Towards Faster Activity Search Using Embedding-based Subsequence Matching

Panagiotis Papapetrou [1], Paul Doliotis [2], and Vassilis Athitsos [2]
[1] Computer Science Department, Boston University, USA
[2] Computer Science and Engineering Department, University of Texas at Arlington, USA

ABSTRACT

Event search is the problem of identifying events or activity of interest in a large database storing long sequences of activity. In this paper, our topic is the problem of identifying activities of interest in databases where such activities are represented as time series. In the typical setup, the user presents a query that represents an activity of interest, and the system needs to retrieve the most similar activities stored in the database. We focus on the case where the best database matches are not segmented a priori: the database contains representations of long, continuous activity, that occurs throughout relatively extensive periods of time, and, given a query, there are no constraints as to when exactly a database match starts and ends within the longer activity pattern where it is contained. Using the popular DTW measure, the best database matches can be found using dynamic programming. However, retrieval time is linear to the size of the database and can become too long as the database size becomes larger. To achieve more efficient retrieval time, we apply to this problem a recently proposed technique called Embedding-based Subsequence Matching (EBSM), and we demonstrate that using EBSM we can obtain significant speedups in retrieval time.

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

Embedding-based subsequence matching (EBSM)

Keywords

subsequence matching, dynamic time warping, time series, embeddings

1. INTRODUCTION

Time series are a natural means of representating human activity. A time series is simply a sequence of vectors, where every vector corresponds to a measurement/observation at a specific time, and the next vector in the sequence corresponds to the measurement at the next time step. Measurements can be defined depending on the specific type of activity that we want to model: they can consist simply of 2D or 3D position of a single person, or they can be extended to incorporate more information, such as body pose, or additional persons involved in the activity.

Suppose that we have a database where we store long time series, representing uninterrupted observations obtained over relatively long time spans. Each such time series can contain observations corresponding to different activities, that occur, in sequence, at different intervals during the time spanned by that time series. A useful functionality in such databases is similarity-based activity retrieval, i.e., being able to identify the best matches for a specific activity of interest. In other words, a user can submit as a query the time series representation of a specific activity, such as walking, running, opening a closet, or turning around the corner, and the system identifies events in the database that best match the query.

An important aspect of this problem is that the database matches we want to retrieve do not consist of an entire time series stored in a database. Instead, the database matches are *subsequences* of the database time series: each match starts at a specific time step and ends at a specific time step within the long database time series that contains the match. Under these assumptions, identifying the best matches for an activity of interest is an instance of the time-series subsequence matching problem.

More formally, subsequence matching is the problem of identifying, given a query time series and a database of time series, the database *subsequence* (i.e., some part of some time series in the database) that is the most similar to the query sequence. Naturally, identifying optimal subsequence matches assumes the existence of a similarity measure between sequences, that can be used to evaluate each match. A key requirement for such a measure is that it should be robust to misalignments between sequences, so as to allow for time warps (such as stretching or shrinking a portion of a sequence along the time axis) and changes in sequence length. This requirement effectively rules out Euclidean and more general L_p measures.

Typically, similarity between time series is measured using dynamic time warping (DTW) [17], 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 [13]. However, the complexity of the DTW algorithm scales linearly with the length of the query and also scales linearly with the size of the database (i.e., the sum of the lengths of all time series in the database). While this complexity is definitely attractive compared to exhaustively matching the query with every possible database subsequence, in practice subsequence matching is still a

PETRA '09, June 09-13, 2009, Corfu, Greece. Workshop on Multimedia Event Analysis for Assistive Environments.
Copyright 2009 ACM ISBN 978-1-60558-409-6 ...$5.00.

computationally expensive operation in many real-world applications, especially in the presence of large database sizes.

In this paper we discuss EBSM (shorthand for Embedding-Based Subsequence Matching), a recently-proposed method for speeding up subsequence matching in time series databases [3]. The key idea behind EBSM is that the subsequence matching problem can be partially converted to the much more manageable problem of nearest neighbor retrieval in a real vector space. This conversion is achieved by defining an 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.

Embeddings are defined 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 embedding of the query is computed by matching the query with the reference sequences, which is typically orders of magnitude faster than matching the query with all database sequences. Then, the nearest neighbors of the embedded query are identified 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.

EBSM is an approximate method, that does 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, EBSM provides very good 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

The topic of efficient sequence matching has received significant attention in the database community. However, several methods assume that sequence similarity is measured using the Euclidean distance [6, 18, 19] or variants [1, 8, 22, 31]. Naturally, such methods cannot handle even the smallest misalignment caused by time warps. In the remaining discussion we restrict our attention to methods that are robust to such misalignments.

Dynamic time warping (DTW) [17] is a distance measure that is robust to misalignments and time warps, and it is widely used for time series matching. Time series 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 [13, 24, 26, 28, 32], and cannot be used for efficient retrieval of subsequences.

In [20] an indexing structure is proposed for unconstrained DTW-based subsequence matching, but retrieval complexity is still linear to the product of the lengths of the query and the database sequence. Furthermore, as database sequences get longer, the time complexity becomes similar to that of unoptimized DP-based matching.

A method for efficient exact ranked subsequence matching is proposed in [9]. In that method, queries and database sequences are broken into segments, and lower bounds are established using LB_Keogh [13], so as to prune the majority of candidate matches. There are two key differences between the method in [9] and the proposed EBSM method: First, the method in [9] can only find subsequence matches that have the exact same length as the query. Our method has no such limitation. In practice, for efficiency, we make the assumption that the optimal subsequence match has length between zero and twice the length of the query. This is a much milder assumption than requiring the subsequence match to have the exact same length as the query. Second, the method in [9] is only applicable to constrained DTW [13], where the warping path has to stay close to the diagonal. Our method can also be applied to unconstrained DTW.

An efficient method for retrieving subsequences under DTW is presented in [21]. The key idea in that method 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 also be used for multidimensional timeseries is PDTW [15]. In PDTW, 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 in the new sequence. We compare our method with a modified, improved version of PDTW in the experiments.

The SPRING method for subsequence matching is proposed in [23]. In SPRING, optimal subsequence matches are identified by running the DTW algorithm between the query and each database sequence. Subsequences are identified by prepending to the shorter sequence a “null” symbol that matches any sequence prefix with zero cost. The complexity of SPRING is still linear to both database size and query size. In EBSM, we use SPRING for matching the query and database sequences with the reference sequences, and for refining the embedding-based retrieval results.

Compared to SPRING, the key source of computational savings in EBSM is that expensive DTW-based matching is only performed between the query and a small fraction of the database, whereas in SPRING the query is matched to the entire database using DTW. The price for this improved efficiency is that EBSM cannot guarantee correct results for all queries, whereas SPRING is an exact method. Still, it is often desirable in database applications to trade accuracy for efficiency, and our method, in contrast to SPRING, provides the capability to achieve such trade-offs.

The method 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 [10], FastMap [5], MetricMap [29], SparseMap [11], and query-sensitive embeddings [2]. Such embeddings can be used for speeding up sequence matching, as done for example in [2, 11]. However , existing embedding methods are only applicable in the context of full sequence matching, not subsequence matching.The method proposed in this paper is applicable for subsequence matching.

3. BACKGROUND: DTW

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 [13] and [23]. 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 $(X_{w_{1,2}}, \ldots, X_{w_{|W|,2}})$:

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

In the remainder of this paper, 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 $X_{w_{|W|,2}}$.

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:

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

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

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

Essentially, to identify $M(R, X, j)$ we simply need to identify the start point i that minimizes the full distance D_{full} between R and $X^{i:j}$.

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.

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 [23]. 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 $W^*(Q,X,j_{\mathrm{opt}})$, where j_{opt} is the endpoint of the optimal match: $j_{\mathrm{opt}} = \mathrm{argmin}_{j=1,\ldots,|X|}\{D_{|Q|,j}(Q,X)\}$.

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 proposes 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 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 $F(X,j)$:

$$F^R(Q) = D_{|R|,|Q|}(R,Q)\ . \tag{15}$$

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

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:

$$F(Q) = (F^{R_1}(Q),\ldots,F^{R_d}(Q))\ . \tag{17}$$

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

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 [7, 25, 4, 12, 30, 16, 27].

Let's consider a very simple example, illustrated in Figure 1. 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:

$$F^R(Q) = D_{|R|,|Q|}(R,Q) = D_{|R|,j}(R,X) = F^R(X,j). \tag{19}$$

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:

- Perturbations along the *temporal* axis, such as repetitions, insertions, or deletions. Unfortunately, for unconstrained DTW,

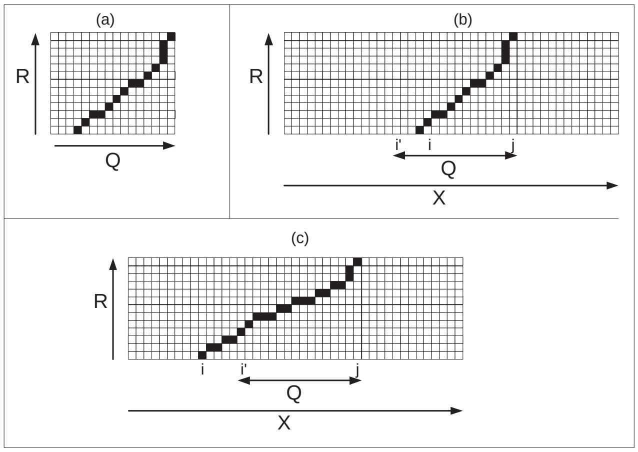

Figure 1: (a) Example of an optimal warping path $W^*(R, Q, |Q|)$ aligning a reference object R to a suffix of Q. $F^R(Q)$ is the cost of $W^*(R, Q, |Q|)$. (b) Example of a warping path $W^*(R, X, j)$, aligning a reference object R to a subsequence $X^{i:j}$ of sequence X. $F^R(X, j)$ is the cost of $W^*(R, X, j)$. 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 $F^R(Q) \neq F^R(X, j)$.

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 $M(Q, X, j)$. This issue remains a topic for future work. In our experiments, although this case did happen, we still obtained good overall results.

5. FILTER-AND-REFINED 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.

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 [10]. 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 [23], i.e., running DTW between Q and X). Retrieval accuracy, given a query, depends on whether that 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 this paper, 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 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 $F(X, j)$ and $F(X, j + 1)$, 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 $F(X, j)$. That is, we only store vectors $F(X, 1), F(X, 1 + \delta), F(X, 1 + 2\delta), \ldots$. Given $F(Q)$, we only compare it with the vectors that we have sampled. If, for a database position (X, j), its vector $F(X, j)$ was not sampled, we simply assign to that position the distance between $F(Q)$ and the vector that was actually sampled among $\{F(X, j - \lfloor\delta/2\rfloor), \ldots, F(X, j + \lfloor\delta/2\rfloor)\}$.

6. EXPERIMENTS

We perform experiments on time series data obtained from the UCR Time Series Data Mining Archive [14]. We compare EBSM to two alternative methods for subsequence matching under unconstrained DTW:

- **SPRING:** the exact method proposed by Sakurai et al. [23], 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. [15].

As formulated in [15], 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, 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 [15] is used as a filtering step, that quickly identifies candidate endpoint positions, exactly as the proposed embeddings do for EBSM. We then

Name	50Words	Wafer	Yoga
Length of each time series	270	152	426
Size of “training set” (used by us as set of queries)	450	1000	300
Number of time series used for validation (subset of set of queries)	192	428	130
Number of time series used for measuring performance (subset of set of queries)	258	572	170
Size of “test set” (used by us to generate the database)	455	6164	3000

Table 1: Description of the three 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 2,337,778).

apply the refine step on top of the original PDTW rankings, using the exact same algorithm (Algorithm 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.

6.1 Datasets

To create a large and diverse enough dataset, we combined three of the datasets from UCR Time Series Data Mining Archive [14]. The three UCR datasets that we used are shown on Table 1.

Each of the three UCR datasets contains a test set and a training set. As can be seen on Table 1, the original split into training and test sets created test sets that were significantly larger than the corresponding training sets, for two of the three datasets. In order to evaluate indexing performance, we wanted to create a sufficiently large database, and thus we generated our database using the large test sets, and we used as queries the time series in the training sets.

More specifically, our database is a single time series X, that was generated by concatenating all time series in the original test sets: 455 time series of length 270 from the 50Words dataset, 6164 time series of length 152 from the Wafer dataset, and 3000 time series of length 426 from the Yoga dataset. The length $|X|$ of the database is obviously the sum of length of all these time series, which adds up to 2,337,778.

Our set of queries was the set of time series in the original training set of the three UCR datasets. In total, this set includes 1750 time series. We randomly chose 750 of those time series as a validation set of queries, that was used for embedding optimization using Algorithm 2. The remaining 1000 queries were used to evaluate indexing performance. Naturally, the set of 1000 queries used for performance evaluation was completely disjoint from the set of queries used during embedding optimization.

6.2 Performance Measures

Our method is approximate, meaning that it does 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:** For each query Q, the DTW cell cost is the ratio of number of cells $[i][j]$ visited by Algorithm 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 1000 queries, we report the average DTW cell cost over all queries.

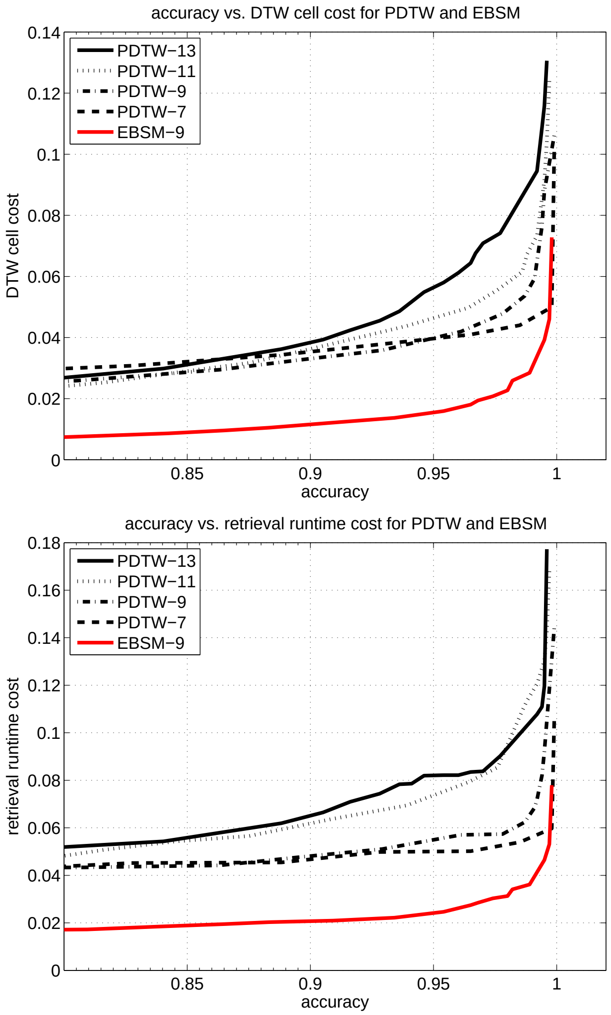

Figure 2: Comparing the accuracy versus efficiency tradeoffs achieved by EBSM with sampling rate 9 and by modified PDTW with sampling rates 7, 9, 11, and 13. The top figure measures efficiency using the DTW cell cost, and the bottom figure measures efficiency using the retrieval runtime cost. The costs shown are average costs over our test set of 1000 queries. Note that SPRING, being an exact method, corresponds to a single point (not shown on these figures), with perfect accuracy 1 and maximal DTW cell cost 1 and retrieval runtime cost 1.

- **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 the SPRING method. For the entire test set, we report the average retrieval runtime cost over all 1000 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 EBSM to PDTW, 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 EBSM.

We remind the reader that the SPRING method simply uses the standard DTW algorithm of Section 3.3. Consequently, by definition, the DTW cell cost of SPRING is always 1, and the retrieval runtime cost of SPRING is always 1. The actual average running time of the SPRING method over all queries we used for performance evaluation was: 4.43 sec/query for queries of length 152, 7.23 sec/query for queries of length 270, and 11.30 sec/query for queries of length 426. The system was implemented in C++, and run on an AMD Opteron 8220 SE processor running at 2.8GHz.

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 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.

6.3 Results

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. The embedding was optimized using a training set of queries, as described in [3].

Figure 2 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 many accuracy settings, EBSM attains costs smaller by a factor of 2 or more compared to PDTW. As highlights, for 99.5% retrieval accuracy our method is about 21 times faster than SPRING (retrieval runtime cost = 0.046), and for 90% retrieval accuracy our method is about 47 times faster than SPRING (retrieval runtime cost = 0.021).

In terms of offline preprocessing costs, selecting 40 reference sequences using Algorithm 2 took about 3 hours, and computing the 40-dimensional embedding of the database took about 240 seconds.

7. CONCLUSIONS

In this paper, we have described EBSM, a general method for subsequence matching of time series, that can be applied for the problem of similarity-based activity retrieval in databases storing long observations of activity patterns. The key advantage of EBSM is that it partially converts the computationally expensive problem of subsequence matching into the much more manageable problem of nearest neighbor search in a vector space. This conversion allows us to efficiently identify a relatively small set of candidate matches using vector comparisons.

While this paper has been oriented mainly towards the theoretical problem of efficient subsequence matching, an interesting topic is applying EBSM in specific real-world data representing patterns of activity. We are in the process of evaluating EBSM on the problem of sign spotting, i.e., the problem of identifying occurrences of signs of interest in large video databases of American Sign Language (ASL) content. Results on the problem will be useful for evaluating the impact of EBSM in real-world applications, as well as potential issues that will need to be addressed in future work.

8. REFERENCES

[1] T. Argyros and C. Ermopoulos. Efficient subsequence matching in time series databases under time and amplitude transformations. In *International Conference on Data Mining*, pages 481–484, 2003.

[2] V. Athitsos, M. Hadjieleftheriou, G. Kollios, and S. Sclaroff. Query-sensitive embeddings. In *ACM International Conference on Management of Data (SIGMOD)*, pages 706–717, 2005.

[3] V. Athitsos, P. Papapetrou, M. Potamias, P. Papapetrou, and G. Kollios. Approximate embedding-based subsequence matching of time series. In *ACM International Conference on Management of Data (SIGMOD)*, pages 365–378, 2008.

[4] Ö. Egecioglu and H. Ferhatosmanoglu. Dimensionality reduction and similarity distance computation by inner product approximations. In *International Conference on Information and Knowledge Management*, pages 219–226, 2000.

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

[6] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *ACM International Conference on Management of Data (SIGMOD)*, pages 419–429, 1994.

[7] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *International Conference on Very Large Databases*, pages 518–529, 1999.

[8] D. Q. Goldin and P. C. Kanellakis. On similarity queries for time-series data: Constraint specification and implementation. In *International Conference on Principles and Practice of Constraint Programming*, pages 137–153, 1995.

[9] W.-S. Han, J. Lee, Y.-S. Moon, and H. Jiang. Ranked subsequence matching in time-series databases. In *International Conference on Very Large Data Bases (VLDB)*, pages 423–434, 2007.

[10] G. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):530–549, 2003.

[11] G. Hristescu and M. Farach-Colton. Cluster-preserving embedding of proteins. Technical Report 99-50, CS Department, Rutgers University, 1999.

[12] K. V. R. Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *ACM International Conference on Management of Data (SIGMOD)*, pages 166–176, 1998.

[13] E. Keogh. Exact indexing of dynamic time warping. In *International Conference on Very Large Data Bases*, pages 406–417, 2002.

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

[15] E. Keogh and M. Pazzani. Scaling up dynamic time warping for data mining applications. In *Proc. of SIGKDD*, 2000.

[16] N. Koudas, B. C. Ooi, H. T. Shen, and A. K. H. Tung. LDC: Enabling search by partial distance in a hyper-dimensional space. In *IEEE International Conference on Data Engineearing*, pages 6–17, 2004.

[17] J. B. Kruskall and M. Liberman. The symmetric time warping algorithm: From continuous to discrete. In *Time Warps*. Addison-Wesley, 1983.

[18] Y. Moon, K. Whang, and W. Han. General match: a subsequence matching method in time-series databases based on generalized windows. In *ACM International Conference on Management of Data (SIGMOD)*, pages 382–393, 2002.

[19] Y. Moon, K. Whang, and W. Loh. Duality-based subsequence matching in time-series databases. In *IEEE International Conference on Data Engineering (ICDE)*, pages 263–272, 2001.

[20] S. Park, W. W. Chu, J. Yoon, and J. Won. Similarity search of time-warped subsequences via a suffix tree. *Information Systems*, 28(7), 2003.

[21] S. Park, S. Kim, and W. W. Chu. Segment-based approach for subsequence searches in sequence databases. In *Symposium on Applied Computing*, pages 248–252, 2001.

[22] D. Rafiei and A. O. Mendelzon. Similarity-based queries for time series data. In *ACM International Conference on Management of Data (SIGMOD)*, pages 13–25, 1997.

[23] Y. Sakurai, C. Faloutsos, and M. Yamamuro. Stream monitoring under the time warping distance. In *IEEE International Conference on Data Engineering (ICDE)*, 2007.

[24] Y. Sakurai, M. Yoshikawa, and C. Faloutsos. FTW: fast similarity search under the time warping distance. In *Principles of Database Systems (PODS)*, pages 326–337, 2005.

[25] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The A-tree: An index structure for high-dimensional spaces using relative approximation. In *International Conference on Very Large Data Bases*, pages 516–526, 2000.

[26] Y. Shou, N. Mamoulis, and D. W. Cheung. Fast and exact warping of time series using adaptive segmental approximations. *Machine Learning*, 58(2-3):231–267, 2005.

[27] E. Tuncel, H. Ferhatosmanoglu, and K. Rose. VQ-index: An index structure for similarity searching in multimedia databases. In *Proc. of ACM Multimedia*, pages 543–552, 2002.

[28] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 216–225, 2003.

[29] X. Wang, J. T. L. Wang, K. I. Lin, D. Shasha, B. A. Shapiro, and K. Zhang. An index structure for data mining and clustering. *Knowledge and Information Systems*, 2(2):161–184, 2000.

[30] R. Weber and K. Böhm. Trading quality for time with nearest-neighbor search. In *International Conference on Extending Database Technology: Advances in Database Technology*, pages 21–35, 2000.

[31] H. Wu, B. Salzberg, G. C. Sharp, S. B. Jiang, H. Shirato, and D. R. Kaeli. Subsequence matching on structured time series data. In *ACM International Conference on Management of Data (SIGMOD)*, pages 682–693, 2005.

[32] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *IEEE International Conference on Data Engineering*, pages 201–208, 1998.