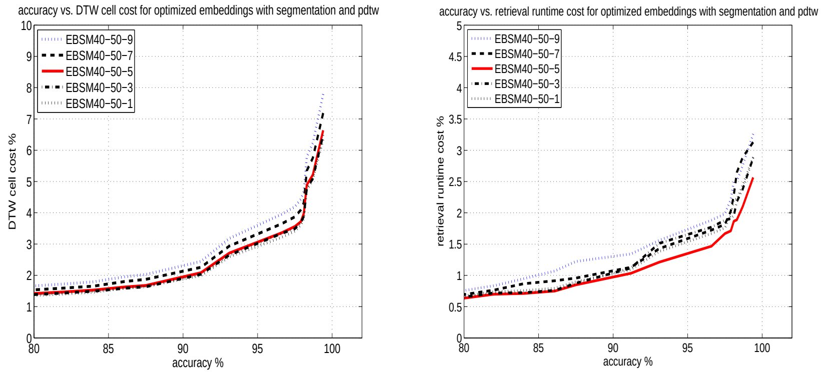

Fig. 8. Accuracy vs. efficiency for EBSM with segmentation rate 50 and averaging factors for pdtw 1, 3, 5, and 7. The left figure measures efficiency using the DTW cell cost, and the right figure measures efficiency using the retrieval runtime cost. The costs shown are average costs over our test set of 2897 queries.

The system was implemented in C++, and run on an AMD Opteron 8220 SE processor running at 2.8GHz. LB_Keogh was also implemented in C++. The code for DTK has been obtained from the authors [Han et al. 2007].

9.3. Experimental Evaluation of EBSM

Trade-offs between accuracy and efficiency can be obtained very easily, for both EBSM and the modified PDTW, by changing parameter p of the refine step (see Algorithm 4.1). Increasing the value of p increases accuracy, but decreases efficiency, by increasing both the DTW cell cost and the running time.

We should emphasize the runtime retrieval cost depends on the retrieval method, the data set, the implementation, and the system platform. On the other hand, the DTW cell cost only depends on the retrieval method and the data set; different implementations of the same method should produce the same results (or very similar, when random choices are involved) on the same data set regardless of the system platform or any implementation details.

We compare EBSM to modified PDTW and SPRING. We note that the SPRING method guarantees finding the optimal subsequence match, whereas modified PDTW (like EBSM) is an approximate method. For EBSM, unless otherwise indicated, we used a 40-dimensional embedding, with a sampling rate of 9. For the embedding optimization procedure of Section 6, we used parameters $l = 5000$ (l was the number of candidate reference objects before selection using the maximum variance criterion) and $k = 1000$ (k was the number of candidate reference objects selected based on the maximum variance criterion).

Figure 5 shows the trade-offs of accuracy versus efficiency achieved. We note that EBSM provides very good trade-offs between accuracy and retrieval cost. Also, EBSM significantly outperforms the modified PDTW, in terms of both DTW cell cost and retrieval runtime cost. For accuracy settings between 80% and 99.5%, EBSM attains costs smaller by a factor of 5 or more compared to PDTW.

We should note that the DTK method for efficient subsequence matching [Han et al. 2007] is only applicable for constrained DTW. Thus, it would not be meaningful to compare EBSM versus DTK. We do compare to DTK in our experimental evaluation of BSE, which is also applicable for constrained DTW.

Figure 6 shows how the performance of EBSM varies with different sampling rates. For all results in that figure, 40-dimensional embeddings were used, optimized using

Algorithm 3.2. Sampling rates between 1 and 15 all produced pretty similar DTW cell costs for EBSM, but a sampling rate of 21 produced noticeably worse DTW cell costs. In terms of retrieval runtime, a sampling rate of 1 performed much worse compared to sampling rates of 9 and 15, because the cost of the filter step is much higher for sampling rate 1: the number of vector comparisons is equal to the length of the database divided by the sampling rate. Notice that while the sampling rate increases further (e.g., over 15) the retrieval runtime increases since, despite the low cost of the filter step, the refine step requires more computations to achieve higher accuracy.

Figure 7 shows the improvement in both accuracy and retrieval runtime when the embedding segmentation technique is used (as described in Section 5.2.2), whereas Figure 8 shows the additional filter step using PDTW is used (Section 5.2.3). For the embedding segmentation technique we varied the segmentation rate within 1, 30, 50, and 70. Note that a segmentation rate of r means that for the embedding segmentation process we used $\frac{|X|}{r}$ segments, where X is the time series database. Notice that as the segmentation rate increases the cell cost increases as well since more database positions need to be examined to achieve high retrieval accuracy. The best trade-off between accuracy and retrieval runtime was achieved at a segmentation rate of 50 (Figure 7(right). Finally, the segmentation rate was fixed to 50 and an experiment was performed that included the additional PDTW filter step varying the PDTW averaging factor k from 1 to 9. The best trade-off between accuracy and retrieval runtime in this case was achieved for $k = 5$ (Figure 8). Figure 9 shows the total improvement (by approximately a factor of 2) of EBSM using segmentation and PDTW at the filter step with segmentation rate 50 and averaging factor 5 compared to EBSM using sampling with rate 9.

Figure 10 shows how the performance of EBSM varies with different embedding dimensionality, for optimized (using Algorithm 3.2) and unoptimized embeddings. For all results in that figure, a sampling rate of 9 was used. For optimized embeddings, in terms of DTW cell cost, performance clearly improves with increased dimensionality up to about 40 dimensions, and does not change much between 40 and 160. Actually, 160 dimensions give a somewhat worse DTW cell cost compared to 40 dimensions, providing evidence that our embedding optimization method suffers from a mild effect of overfitting as the number of dimensions increases. When reference objects are selected randomly, overfitting is not an issue. As we see in Figure 10, a 160-dimensional unoptimized embedding yields a significantly lower DTW cell cost than lower-dimensional unoptimized embeddings.

Finally, in Tables IV and V we can see a summary of the improvements in performance of EBSM using (1) embedding optimization, (2) segmentation, and (3) PDTW for filtering vs. EBSM without embedding optimization (using only the first two lines of Algorithm 3.2), and PDTW. The first table shows the trade-offs between accuracy and cell cost, whereas the second table shows the trade-offs between accuracy and retrieval runtime. It can be seen that EBSM can achieve speedups of over an order of magnitude compared to SPRING and retrieval runtime of a factor of 10 faster than PDTW for 95% accuracy. Using the unoptimized version of EBSM (second column of both tables) achieves decent trade-offs compared to SPRING and PDTW, while being inferior to the optimized EBSM by approximately a factor of 2 for 95% accuracy.

In terms of offline preprocessing costs, selecting 40 reference sequences using Algorithm 3.2 took about 3 hours, and computing the 40-dimensional embedding of the database took about 240 seconds.

We also performed an experiment that demonstrates the scalability of EBSM with respect to database size. For this experiment, the queries were taken from the training set of the Wafer dataset (1000 queries) and we varied the database size as follows:

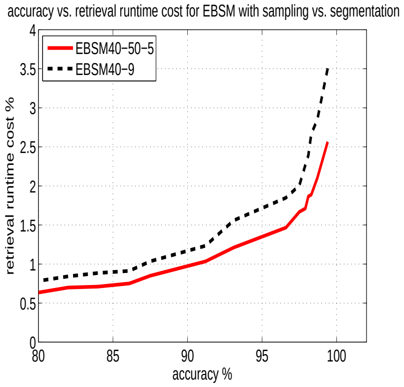

Fig. 9. Accuracy vs. retrieval runtime cost for EBSM with sampling vs. EBSM with segmentation and PDTW.

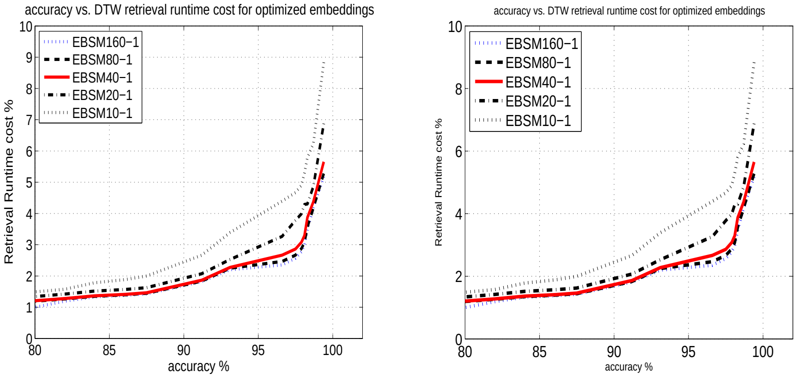

Fig. 10. Accuracy vs. efficiency for EBSM, using embeddings with different dimensionality. The plots show results for embeddings optimized using Algorithm 3.2.

Table II. EBSM vs. PDTW: DTW cell cost

Accuracy	EBSM40-50-5	EE40-50-5	PDTW-11	PDTW-9	PDTW-7
99%	6.63%	11.55%	8.75%	8.58 %	9.73%
98%	3.97%	8.86%	8.28%	8.11%	8.99%
95%	3.12%	6.01%	7.79%	7.76%	8.51%
90%	1.89%	3.99%	7.60%	7.51%	7.89%
85%	1.55%	3.21%	6.71%	6.77%	7.10%
80%	1.42%	3.02%	6.55%	6.11%	6.07%

Comparison of Cell Cost for EBSM with segmentation rate 50 and applying PDTW at the refine step vs. (1) EBSM with max variance and without embedding optimization and (2) the modified version of PDTW for averaging factors of 11, 9, and 7.

$|X| = 938448$ (using only the test set sequences of the Wafer dataset, test set 19, which corresponds to roughly 25% of the original database size), $|X| = 1787038$ (using test sets 4, 7, 18, and 19, which corresponds to roughly 50% of the original database size), $|X| = 2452815$ (using test sets $1-19$, which corresponds to roughly 75% of the original database size), and finally using the whole original database. For each experiment, the reference sequences were randomly selected from the corresponding database and

Table III. EBSM vs. PDTW: retrieval runtime

Accuracy	EBSM40-50-5	EE40-50-5	PDTW-11	PDTW-9	PDTW-7
99%	2.88%	4.92%	14.11%	12.89 %	13.56%
98%	2.21%	4.55%	12.81%	11.12%	12.25%
95%	1.55%	3.76%	10.99%	10.65%	11,55%
90%	1.09%	2.33%	10.32%	10.04%	10.99%
85%	0.77%	1.52%	9.55%	9.21%	9.85%
80%	0.66%	1.33%	7.79%	7.96%	8.93%

Comparison of Retrieval Runtime for EBSM with segmentation rate 50 and applying PDTW at the refine step vs. (1) EBSM with max variance and without embedding optimization and (2) the modified version of PDTW for averaging factors of 11, 9, and 7.

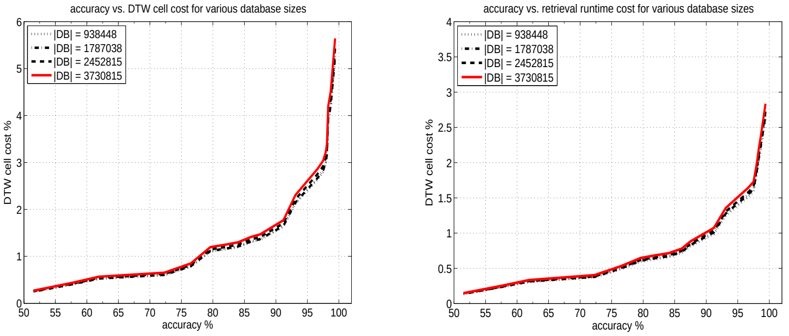

Fig. 11. Cell Cost and Retrieval Runtime of EBSM using random reference sequences for various database sizes ($|X| = 938448$, $|X| = 1787038$, $|X| = 2452815$, $|X| = 3730815$). For queries we used the training set of the Wager data set. The segmentation rate used was 50 and the PDTW averaging factor used for the filter step was set to 5.

EBSM was run using the optimal settings found by the previous experiments, i.e., segmentation rate of 50 and PDTW averaging factor of 5. Trade-offs between cell cost/retrieval runtime and accuracy are shown in Figure 11. It is apparent that the database size does not affect the performance of EBSM since the proportion of cells of the dynamic programming computation of SPRING and the refine step of EBSM is in both cases the same, irrespective of database size.

Finally, we studied the effect of the length of reference sequences. For this experiment we used the Wafer dataset for queries against the whole database sequence. To evaluate the performance we measured for each query the rank of the nearest neighbor. In Figure 12 we see in the x-axis the rank (normalized in [0,1]) and in the y-axis the percentage of queries with rank at most as equal as that shown in the x-axis. As expected, the optimal setting is when the reference sequences have length similar to that of the query (length of 150 for the reference sequences, compared to length of 152 for queries from the Wafer dataset). The more the length of the reference sequences increases or decreases from that optimal setting, the more performance deteriorates.

9.4. Experimental Evaluation of BSE

The main focus of the experimental evaluation of BSE is to demonstrate the good accuracy/efficiency tradeoffs obtained by BSE, and the robustness of BSE with respect to query size and warping width. In particular, our experiments demonstrate:

- significant speedups, at the cost of modest loss in retrieval accuracy, compared to the exact methods LB_Keogh [Keogh 2002] and DTK [Han et al. 2007]

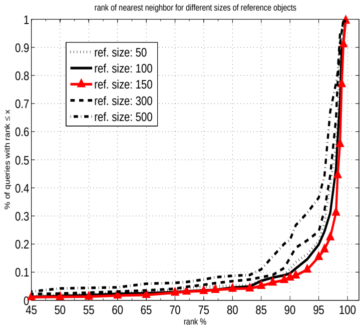

Fig. 12. Rank of nearest neighbor for the Wafer dataset using randomly generated reference sequences. The x-axis shows the rank (normalized in [0,1]) of the nearest neighbor and the y-axis shows the percentage of queries with rank at most as equal as that shown in the x-axis.

Table IV. BSE vs. Competitors: DTW cell cost using optimization

Accuracy	BSE40-50	EE40-50	DTK	LB_Keogh
100%			0.95%	26.22%
99%	0.65%	1.42%		
98%	0.59%	1.03%		
95%	0.41%	0.69%		
90%	0.33%	0.58%		
85%	0.22%	0.37%		
80%	0.14%	0.25%		

Comparison of Cell Cost for BSE with training and embedding segmentation, EE with training and embedding segmentation, DTK and LB_Keogh for the UCR dataset. Note that DTK and LB_Keogh are exact and thus have 100% retrieval accuracy.

— the performance gains of bidirectional embeddings, compared to using EBSM-style endpoint embeddings, .
— the effect of training in the new embedding scheme, and the fact that competitive results are obtained even when not using training.
— the robustness of our method with respect to query size and warping width.

To further evaluate the robustness of BSE we created an additional random walk synthetic dataset. In this dataset the database time series X was generated as follows: for each value X_i we produce a random real number r and if r is positive, $X_i = X_{i-1} + 0.005$, else $X_i = X_{i-1} - 0.005$. X_0 is set to 1.5. Queries were generated in the same way. The query size varied from 100 to 1000 in increments of 100. We used 100 queries per query size.

Before proceeding with the experimental analysis we should explicitly state the parameter settings used in the experiments. One parameter that we need to set is the dimensionality of the BSE embedding. Unless noted otherwise, we use a 40-dimensional embedding. Also, unless noted otherwise, we use $segmentation\ rate = 50$, i.e., the number of embedding segments is $\frac{|X|}{50}$.

9.4.1. Accuracy vs. Efficiency. Applying LB_Keogh with a sliding window on the UCR dataset yielded a cell cost of 0.72% with an average retrieval runtime of 8.21 seconds per query. On the other hand, the performance of DTK is poor in terms of both cell cost (18.73%) and retrieval runtime (17.93 sec). In Figures 13 and 14 we see results with respect to cell cost and retrieval runtime; the results are also summarized in Tables IV, V, VI, and VII. For an accuracy of 99% BSE embeddings (constructed via

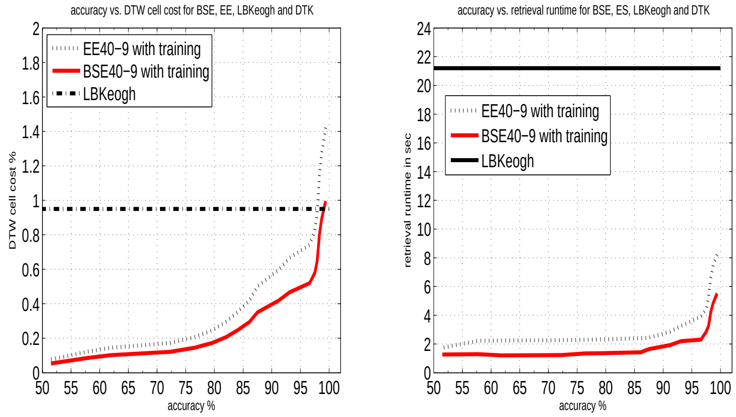

Fig. 13. Cell cost (left) and retrieval time (right) vs. retrieval accuracy attained by BSE embeddings and endpoint embeddings (EE), **both embeddings constructed using learning**, for the UCR dataset. Dimensionality = 40 and sampling rate = 9. Warping width is 5% of the query size. The cell cost is also shown for LB_Keogh (corresponding to 100% accuracy). Notice that in the left figure, the value for DTK is 26.22% and in the right figure the value for DTK is 46.77 sec and for LB_Keogh it is 21.20 sec; thus they do not appear in the plots.

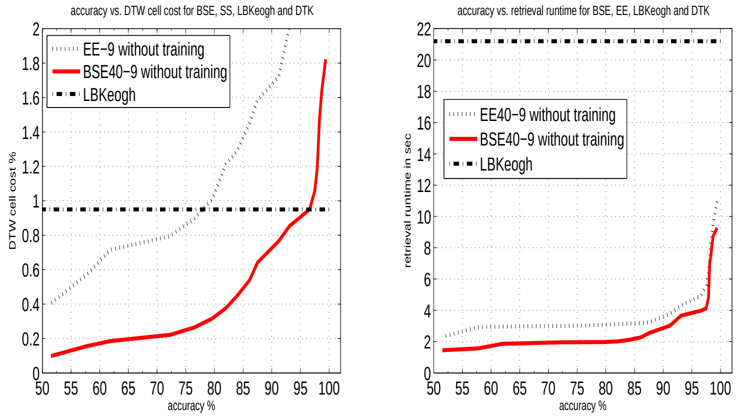

Fig. 14. Cell cost (left) and retrieval time (right) vs. retrieval accuracy attained by BSE embeddings and endpoint embeddings (EE), **both embeddings constructed using the max variance heuristic**, for the UCR dataset. Dimensionality = 40 and sampling rate = 9. Warping width is 5% of the query size. Results are also shown for LB_Keogh, as horizontal bars corresponding to the costs for 100% retrieval accuracy. Notice that in the left figure, the value for DTK is 26.22% and in the right figure it is 46.77 sec; thus they do not appear in the plots.

learning) are faster than LB_Keogh by a factor of 22.2 in terms of retrieval runtime. For an accuracy of 80%, BSE embeddings (constructed via learning) yield a speedup of two orders of magitude compared to LB_Keogh and DTK. As seen in Table VII, BSE embeddings constructed via max variance (and thus not requiring a training set of queries) also perform well, being faster by a factor of 15.8 and 39.1 over LB_Keogh, for retrieval accuracy 99% and 80% respectively.

In terms of cell cost LB_Keogh appears to have a better performance for accuracies above 95%. However, the cell cost does not consider the cost of the filter step. The filter

Table V. BSE vs. competitors: retrieval runtime using optimization

Accuracy	BSE40-50	EE40-50	DTK	LB_Keogh
100%			26.22	21.20
99%	4.45	6.81		
98%	3.49	5.23		
95%	2.20	3.51		
90%	1.91	2.68		
85%	1.51	2.26		
80%	1.47	2.20		

Comparison of Retrieval Runtime for BSE with training and embedding segmentation, EE with training and embedding segmentation, DTK and LB_Keogh for the UCR dataset. Note that DTK and LB_Keogh are exact and thus have 100% retrieval accuracy.

Table VI. BSE vs. competitors: DTW cell cost and no optimization

Accuracy	BSE40-50	EE40-50	DTK	LB_Keogh
100%			0.95%	26.22%
99%	0.77%	1.76%		
98%	0.65%	1.12%		
95%	0.52%	0.73%		
90%	0.41%	0.65%		
85%	0.33%	0.48%		
80%	0.26%	0.39%		

Comparison of Cell Cost for BSE constructed using max variance and embedding segmentation, EE constructed using max variance and embedding segmentation, DTK and LB_Keogh for the UCR dataset. Note that DTK and LB_Keogh are exact and thus have 100% retrieval accuracy.

Table VII. BSE vs. competitors: retrieval runtime and no optimization

Accuracy	BSE40-50	EE40-50	DTK	LB_Keogh
100%			26.22	21.20
99%	6.57	8.98		
98%	5.15	7.12		
95%	3.56	5.22		
90%	2.79	4.11		
85%	1.89	3.67		
80%	1.66	2.89		

Comparison of Retrieval Runtime for BSE constructed using max variance and embedding segmentation, EE constructed using max variance, DTK and LB_Keogh for the UCR dataset. Note that DTK and LB_Keogh are exact and thus have 100% retrieval accuracy.

step of LB_Keogh is much more expensive than that of BSE. This can be seen in Table V for the UCR dataset.

9.4.2. Robustness. Here we present experimental results that demonstrate that the performance of BSE embeddings is more robust than that of LB_Keogh and DTK, with respect to changes in the warping width w and changes in the length of queries.

Table VIII shows the effect of the warping width w for both LB_Keogh and BSE, as measured on the random walk dataset. For BSE, we selected an accuracy of 95%, a dimensionality of 40 and a sampling rate of 9. It can be seen that as w increases, the pruning power of LB_Keogh deteriorates fast. A similar observation is also made in

Table VIII. Warping width vs. DTW cell cost and retrieval runtime

	LB_Keogh		BSE	
Warping width	Cell Cost	Runtime	Cell Cost	Runtime
0.5%	0.52%	1.91	0.81%	0.23
1.0%	0.93%	2.87	0.82%	0.34
2.5%	1.61%	4.65	0.89%	0.55
5.0%	2.68%	7.89	0.97%	0.81
10.0%	4.68%	12.62	1.02%	1.36
15.0%	10.19%	25.33	1.16%	2.09
20.0%	25.22%	61.73	1.27%	2.86

Behavior of BSE (for 95% retrieval accuracy) vs. LB_Keogh for different warping widths for the Random Walk dataset. Query size is set to 400.

Table IX. Query size vs. DTW cell cost

Query size	BSE40-9 (95%)	LB_Keogh	DTK
100	0.375%	0.0672%	12.14%
200	0.552%	0.1972%	10.53%
300	0.765%	0.9082%	9.55%
400	0.974%	2.6834%	13.63%
500	1.183%	3.8764%	17.34%
600	1.212%	6.8772%	28.35%
700	1.491%	7.8972%	36.86%
800	1.527%	13.7644%	52.88%
900	1.753%	32.0987%	77.71%
1000	1.849%	46.5289%	89.35%

Effect of query size on Cell Cost for the Random Walk dataset. Warping width is set to 5% of the query size. For BSE we show the cell costs for 95% accuracy.

[Shou et al. 2005]. The runtime of BSE also deteriorates, but at a much smaller pace: increasing w from 0.5% of the query length to 20% of the query length makes LB_Keogh 32 times slower, and BSE about 13 times slower.

The effect of query size is studied next, by setting the warping width to 5% and varying the query size from 100 to 1000. Tables IX and X summarize our findings regarding cell cost and retrieval runtime respectively, for the two competitor methods and BSE, as measured on the random walk dataset. For BSE, we selected an accuracy of 95%, a dimensionality of 40 and a sampling rate of 9. For query sizes up to 300 the performance of DTK is improved as the query sizes increases; after that point, DTK deteriorates rapidly as the query size keeps increasing. Overall, increasing the query length from 100 to 1000 makes LB_Keogh more than 250 times slower, DTK about 38 times slower, and BSE about 8.6 times slower; BSE clearly demonstrates the slowest deterioration with increasing query length.

9.4.3. Further Analysis of BSE. This section provides a further analysis of BSE. We compare BSE embeddings with endpoint embeddings (EE), we compare performance of BSE embeddings optimized using the max variance heuristic vs. BSE embeddings optimized using learning, and we analyze the effects of dimensionality and sampling rate on the performance of BSE.

The performance of BSE is compared with that of using only endpoint embeddings (denoted as EE embeddings). In Figures 13 and 14 we can see the performance of BSE vs. EE with respect to cell cost and retrieval runtime; the results are also summarized in Tables IV, V, VI, and VII. In terms of retrieval runtime, BSE embeddings outperform EE embeddings across the board. The difference is even more pronounced for embeddings optimized via max variance; as Table VII shows, BSE embeddings lead to

Table X. Query size vs. retrieval runtime

Query size	BSE40-9 (95%)	LB_Keogh	DTK
100	0.66	1.19	15.89
200	0.72	3.42	11.23
300	0.75	5.32	9.52
400	0.81	8.57	13.56
500	0.97	14.35	19.66
600	1.35	25.92	42.33
700	1.84	49.22	84.47
800	2.58	98.80	156.22
900	3.22	173.33	311.18
1000	5.65	302.57	609.56

Effect of query size on the retrieval runtime cost for the Random Walk dataset. Warping width is set to 5% of the query size. For BSE we show the cell costs for 95% accuracy.

runtimes between 2.5 and 4.5 times smaller compared to the runtimes attained using EE embeddings.

In Figures 13 and 14, and Tables IV, V, VI, and VII we see the results obtained using BSE embeddings constructed using each of the two methods described in Section 6: the max variance heuristic and the greedy learning algorithm that uses a training set of queries. We see that the greedy learning algorithm invariably produces better results.

It is also interesting to compare how using learning affects BSE embeddings and EE embeddings. Comparing Figures 13 and 14 it can be seen that the learning method affects the performance of BSE embeddings much less than it affects the performance of EE embeddings. For example, for 99% accuracy the retrieval runtime is decreased by a factor of 1.4 for BSE embeddings, and by a factor of 4.44 for EE embeddings. In these experiments, BSE embeddings are shown to be less reliant on learning than EE embeddings. This is an additional advantage of BSE embeddings, as the learning method is not always a realistic option, as discussed in Section 6.

For this set of experiments, the sampling rate was set to 1 and the dimensionality of the embedding varied from 10 to 160. In Figure 15 we can see the performance of BSE (optimized using learning) with respect to accuracy vs. cell cost and retrieval runtime respectively. We note that an embedding of dimensionality 40 produces the best accuracy with respect to both cell cost and retrieval runtime. The fact that the cell cost (which excludes the cost of comparing high-dimensional vectors) increases as the dimensionality goes from 40 to 80 and 160 is evidence that the learning algorithm suffers from overfitting, i.e., it tries to fit the training data too much. Using more training data is the standard way to avoid overfitting.

Moreover, the effect of the segmentation rate on both cell cost and retrieval runtime is studied. For this set of experiments, the dimensionality of the embedding was set to 40 and the segmentation rate varied from 30 to 70. In Figure 16 we can see a comparison of accuracy vs. cell cost and retrieval runtime respectively for BSE. Based on the experimental evaluation on the UCR dataset, for the best dimensionality determined in the previous paragraph, the best segmentation rate is 50. At the same time, we note that sampling rates of 30, 40, 60, and 70 give results fairly similar to each other, and thus the performance of BSE embeddings is not particularly sensitive to the choice of segmentation rate.

10. DISCUSSION AND CONCLUSIONS

We have described an embedding-based framework for speeding up subsequence matching queries in large time series databases, under both unconstrained and con-

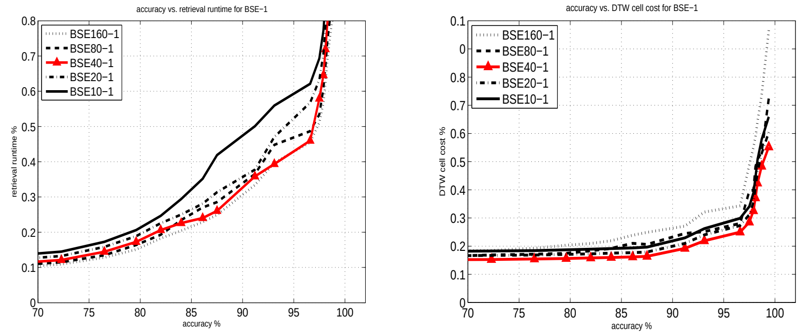

Fig. 15. Cell Cost and Retrieval Runtime of BSE embeddings optimized via learning for the UCR dataset, for different embedding dimensionalities. Sampling rate was set to 1 and the dimensionality of the embedding varies from 10 to 160. Warping width is 5% of the query size.

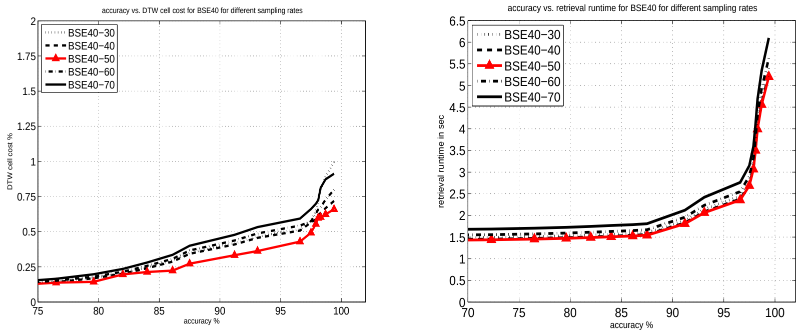

Fig. 16. Cell Cost and Retrieval Runtime of BSE embeddings optimized via learning for the UCR dataset, for different segmentation rates. The dimensionality of the embedding is set to 40 and the segmentation rate r varies from 30 to 70, meaning that in the embedding segmentation process we used $\frac{|X|}{r}$ segments. Warping width is 5% of the query size. Training has been performed on BSE.

strained DTW. By partially converting DTW-based subsequence matching to similarity search in a vector space, our framework allows for designing efficient filtering methods, that identify a relatively small number of candidate matches. The two methods derived from this framework, EBSM and BSE, were shown to significantly outperform the current state-of-the-art methods for subsequence matching.

We note that, in both EBSM and BSE, there are certain free parameters that must be specified in any implementation. Those free parameters can be divided into two categories. The first category includes parameters where it is not clear whether increasing or decreasing the value will improve accuracy or efficiency. Such parameters are the dimensionality of the embedding, the sampling rate, the segmentation rate, and the PDTW averaging factor. The value of those parameters can be chosen so as to optimize performance on a representative workload (i.e., a "training set") of queries.

The second category of free parameters are parameters that trade quality or accuracy for efficiency. For the selection of reference sequences, these parameters include the size of the training set of queries and the number of candidate reference sequences. Higher values of these parameters are expected to lead to a selection of reference sequences that is as good as, or better than, what we would obtain using lower values. In such cases, we recommend starting with relatively small values for these parameters, and increasing them exponentially until either no improvement in quality is observed, or the selection of reference sequences becomes too time consuming. At retrieval time, accuracy vs. efficiency is traded by choosing the number of candidate matches surviving each filtering operation. Clearly, the more candidates we keep, the more likely we are to include the correct match among the candidates. Choosing a good trade-off between accuracy and efficiency depends on domain-specific considerations, such as the relative cost of an inacurrate result vs. the cost of more time spent to obtain an accurate result.

An open problem, that is interesting to explore, is combining startpoint and endpoint embeddings under unconstrained DTW, where the subsequence match can have different length than the query. A related problem is to remove the constraint that the match must have the same length as the query for cDTW. This constraint is currently used not only by our method, but also by the other existing methods for subsequence matching under cDTW, i.e., LB_Keogh [Keogh 2002] and DTK [Han et al. 2007].

Another open problem is using vector indexing methods to further speed up the embedding-based filter step. An additional challenge here is that BSE embeddings are query-sensitive: the reference sequences used depend on the query length, and the final combination of startpoint and endpoint embeddings also depends on the query length. Applying standard vector indexing methods [Böhm et al. 2001; Hjaltason and Samet 2003b] in this setting is not a straightforward task, and developing appropriate indexing methods is an interesting topic for future work. An interesting work that can be used to improve similarity search over arbitrary subspaces under an L_p norm distance appeared recently [Lian and Chen 2008] and is related to this problem.

ACKNOWLEDGMENTS

Panagiotis Papapetrou has been supported in part by the Finnish Centre of Excellence for Algorithmic Data Analysis Research (AlGODAN). Vassilis Athitsos has been partially funded by grants from the National Science Foundation: IIS-0705749, IIS-0812601, CNS-0923494. This research has also been supported by a UTA startup grant to Professor Athitsos, and UTA STARS awards to Professors Chris Ding and Fillia Makedon. George Kollios and Michalis Potamias were partially supported by NSF garnt IIS-0812309. Dimitrios Gunopulos' research was supported by the SemsorGrid4Env and the MODAP EC projects.

REFERENCES

ARGYROS, T. AND ERMOPOULOS, C. 2003. Efficient subsequence matching in time series databases under time and amplitude transformations. In *International Conference on Data Mining*. 481–484.

ASSENT, I., WICHTERICH, M., KRIEGER, R., KREMER, H., AND SEIDL, T. 2009. Anticipatory dtw for efficient similarity search in time series databases. *Proc. VLDB Endow. 2,* 1, 826–837.

ATHITSOS, V., ALON, J., SCLAROFF, S., AND KOLLIOS, G. 2004. BoostMap: A method for efficient approximate similarity rankings. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 268–275.

ATHITSOS, V., HADJIELEFTHERIOU, M., KOLLIOS, G., AND SCLAROFF, S. 2005. Query-sensitive embeddings. In *ACM International Conference on Management of Data (SIGMOD)*. 706–717.

ATHITSOS, V., PAPAPETROU, P., POTAMIAS, M., KOLLIOS, G., AND GUNOPULOS, D. 2008. Approximate embedding-based subsequence matching of time series. In *ACM International Conference on Management of Data (SIGMOD)*. 365–378.

BINGHAM, E., GIONIS, A., HAIMINEN, N., HIISILÄ, H., MANNILA, H., AND TERZI, E. 2006. Segmentation and dimensionality reduction. In *SIAM International Data Mining Conference (SDM)*.

BÖHM, C., BERCHTOLD, S., AND KEIM, D. A. 2001. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys 33,* 3, 322–373.

BURKHARDT, S., CRAUSER, A., FERRAGINA, P., LENHOF, H.-P., RIVALS, E., AND VINGRON, M. 1999. q-gram based database searching using a suffix array (quasar). In *International Conference on Computational Molecular Biology (RECOMB)*. 77–83.

CHAKRABARTI, K. AND MEHROTRA, S. 2000. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *International Conference on Very Large Data Bases (VLDB)*. 89–100.

CHAN, K.-P. AND FU, A. W.-C. 1999. Efficient time series matching by wavelets. In *IEEE International Conference on Data Engineearing (ICDE)*. 126–133.

CHEN, L. AND NG, R. T. 2004. On the marriage of lp-norms and edit distance. In *International Conference on Very Large Data Bases (VLDB)*. 792–803.

CHEN, L., ÖZSU, M. T., AND ORIA, V. 2005. Robust and fast similarity search for moving object trajectories. In *ACM International Conference on Management of Data (SIGMOD)*. 491–502.

CHEN, Y., CHEN, G., CHEN, K., AND OOI, B. C. 2009. Efficient processing of warping time series join of motion capture data. In *ICDE*. 1048–1059.

CHEN, Y., NASCIMENTO, M. A., OOI, B. C., AND TUNG, A. K. H. 2007. Spade: On shape-based pattern detection in streaming time series. In *ICDE*. 786–795.

EGECIOGLU, Ö. AND FERHATOSMANOGLU, H. 2000. Dimensionality reduction and similarity distance computation by inner product approximations. In *International Conference on Information and Knowledge Management*. 219–226.

FALOUTSOS, C. AND LIN, K. I. 1995. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *ACM International Conference on Management of Data (SIGMOD)*. 163–174.

FALOUTSOS, C., RANGANATHAN, M., AND MANOLOPOULOS, Y. 1994. Fast subsequence matching in time-series databases. In *ACM International Conference on Management of Data (SIGMOD)*. 419–429.

FU, A. W.-C., KEOGH, E., LAU, L. Y. H., RATANAMAHATANA, C., AND WONG, R. C.-W. 2008. Scaling and time warping in time series querying. *The Very Large DataBases (VLDB) Journal 17,* 4, 899–921.

GIONIS, A., INDYK, P., AND MOTWANI, R. 1999. Similarity search in high dimensions via hashing. In *International Conference on Very Large Databases*. 518–529.

HAN, W.-S., LEE, J., MOON, Y.-S., AND JIANG, H. 2007. Ranked subsequence matching in time-series databases. In *International Conference on Very Large Data Bases (VLDB)*. 423–434.

HJALTASON, G. AND SAMET, H. 2003a. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence 25,* 5, 530–549.

HJALTASON, G. R. AND SAMET, H. 2003b. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems 28,* 4, 517–580.

HRISTESCU, G. AND FARACH-COLTON, M. 1999. Cluster-preserving embedding of proteins. Tech. Rep. 99-50, CS Department, Rutgers University.

KANTH, K. V. R., AGRAWAL, D., AND SINGH, A. 1998. Dimensionality reduction for similarity searching in dynamic databases. In *ACM International Conference on Management of Data (SIGMOD)*. 166–176.

KEOGH, E. 2002. Exact indexing of dynamic time warping. In *International Conference on Very Large Data Bases*. 406–417.

KEOGH, E. 2006. The UCR time series data mining archive. http://www.cs.ucr.edu/ eamonn/tsdma/index.html.

KEOGH, E., CHU, S., HART, D., AND PAZZANI, M. 1993. Segmenting time series: A survey and novel approach. In *In an Edited Volume, Data mining in Time Series Databases. Published by World Scientific*. Publishing Company, 1–22.

KEOGH, E. AND LIN, J. 2005. Hot sax: Efficiently finding the most unusual time series subsequence. In *IEEE International Conference on Data Mining (ICDM)*. 226–233.

KEOGH, E. AND PAZZANI, M. 2000. Scaling up dynamic time warping for data mining applications. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

KOUDAS, N., OOI, B. C., SHEN, H. T., AND TUNG, A. K. H. 2004. LDC: Enabling search by partial distance in a hyper-dimensional space. In *IEEE International Conference on Data Engineearing*. 6–17.

KRUSKAL, J. B. AND LIBERMAN, M. 1983. The symmetric time warping algorithm: From continuous to discrete. In *Time Warps*. Addison-Wesley.

LATECKI, L., MEGALOOIKONOMOU, V., WANG, Q., LAKÄMPER, R., RATANAMAHATANA, C., AND KEOGH, E. 2005. Elastic partial matching of time series. In *European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*. 577–584.

LEE, H. AND KIM, J. 1999. An HMM-based threshold model approach for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 21,* 10, 961–973.

LEVENSHTEIN, V. I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics 10,* 8, 707–710.

LI, C., CHANG, E., GARCIA-MOLINA, H., AND WIEDERHOLD, G. 2002. Clustering for approximate similarity search in high-dimensional spaces. *IEEE Transactions on Knowledge and Data Engineering 14,* 4, 792–808.

LI, C., WANG, B., AND YANG, X. 2007. Vgram: improving performance of approximate queries on string collections using variable-length grams. In *International Conference on Very Large Data Bases (VLDB)*. 303–314.

LIAN, X. AND CHEN, L. 2008. Similarity search in arbitrary subspaces under lp-norm. In *IEEE International Conference on Data Engineearing (ICDE)*. 317–326.

LIN, J., KEOGH, E., WEI, L., AND LONARDI, S. 2007. Experiencing sax: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery (DMKD) 15*, 107–144.

MEEK, C., PATEL, J. M., AND KASETTY, S. 2003. OASIS: An online and accurate technique for local-alignment searches on biological sequences. In *International Conference on Very Large Data Bases (VLDB)*. 910–921.

MOON, Y., WHANG, K., AND HAN, W. 2002. General match: a subsequence matching method in time-series databases based on generalized windows. In *ACM International Conference on Management of Data (SIGMOD)*. 382–393.

MOON, Y., WHANG, K., AND LOH, W. 2001. Duality-based subsequence matching in time-series databases. In *IEEE International Conference on Data Engineering (ICDE)*. 263–272.

MORGUET, P. AND LANG, M. 1998. Spotting dynamic hand gestures in video image sequences using hidden Markov models. In *IEEE International Conference on Image Processing*. 193–197.

MORSE, M. AND PATEL, J. 2007. An efficient and accurate method for evaluating time series similarity. In *ACM International Conference on Management of Data (SIGMOD)*. 569–580.

NAVARRO, G. AND BAEZA-YATES, R. 1999. A new indexing method for approximate string matching. In *Combinatorial Pattern Matching, 10th Annual Symposium*. 163–185.

OKA, R. 1998. Spotting method for classification of real world data. *The Computer Journal 41,* 8, 559–565.

PAPAPETROU, P., ATHITSOS, V., KOLLIOS, G., AND GUNOPULOS, D. 2009. Reference-based alignment in large sequence databases. *Proceedings of the Very Large Database Endowment (PVLDB) 2*, 205–216.

PARK, S., CHU, W. W., YOON, J., AND WON, J. 2003. Similarity search of time-warped subsequences via a suffix tree. *Information Systems 28,* 7.

PARK, S., KIM, S., AND CHU, W. W. 2001. Segment-based approach for subsequence searches in sequence databases. In *Symposium on Applied Computing*. 248–252.

RAFIEI, D. AND MENDELZON, A. O. 1997. Similarity-based queries for time series data. In *ACM International Conference on Management of Data (SIGMOD)*. 13–25.

RATANAMAHATANA, C. AND KEOGH, E. J. 2005. Three myths about dynamic time warping data mining. In *SIAM International Data Mining Conference (SDM)*.

RATH, T. M. AND MANMATHA, R. 2003. Word image matching using dynamic time warping. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2. 521–527.

SAKURAI, Y., FALOUTSOS, C., AND YAMAMURO, M. 2007. Stream monitoring under the time warping distance. In *IEEE International Conference on Data Engineering (ICDE)*.

SAKURAI, Y., YOSHIKAWA, M., AND FALOUTSOS, C. 2005. FTW: fast similarity search under the time warping distance. In *Principles of Database Systems (PODS)*. 326–337.

SAKURAI, Y., YOSHIKAWA, M., UEMURA, S., AND KOJIMA, H. 2000. The A-tree: An index structure for high-dimensional spaces using relative approximation. In *International Conference on Very Large Data Bases*. 516–526.

SHOU, Y., MAMOULIS, N., AND CHEUNG, D. W. 2005. Fast and exact warping of time series using adaptive segmental approximations. *Machine Learning 58,* 2-3, 231–267.

SMITH, T. F. AND WATERMAN, M. S. 1981. Identification of common molecular subsequences. *Journal of Molecular Biology 147*, 195–197.

TAO, Y., YI, K., SHENG, C., AND KALNIS, P. 2009. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD Conference*. 563–576.

TUNCEL, E., FERHATOSMANOGLU, H., AND ROSE, K. 2002. VQ-index: An index structure for similarity searching in multimedia databases. In *Proc. of ACM Multimedia*. 543–552.

VENKATESWARAN, J., LACHWANI, D., KAHVECI, T., AND JERMAINE, C. 2006. Reference-based indexing of sequence databases. In *International Conference on Very Large Databases (VLDB)*. 906–917.

VLACHOS, M., GUNOPULOS, D., AND DAS, G. 2004. Rotation invariant distance measures for trajectories. In *KDD'04*. 707–712.

VLACHOS, M., GUNOPULOS, D., AND KOLLIOS, G. 2002. Discovering similar multidimensional trajectories. In *IEEE International Conference on Data Engineering (ICDE)*. 673–684.

VLACHOS, M., HADJIELEFTHERIOU, M., GUNOPULOS, D., AND KEOGH, E. 2003. Indexing multidimensional time-series with support for multiple distance measures. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 216–225.

WANG, X., WANG, J. T. L., LIN, K. I., SHASHA, D., SHAPIRO, B. A., AND ZHANG, K. 2000. An index structure for data mining and clustering. *Knowledge and Information Systems 2,* 2, 161–184.

WEBER, R. AND BÖHM, K. 2000. Trading quality for time with nearest-neighbor search. In *International Conference on Extending Database Technology: Advances in Database Technology*. 21–35.

WEBER, R., SCHEK, H.-J., AND BLOTT, S. 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *International Conference on Very Large Data Bases*. 194–205.

WHITE, D. A. AND JAIN, R. 1996. Similarity indexing: Algorithms and performance. In *Storage and Retrieval for Image and Video Databases (SPIE)*. 62–73.

WU, H., SALZBERG, B., SHARP, G. C., JIANG, S. B., SHIRATO, H., AND KAELI, D. R. 2005. Subsequence matching on structured time series data. In *ACM International Conference on Management of Data (SIGMOD)*. 682–693.

YI, B.-K., JAGADISH, H. V., AND FALOUTSOS, C. 1998. Efficient retrieval of similar time sequences under time warping. In *IEEE International Conference on Data Engineering*. 201–208.

ZHOU, M. AND WONG, M. H. 2008. Efficient online subsequence searching in data streams under dynamic time warping distance. In *IEEE International Conference on Data Engineering (ICDE)*. 686–695.

ZHU, Y. AND SHASHA, D. 2003. Warping indexes with envelope transforms for query by humming. In *ACM International Conference on Management of Data (SIGMOD)*. 181–192.

Received September 2010; revised January 2011; accepted March 2011