Indexing Multi-Dimensional Time-Series

Michail Vlachos
Computer Science Department
University of California, Riverside
mvlachos@cs.ucr.edu

Marios Hadjieleftheriou
Computer Science Department
University of California, Riverside
marioh@cs.ucr.edu

Eamonn Keogh
Computer Science Department
University of California, Riverside
eamonn@cs.ucr.edu

Dimitrios Gunopulos
Computer Science Department
University of California, Riverside
dg@cs.ucr.edu

Abstract

While most time-series data mining research has concentrated on providing solutions for a single distance function, in this work we motivate the need for a single index structure that can support multiple distance measures. Our specific area of interest is the efficient retrieval and analysis of trajectory similarities. Trajectory datasets are very common in environmental applications, mobility experiments, video surveillance and are especially important for the discovery of certain biological patterns. Our primary similarity measure is based on the Longest Common Subsequence (LCSS) model, which offers enhanced robustness, particularly for noisy data, which are encountered very often in real world applications. However, our index is able to accommodate other distance measures as well, including the ubiquitous Euclidean distance, and the increasingly popular Dynamic Time Warping (DTW). While other researchers have advocated one or other of these similarity measures, a major contribution of our work is the ability to support all these measures without the need to restructure the index. Our framework guarantees no false dismissals and can also be tailored to provide much faster response time at the expense of slightly reduced precision/recall. The experimental results demonstrate that our index can help speedup the computation of expensive similarity measures such as the LCSS and the DTW.

1 Introduction

In this work we present an efficient, compact, external memory indexing technique for fast discovery of similar trajectories. Trajectory data are common in many diverse fields, including meteorology, GPS tracking, wireless applications, video tracking [8], motion capture [47], etc. Recent advances in mobile computing, sensor networks and GPS technology have made it possible to collect large amounts of spatiotemporal data. Consequently, there is an increasing interest in performing data analysis tasks [6, 46]. In mobile computing, users equipped with mobile devices move in space and register their locations at different time instants to spatiotemporal databases via wireless links. In environmental information systems, applications that track animal migrations, cyclone trajectories [13] etc., produce large datasets by storing locations of observed objects over time. Human motion data generated by simultaneously tracking various body joints can also be considered as multi-dimensional trajectories. In the field of computer graphics a fundamental operation is clustering similar movements, which leads to a multitude of applications such as interactive generation of

motions [4]. Spatiotemporal data are also produced by migrating particles in biological sciences, focusing on the discovery of subtle patterns during cellular mitoses [49]. In general, any dataset that involves storage of multi-attribute data sequences can be considered and treated as a set of multi-dimensional trajectories.

Some very important data mining tasks for multi-dimensional trajectories involve the discovery of objects that move similarly or follow closely given query patterns. An important consideration for these operations is the similarity/distance measure that will be used for discovering the most appropriate trajectories (e.g., Euclidean distance). A major difficulty that affects the choice of a good similarity measure is the presence of noise (introduced due to electromagnetic anomalies, transceiver problems, etc.). Another, is that objects may follow similar motion patterns (spatial domain) but at different rates (temporal domain). Hence, the similarity models should be robust to noise, and support elastic and imprecise matches.

For some applications, choosing the Euclidean distance as the similarity model may produce poor results because its performance degrades rapidly in the presence of noise and it is also sensitive to small variations in the time domain. This work concentrates on two similarity/distance models: The first is an extension of Dynamic Time Warping (DTW) for higher dimensions (*a distance measure*). (Note that virtually all research which uses the DTW distance has considered only the 1D case. Here, a formulation for sequences of arbitrary dimensions is presented.) The second model is a modification of the Longest Common Subsequence (LCSS), specially adapted for continuous values (*a similarity measure*). Both measures represent a significant improvement in accuracy of classification, and precision/recall of indexing, compared to the Euclidean distance. Moreover, LCSS is more robust than DTW under noisy conditions with many outliers [50].

By incorporating the ability to allow warping in time as a requirement to our model our algorithms are instantly challenged with quadratic execution time. Nevertheless, to speedup execution one can devise low cost, *upper/lower-bounding* functions. In that respect, we introduce a fast prefiltering scheme that returns upper(lower)-bound estimates for the LCSS(DTW) similarity(distance) between the query and the indexed trajectories. Apart from providing upper/lower-bounding measures that guarantee *no false dismissals*, similarity/distance approximations that significantly reduce index response time are also proposed. Finally, an indexing technique that can simultaneously support a variety of similarity/distance measures, is presented.

In order to index the trajectories and be able to use the aforementioned similarity models, we propose an indexing technique that works by approximating the trajectories with multi-dimensional MBRs and storing them in an R-tree [25]. For a given query, a *Minimum Bounding Envelope* (MBE) that covers all the possible matching areas of the query under warping conditions is constructed. The MBE is itself decomposed into MBRs and probed in the R-tree, which helps efficiently discover all candidate trajectories that could potentially be similar to the query. The index is compact and its construction time scales well with the trajectory length and the database size, allowing the suggested method to be used for massive data mining tasks.

The two most significant advantages of this approach are generality and flexibility. The user is

given the ability to pose queries of variable warping length without the need to reconstruct the index. By adjusting the width of the bounding envelope of the query the proposed method can support Euclidean distance, constrained warping, and full warping. Also, the user can choose between faster retrieval and approximate solutions, or exact answers at the expense of prolonged execution time. In both situations, experimental results indicate that the proposed solution is significantly faster than sequential scan.

The main contributions of this work are:

■ We present the first external memory index for multi-dimensional trajectories that supports multiple similarity/distance functions (such as LCSS, DTW and Euclidean), *without* the need to rebuild the index.

■ We give efficient techniques for upper(lower)-bounding and for approximating the LCSS(DTW) for a set of trajectories. We incorporate these techniques into the indexing framework.

■ We provide a flexible method that allows the user to specify queries of variable warping length. The technique can be tuned to optimize the retrieval time or the accuracy of the solution.

The rest of the paper is organized as follows. In section 2 related work is presented. In section 3 we formalize the new similarity functions by extending the LCSS and DTW models. Section 4 discusses index construction and Section 5 presents available methods for producing efficient trajectory approximations. Section 6 describes the proposed pre-filtering scheme and proves the claim of no false dismissals. Section 7 discusses various possible optimizations and Section 8 discusses how the proposed index can support multiple distance measures without the need for rebuilding. Section 9 provides the experimental evaluation of the accuracy and efficiency of the proposed approach and section 10 illustrates some real world applications that can utilize our methodology. Finally, Section 11 concludes the paper and summarizes our contributions.

2 Related Work

Previous work on trajectory similarity is mostly related with time-series analysis, which has concentrated on the use of metric L_p-norms. The advantage of this simple metric is that it enables efficient indexing by utilizing a dimensionality reduction technique [2, 51, 18]. On the other hand, the model cannot deal well with outliers and is very sensitive to small distortions in the time domain ([50]). There are a number of interesting extensions to the above model to support transformations such as scaling [12, 43], shifting [12, 22], normalization [22] and moving average [43]. Recent works that use Euclidean metrics include [30, 28]. A domain independent framework for defining queries in terms of similarity of objects is presented in [27].

Other techniques for defining time-series similarity are based on extracting features (Landmarks [39] or signatures [16]) from each series. Another approach uses the directional characteristics of a sequence at regular time intervals [42]. Ge and Smyth [21] present a sequence similarity model that is based on probabilistic matching using Hidden Markov Models. Their approach, however, does not scale well for large datasets. A recent method for clustering trajectory data is due to Gaffney and Smyth [19]; it uses a variation of the Expectation Maximization algorithm to cluster small sets of trajectories. This method is a model based approach that has severe scalability issues. All work

referenced above has concentrated primarily on 1-dimensional time-series only. Techniques that deal with multi-dimensional sequences appeared in [35, 29]. Nevertheless, these approaches support only the Euclidean distance.

Even though the vast majority of database/data mining research on time-series has focused on the Euclidean distance, virtually all real world systems that need to discover similarities utilize measures that allow time warping. In retrospect, this is not very surprising since most real world processes can evolve at varying rates. For example, in bioinformatics it is well understood that functionally related genes will express themselves in similar ways but possibly at different rates. As expected, DTW has been used for gene expression data mining [1, 5]. In that respect, chemical and industrial processes can exhibit, overall, similar patterns that are slightly offset in the time domain due to minor changes in the environment. Gollmer, and Posten [23] have demonstrated that using DTW allows robust detection of such patterns in real data. Dynamic Time Warping is a ubiquitous tool in the biometric/surveillance community; it has been used for tracking time-series extracted from video [20], classifying handwritten text [45, 37] and even fingerprint indexing [34]. Finally, perhaps the most familiar example of easily classifiable patterns which, nevertheless, cause problems to L_p metrics, are electrocardiograms. The cardiological community has used DTW for several decades [48].

While the above examples testify to the utility of a time warped distance measure, they all echo the same complaint: DTW has serious scalability issues. Work that attempted to mitigate its large computational cost has appeared in [33] and [52], where lower bounding measures are introduced for speeding up execution. However, these lower bounds can be very loose approximations of the original distance, especially for normalized data. In [38] a different approach based on suffix trees was used for indexing DTW. Nonetheless, the index required excessive disk space (about 10 times the size of the original data).

The flexibility provided by DTW is very important. However, its effectiveness deteriorates for noisy data since by matching all sequence points it also matches the outliers, distorting the true distance between the sequences. An alternative approach is based on *Longest Common Subsequence* (LCSS), which is a variation of the edit distance [36]. The basic idea is to match two sequences by allowing them to stretch without rearranging the order of the elements but permitting some elements to remain *unmatched*. Using the LCSS of two sequences, one can define a distance measure using the length of the matched subsequence [3, 10, 9, 14]. In [50] a main memory index for LCSS was proposed. It also demonstrated that while LCSS presents similar advantages to DTW it does not share its volatile performance in the presence of outliers.

The work of [31] is closest in spirit to our approach, however, it only addresses 1-dimensional time-series and supports a single distance measure. The author uses constrained DTW as the distance function, and creates an envelope around all indexed sequences. The envelope is approximated by a modified version of a Piecewise Approximation, which is later stored as a set of equi-length MBRs in an R-tree [25]. A lower bounding function that produces significantly tighter approximations of the original distance than the previous approaches is also introduced. However, by using

DTW such an approach is susceptible to high bias of outliers. Also, the fixed MBR size (even though it simplifies the index operations) can lead to degenerate approximations of the original sequence. Moreover, the embedding of the envelope in the indexed sequences can slow the index construction time and limit the user's query capabilities to a predefined warping length.

The use of LCSS as our primary similarity measure lends itself to a more natural use of the R-tree, where the similarity estimates are computed by simply calculating MBR intersections. Additionally, unline all other work, the proposed index can support multiple distance measures and since it is not constructed for a specified matching window, the user can pose queries of variable warping length.

3 Distance Measures

In this section we present extension of the LCSS and DTW models to describe the similarity between trajectories.

3.1 LCSS Model For Multi-dimensional Trajectories

The original LCSS model refers to 1-dimensional sequences, it must therefore be extended to the multi-dimensional case. In addition, the LCSS paradigm matches discrete values, however in our model we want to also allow matchings when the values are within a certain range in space and time (note that this also helps in avoiding distant and degenerate matchings).

Let's assume that measurements are taken at fixed and discrete time intervals. If this is not the case then interpolation can be used [40, 24]. Let A and B be 2-dimensional trajectories with sizes n and m respectively, where $A = ((a_{x,1}, a_{y,1}), \ldots, (a_{x,n}, a_{y,n}))$ and $B = ((b_{x,1}, b_{y,1}), \ldots, (b_{x,m}, b_{y,m}))$. Let $Head(A) = ((a_{x,1}, a_{y,1}), \ldots, (a_{x,n-1}, a_{y,n-1}))$.

Definition 1 *Given integers δ and ϵ, we define $LCSS_{\delta,\epsilon}(A, B)$ as follows:*

$$LCSS_{\delta,\epsilon}(A,B) = \begin{cases} 0 \quad if\ A\ or\ B\ is\ empty \\ 1 + LCSS_{\delta,\epsilon}(Head(A), Head(B)) \\ \quad if \quad |a_{x,n} - b_{x,m}| < \epsilon \\ \quad and\ |a_{y,n} - b_{y,m}| < \epsilon \\ \quad and\ |n - m| \leq \delta \\ max(LCSS_{\delta,\epsilon}(Head(A), B), \\ \quad LCSS_{\delta,\epsilon}(A, Head(B))), \\ \quad otherwise \end{cases}$$

Constant δ controls the flexibility of matching in the time domain and constant ϵ is the matching threshold is space. This LCSS model can be computed using dynamic programming. Its time complexity is in the order of $O(\delta(n+m))$, if we only allow a matching window δ in time [14]. An instance of the dynamic programming execution between two trajectories is depicted in Figure 1, where the gray region indicates the matching window in time. Extensions to more dimensions are straightforward.

The computed value of LCSS is unbounded and depends on the length of the compared sequences. In order to support sequences of variable lengths, the values have to normalized. One can derive a normalized distance based on the LCSS similarity as follows:

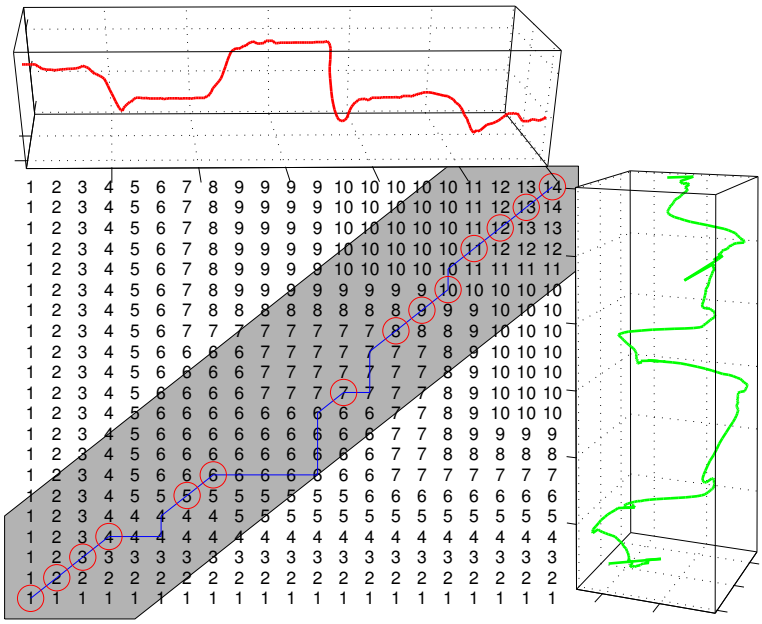

Figure 1: Dynamic programming computation of LCSS. The matching windows in time is indicated by the gray area ($\delta = 6$).

Definition 2 *The distance $D_{\delta,\epsilon}$ expressed in terms of the LCSS similarity between two trajectories A and B is given by:*

$$D_{\delta,\epsilon}(A,B) = 1 - \frac{LCSS_{\delta,\epsilon}(A,B)}{\min(n,m)} \tag{1}$$

3.2 DTW Model For Multi-dimensional Trajectories

This section presents an extension of the original DTW function as described by Berndt and Clifford [7] for multi-dimensional trajectories. For simplicity we talk about 2-dimensional trajectories.

Definition 3 *The DTW distance between 2-dimensional trajectories A and B is defined as follows:*

$$\begin{aligned} DTW(A,B) \quad = \quad & L_p((a_{x,n}, a_{y,n}), (b_{x,m}, b_{y,m})) + \\ & \textbf{\textit{min}}\{DTW(Head(A), \\ & \quad Head(B)), DTW(Head(A), B), \\ & \quad DTW(A, Head(B))\} \end{aligned} \tag{2}$$

L_p is any L_p-norm. The computation of DTW utilizes a dynamic programming technique similar to LCSS. Again, constraining the matching region within δ, the time required to compute DTW is in the order of $O(\delta(n+m))$, the same with LCSS. To normalize distances between sequences of different lengths, the quantity in equation 3 should be normalized by the length of the warping path. Figure 2 shows an example of a DTW matching between two trajectories.

3.3 Constraining the Warping

We have already mentioned that restricting the allowed warping length in time, can help speed up the execution of the dynamic programming algorithms substantially. However, as we will demonstrate

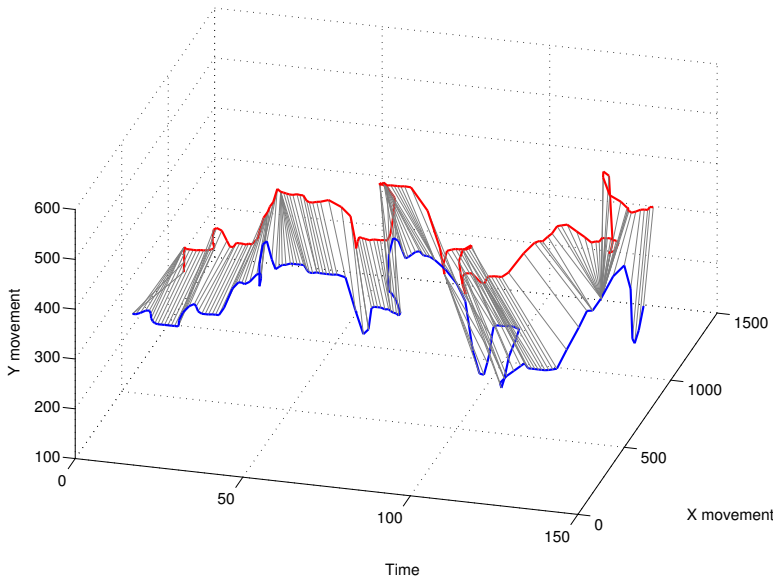

Figure 2: DTW matching between two 2-dimensional trajectories.

(using real datasets) the restrained matching window can offer more advantages:

1. For most datasets there is no need to perform full length warping. In practice, constraining the time warping to be at most 20% of the sequence's length, proves to be sufficient in most applications. Usually, convergence in the pairwise similarity between sequences is observed, after a certain warping length. Therefore, allowing wider warping at the expense of prolonged execution time, would not yield substantial changes in the computed similarity.

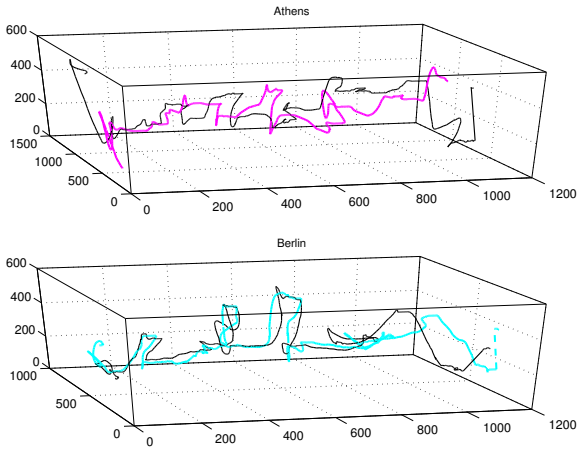

Figure 3: Two words of the cameraMouse dataset (two instances each).

Using the *Camera Mouse* program [8], we have obtained multiple instances of various words created by tracking human movement over time while utilizing a 'virtual keyboard'. In this manner, spatiotemporal sequences representing various words were collected (Figure 3). In figure 4 one can observe the trend in the computed LCSS similarity, for increasing warping

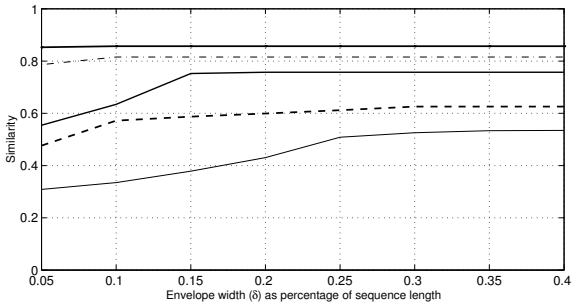

Figure 4: Pairwise LCSS similarity over increasing warping windows).

lengths, for five pairwise computations. It is apparent that the similarity seems to follow the law of diminishing returns, since warping length larger than 20% does not change the computed sequence similarity. Similar behaviour was observed for all the datasets we had at our disposal.

2. For certain datasets extended warping not only yields no additional gains, but it also hurts accuracy. The following simple experiment was performed: We tried to label each sequence of the test datasets using a "leave-one-out" classification scheme. Therefore, each trajectory was given the label of its nearest neighbor among all remaining sequences in each iteration. Figure 5 illustrates the accuracy results obtained for the *cameraMouse* dataset and a subset of the Australian Sign Language (*ASL*) dataset [1]. While for the *cameraMouse* data extensive warping does not improve accuracy, in the case of *ASL* increasing the warping to more than 30%, in fact, penalizes accuracy. It is apparent that excessive matching envelopes not only distort the true distance, but also "force" instances of words from different classes to match with one another by allowing long and degenerate matching correspondences. Very similar results on the utility of tight constraints have been noted in [44].

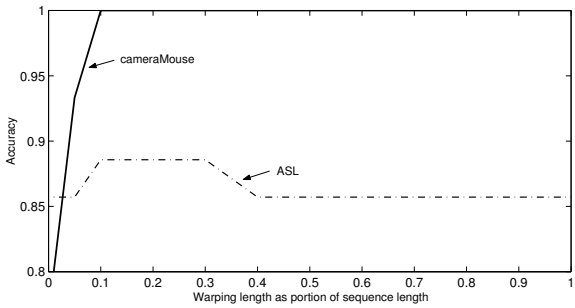

Figure 5: "Leave-one-out" classification accuracy for two datasets. We observe that excessive warping can hurt the performance in certain datasets.

Clearly, constraining the time warping, not only improves running time, but is essential for improving classification/clustering accuracy as well.

[1] `http://kdd.ics.uci.edu/databases/auslan2/auslan.data.html`

4 Index Construction

Even though imposing a matching window δ can help speedup the execution, the computation can still be quadratic when δ is a significant portion of the sequence's length. Therefore, comparing a query to all the trajectories becomes intractable for large databases. We are seeking ways to avoid examining the trajectories that are very distant to the given query. This can be accomplished by discovering a close match to the query, as early as possible. A fast pre-filtering step can be employed that eliminates the majority of distant matches. Only for some qualified sequences will the costly (but accurate) quadratic time algorithm be executed. This scheme is shown in figure 6 and has also been successfully used in [52, 31]. There are certain preprocessing steps that need to be followed:

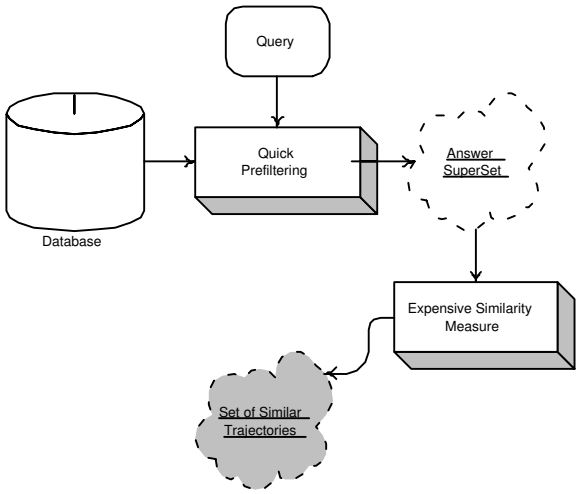

Figure 6: Very distant trajectories can be discarded by using a fast pre-filtering step. For the remaining trajectories an expensive, but accurate, similarity measure can be used.

1. The trajectories are segmented into MBRs, which are stored in an R-tree.
2. Given a query Q, the areas of possible matching are discovered by constructing its *Minimum Bounding Envelope* (MBE_Q).
3. MBE_Q is decomposed into MBRs that are probed in the index.
4. Based on the MBR intersections, similarity estimates are computed and the exact LCSS (or DTW) is performed only on the qualified trajectories.

These steps are illustrated in figure 7. The following sections explain in detail how these steps can be applied when using the LCSS and DTW models.

4.1 Upper-Bounding the LCSS

Consider first a 1-dimensional trajectory and let sequence $A = (a_{x,1}, \ldots, a_{x,n})$. Ignoring for now the parameter ϵ, we would like to perform a very fast $LCSS_\delta$ match between sequence A and some query $Q = (q_{x,1}, \ldots, q_{x,n})$. Suppose that we replicate each point $q_{x,i}$ for δ time instances before

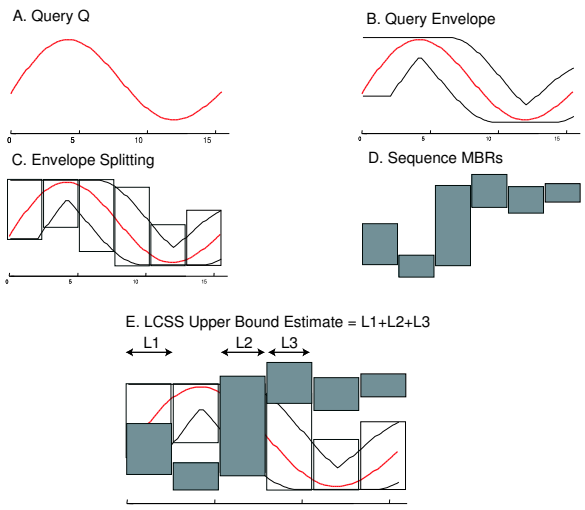

Figure 7: An 1-dimensional example of the proposed approach. The query is extended into a bounding envelope, which in turn is approximated with a set of MBRs. Overlap between the query and the index MBRs suggest areas of possible matching and yield candidate trajectories.

and after time i. The envelope that includes all these points defines the areas of possible matching. Everything outside this envelope can never be matched.

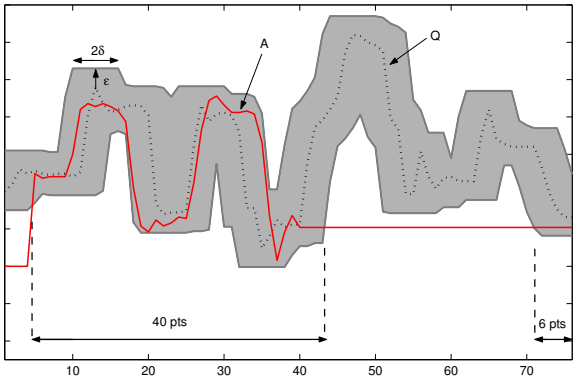

Figure 8: The *Minimum Bounding Envelope* (MBE) within δ in time and ϵ in space of a sequence. Everything that lies outside this envelope can never be matched.

We call this envelope, the *Minimum Bounding Envelope* (MBE) of a sequence. Once the matching within ϵ in space is incorporated, the new envelope should extent ϵ above and below the original envelope in space (Figure 8). The notion of a bounding envelope can be trivially extended to more dimensions. For instance, $MBE(\delta, \epsilon)$ for a 2-dimensional trajectory $Q = ((q_{x,1}, q_{y,1}), \ldots, (q_{x,n}, q_{y,n}))$ covers the following area:

$$EnvLow \leq MBE(\delta, \epsilon) \leq EnvHigh,$$

where for dimension d at position i:

$$\begin{cases} EnvHigh_{d,i} = max(q_{d,j} + \epsilon) & , |i - j| \leq \delta \\ EnvLow_{d,i} = min(q_{d,j} - \epsilon) & , |i - j| \leq \delta \end{cases}$$

The LCSS similarity between the envelope of Q and a sequence A is defined as:

$$LCSS(MBE_Q, A) = \sum_{i=1}^{n} \begin{cases} 1 & \text{if A[i] within envelope} \\ 0 & \text{otherwise} \end{cases}$$

For example, in figure 8 the LCSS similarity between MBE_Q and sequence A is 46. This value represents an *upper-bound* of the *similarity* of Q and A. One can use the MBE_Q to compute a *lower-bound* on the *distance* between trajectories, as well:

Lemma 1 *For any two trajectories Q and A the following holds: $D_{\delta,\epsilon}(MBE_Q, A) \leq D_{\delta,\epsilon}(Q, A)$,*

Proof (Sketch): $D_{\delta,\epsilon}(MBE_Q, A) = 1 - \frac{LCSS_{\delta,\epsilon}(MBE_Q, A)}{min(|Q|,|A|)}$, therefore it is sufficient to show that: $LCSS_{\delta,\epsilon}(MBE_Q, A) \geq LCSS_{\delta,\epsilon}(Q, A)$. This is true since MBE_Q by construction contains all possible areas within δ and ϵ of the query Q. Therefore, no possible matching points will be missed. □

4.2 Lower-Bounding the DTW

Before presenting our lower-bounding approach for the DTW, we will briefly revisit other lower-bounds that appeared in the literature. While referring the interested reader to the original papers for detailed explanations, below we give a visual intuition and brief description of each.

Note that both these lower-bounding functions where originally defined only for 1-dimensional time-series (and are explained below for the 1-dimensional case). However their extensions to multi-dimensional time-series are straightforward.

4.2.1 Existing lower-bounding measures

The lower-bounding function introduced by Kim et al. [33] (hereafter referred to as LB-Kim), works by extracting a 4-tuple feature vector from each sequence. The features are the first and last elements of the sequence, together with the maximum and the minimum values. The maximum between the squared differences of corresponding features is reported as the lower-bound. Figure 9 illustrates the idea.

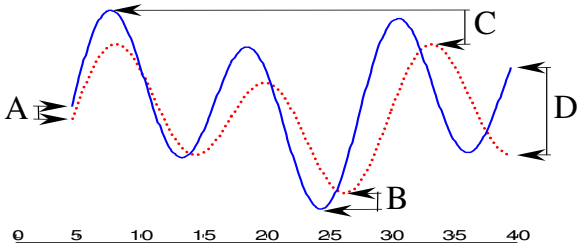

Figure 9: A visual intuition of the lower-bounding measure introduced by Kim et al. The maximum of the squared difference between the first (A), last (D), minimum (B) and maximum points (C) of the two sequences is returned as the lower-bound.

The lower-bounding function introduced by Yi et al. [52] (hereafter referred to as LB-Yi) takes advantage of the observation that all the points in one sequence that are larger (smaller) than the

maximum (minimum) of the other sequence must contribute at least the squared difference of their value and the maximum (minimum) value of the other sequence, to the final DTW distance. Figure 10 illustrates the idea.

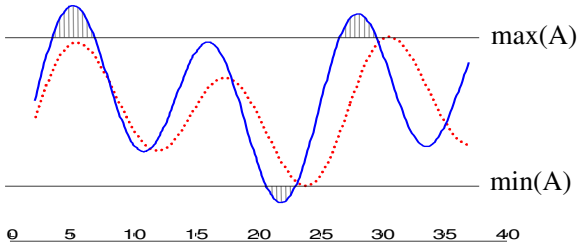

Figure 10: A visual intuition of the lower-bounding measure introduced by Yi et al. The sum of the squared length of the gray lines represents the minimum of the corresponding points' contribution to the overall DTW distance, and thus can be returned as the lower-bounding measure.

Note that while LB-Kim is trivially indexable, LB-Yi is not (since it requires the original query). Its utility comes from its use in conjunction with a technique for approximate indexing of DTW that utilizes FastMap [17]. The idea is to embed the sequences into Euclidean space such that the distances structure of the original space is approximately preserved. Then, a traditional multi-dimensional index structure can be used to index the Euclidean space. The LB-Yi function is used to prune some of the inevitable false hits that will be introduced by this method.

4.2.2 Lower Bounding the multidimensional DTW distance

In order to approximate the DTW distance first we construct the MBE of the query Q, described by a low and high envelope (as defined above but setting $\epsilon = 0$). Contrary to the LCSS model, instead of calculating overlapping regions we evaluate the distance of the MBE_Q from all other sequences. For sequences Q and A with dimensionality D, the distance between A and MBE_Q is:

$$DTW(MBE_Q, A) = \sqrt{\sum_{d=1}^{D}\sum_{i=1}^{n}\begin{cases}(a_{d,i} - EnvHigh_{d,i})^2 & \text{if } a_{x,i} > EnvHigh_{d,i} \\ (a_{d,i} - EnvLow_{d,i})^2 & \text{if } a_{x,i} < EnvLow_{d,i} \\ 0 & \text{otherwise}\end{cases}}$$

The function can be perceived as the Euclidean distance between any part of the candidate matching sequence not falling within the envelope and the nearest (orthogonal) corresponding section of the envelope of Q, as depicted in Figure 11. This lower bound could be seen as a generalization for multidimensional time-series of the technique proposed in [31].

It can be proven that this distance *lower-bounds* the actual time warping distance. A proof for the 1-dimensional case appeared in [31]. The extension to multiple dimensions is straightforward. This measure has since been extended and used by several researchers, including [53, 45].

Since the tightness of this bound is proportional to the number and length of the gray hatch lines, we can see, in this example at least, that the new lower-bound provides a tighter bound than LB-Kim or LB-Yi. It should also be noted that the proposed measure will always be at least as

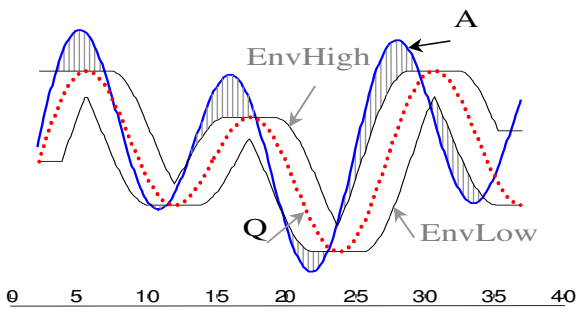

Figure 11: An illustration of the lower-bounding function for DTW. The original sequence Q (shown dotted), is enclosed in a *Minimum Bounding Envelope*. The squared sum of the distances from every part of sequence A not falling within MBE_Q, to the nearest orthogonal edge of MBE_Q is returned as the lower-bound.

good as LB-Yi, which is a special case arising when the bounding envelope is allowed to cover the whole sequence length.

In the last two sections we have described ways to lower-bound the similarity/distance (whether it is computed using LCSS or DTW). According to the GEMINI framework [2], we can use the new distance functions to create an index that guarantees no false dismissals. However, the described upper/lower-bounds can be computed using the raw trajectory data. In the sections that follow we will approximate the trajectories using a number of MBRs, to accommodate their storage into a multi-dimensional R-tree. We will show how to compute the upper/lower-bounds using only the trajectory approximation MBRs.

5 MBR generation

Given a multi-dimensional trajectory (or an MBE) our objective is to minimize the volume of the sequence using k MBRs. Clearly, the best approximation of a trajectory (or an MBE) using a fixed number of MBRs is the set of MBRs that completely contain the sequence and minimize the volume consumption. The following lemma holds:

Lemma 2 *Minimizing the volume of the Minimum Bounding Envelope, minimizes the expected similarity approximation error.*

Four different approaches are considered:

1. **k-Optimal**. The k MBRs of a sequence that take up the least volume can be obtained by using a dynamic programming algorithm that requires $O(n^2k)$ time [26], where n is the length of the given sequence. Since this approach is very expensive for large databases, we are motivated to consider approximate and faster solutions.

2. **Equi-Split**. This technique produces MBRs of fixed length. It is a simple approach with cost linear in the length of a sequence. However, in pathological cases increasing the number of MBRs can result in larger space utilization, therefore the choice of the MBR length becomes a critical parameter (see Figure 12 for an example).

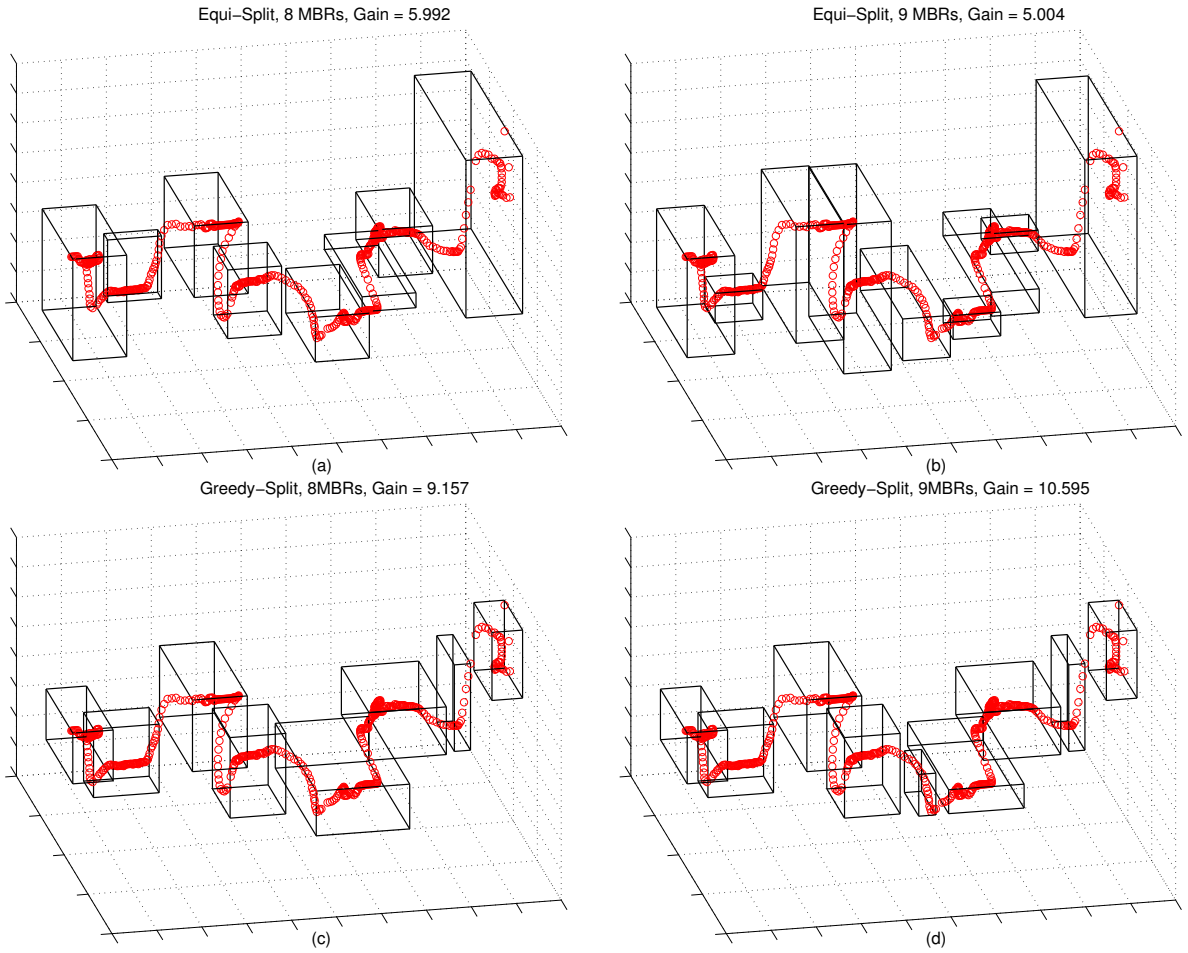

Figure 12: *(a):* 8 MBRs produced using Equi-Split. The volume gain over having only one MBR is 5.992. *(b):* Segmenting into 9 MBRs decreases the volume gain to 5.004. So, disk space is wasted without providing a better approximation of the trajectory. *(c):* 8 MBRs using Greedy-Split. The volume gain over having only one MBR is 9.157. *(d):* Every additional split will yield better space utilization. Segmentation into 9 MBRs increases volume gain to 10.595.

3. **Random-Split**. With this approach the positions where the trajectory is split into consecutive MBRs are produced randomly. As we shall see in our experiments this approach performs competitive with Equi-Split for a number of datasets [41].

4. **Greedy-Split**. The Greedy approach is our implementation choice in this paper. Initially, we assign an MBR to each of the n sequence points and at each subsequent step we merge the consecutive MBRs that will introduce the least volume consumption. The algorithm has a running time of $O(nlogn)$. A sketch of the method in shown in Algorithm 1. Alternatively, instead of assigning the same number of splits to all objects, according to our space requirements we can assign a total of K splits to be distributed among all objects. This method can provide better results, since we assign more splits to the objects that will yield more space

gain. Also, this approach is more appropriate when one is dealing with sequences of different lengths. The complexity of this approach is $O(K + NlogN)$, for a total of N objects [26].

Algorithm 1 Greedy-Split

Input: Trajectory T, integer k denoting the number of final MBRs.
Output: A set of MBRs that cover T.

```
for 0 ≤ i < n do
    Compute the volume of the MBR produced by merging T_i and T_{i+1}.
    Store the results in a priority queue.
end for
while #MBRs < k do
    Using the priority queue merge the pair of consecutive MBRs that yield the smallest increase in volume.
    Delete the two merged MBRs and insert the new one in the priority queue.
end while
```

Using more splits always leads to smaller or equal volume consumption. In Figure 12 we examine how additional MBRs affect the volume gain of Equi and Greedy-split. The volume gain conveys how much space reduction is achieved using k MBRs when compared to using only one MBR. For this specific example, when using Equi-Split an additional MBR in fact *increases* space utilization, while the Greedy-Split performs better consistently. A similar greedy algorithm is used for splitting the MBE of the query trajectory Q, as well.

6 Quick Pruning of Dissimilar Trajectories

Suppose that we have an index with the segmented trajectories and the user provides a query Q. Our goal is the discovery of the k closest trajectories to the given query according to the LCSS similarity. A prefiltering step will aid the quick discovery of a close match to the query, helping us discard the distant trajectories without using the costly quadratic algorithm. Therefore, in this phase, we compute estimates of the similarity between the query and the indexed sequences using their MBRs.

Algorithm 2 shows how to find the closest trajectory to a given query according to the LCSS similarity. This algorithm can be adjusted to return the kNN sequences simply by comparing with the k^{th} $bestSoFar$ match. DTW can be handled by reversing the signs in the equations and utilizing the relevant estimates. Next, we examine possible similarity estimates for LCSS and DTW. Some of them guarantee that a best match will be found (they lower-bound the original distance or upper-bound the original similarity), while others provide faster but approximate results.

6.1 Estimates for the LCSS

We will show how to compute estimates of the LCSS similarity based on the geometric properties of the trajectory MBRs and their intersections. An upper-bound estimate is provided by the length of the MBR intersection and an approximate estimate is given as a parameter of the intersecting volume. To formalize these notions first we present several operators.

Each trajectory T can be decomposed into a number of MBRs. The i_{th} 3-dimensional MBR of

Algorithm 2 Prune according to LCSS similarity estimates

Input: Query Q, Index I with trajectory MBRs, Method
Output: Most similar trajectory to Q is returned

```
1: Box Env = constructMBE_{δ,ε}(Q)
2: Vector V_Q = CreateMBRs(Env)                          ▷ V_Q contains a number of boxes
3: Priority queue PQ ← ∅                  ▷ Keep trajectories sorted according to similarity estimates
4: for each box B in V_Q do
5:     V = I.intersectionQuery(B)          ▷ V contains all trajectory MBRs that intersect with B
6:     if Method == Exact then PQ ← computeL-SimilarityEstimates(V, B)        ▷ upper-bound
7:     else  PQ ← computeV-SimilarityEstimates(V, B)                          ▷ approximate
8: end for
9: BestSoFar = 0; Best ← ∅
10: while PQ not empty do
11:     E ← PQ.top
12:     if E.estimate < BestSoFar then break
13:     else
14:         D = computeLCCS_{δ,ε}(Q, E)                                        ▷ exact
15:         if D > BestSoFar then BestSoFar = D; Best ← E
16: end while
17: Report Best
```

T consists of six numbers: $M_{T,i} = \{t_l, t_h, x_l, x_h, y_l, y_h\}$. Now, let us define the operators $\bigcap_t^{(c)}$, $\bigcap_t^{(p)}$ and $\bigcap_V$ between two MBRs $M_{P,i}$ and $M_{R,j}$, belonging to objects P and R, respectively:

1. $\bigcap_t^{(c)}(M_{P,i}, M_{R,j}) = ||Intersection||_t$, ($c$ stands for 'complete')
   where $M_{R,j}.x_l \leq M_{P,i}.x_l \leq M_{R,j}.x_h$ and
   $M_{R,j}.x_l \leq M_{P,i}.x_h \leq M_{R,j}.x_h$ and
   $M_{R,j}.y_l \leq M_{P,i}.y_l \leq M_{R,j}.y_h$ and
   $M_{R,j}.y_l \leq M_{P,i}.y_h \leq M_{R,j}.y_h$
   or similarly by rotating $M_{R,j} \rightleftharpoons M_{P,i}$

   Therefore, this operator computes the time intersection of two MBR when one fully contains the other in the x,y dimensions.

2. $\bigcap_t^{(p)}(M_{P,i}, M_{R,j}) = ||Intersection||_t$, otherwise ($p$ stands for 'partial')

3. $\bigcap_V(M_{P,i}, M_{R,j}) = ||Intersection||_t * ||Intersection||_x * ||Intersection||_y$

We can find upper-bounds or approximate estimates for the similarity:
1. **Upper-bound estimates (L-similarity estimate)**. Such estimates are computed using the following data structures:
■ The list $L_{t,complete}$, an element $L(P)$ of which is defined as:

$$L(P) = \sum_m \sum_n M_{Q,m} \bigcap_t^{(c)} M_{P,n}$$

where Q is a query and P is a trajectory in the index. So the list stores for *each* trajectory P the total time of the intersection between P's and Q's MBRs. The list records only the intersections

where a query MBR is fully contained in all spatial dimensions by a trajectory MBR (or vice versa; see Figure 13, top-right).

■ The list $L_{t,partial}$, an element $L(P)$ of which is defined as:

$$L(P) = \sum_m \sum_n M_{Q,m} \bigcap\nolimits_t^{(p)} M_{P,n}$$

This list records for each sequence the total intersection in time for those query MBRs that are not fully contained within the spatial dimensions of the trajectory MBRs (or vice versa; see Figure 13, top-left).

Regarding a query Q for any trajectory P, the sum of $L_{t,complete}(P) + L_{t,partial}(P)$ will provide an *upper-bound* on the similarity of P and Q:

$$LCSS(P,Q) \leq L_{t,complete}(P) + L_{t,partial}(P)$$

Subsequently, we can use these estimates to provide a guarantee that *no false dismissals* will be introduced.

The reason for the distinction of the *L-similarity estimate* into two separate lists derives from the fact that the estimates stored in list $L_{t,partial}$ can significantly overestimate the LCSS similarity. A more detailed explanation is deferred until Section 7, where potential optimizations will be delineated.

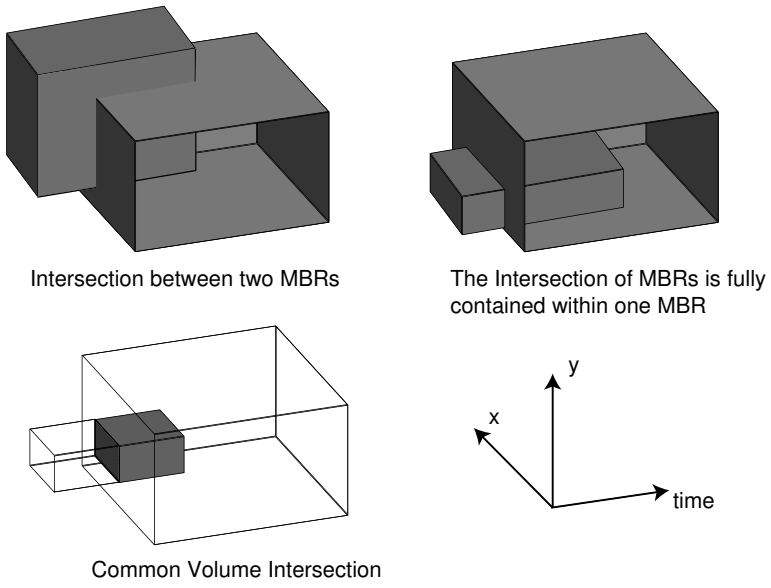

Figure 13: *Top-left*: Intersection recorded in list $L_{t,partial}$. *Top-right*: Intersection recorded in list $L_{t,complete}$. *Bottom-left*: Percentage of Volume Intersection kept in L_V.

2. **Approximate estimates (V-similarity estimate)**. This second estimate is based on the intersecting volume of the MBRs. This type of estimates are stored in list L_V:

■ Any element $L_V(P)$ of list L_V records similarity estimates between trajectory P and query Q,

based on the total volume intersection between the MBRs of P and Q.

$$L(P) = \frac{1}{length(P)} \sum_m \sum_n \frac{M_{Q,m} \bigcap_V M_{P,n}}{||M_{Q,m}||_V} ||M_{Q,m}||_t$$

where $||M||_V$ denotes the volume of MBR M and $||M||_t$ its length on the time dimension.

The *L-similarity* overestimates the $LCSS_{\delta,\epsilon}$ between two sequences A and B and so it can be deployed for the design of an index structure.

Lemma 3 *The use of the L-similarity estimate upper-bounds the $LCSS_{\delta,\epsilon}$ similarity between two sequences A and B and therefore does not introduce any false dismissals.*

The V-similarity estimate can be used for approximate query answering. Even though it does not guarantee the absence of false dismissals the results will be close to the optimal ones, with high probability. Also, because this estimate provides a tighter approximation of the original distance, we expect faster response time. Indeed, as we show in the experimental section, index performance improves while the error in similarity is typically around 5%.

6.2 Estimates for the DTW

When using the DTW distance, the index helps us obtain a lower-bound of the actual distance, by evaluating the distance of the query MBE and the trajectory MBRs.

Supposing that the time dimension corresponds to $d = 1$ (from a total of D dimensions), the `MINDIST` function between two MBRs Q and P is:

$$MINDIST(Q,P) = \sqrt{\sum_{2 \le d \le D} Q \bigcap_t R \times x_d^2}$$

$$where \;\; x_d = \begin{cases} |h_{Q,d} - l_{P,d}|, & if \;\; h_{Q,d} < l_{P,d} \\ |l_{Q,d} - h_{P,d}|, & if \;\; h_{P,d} < l_{Q,d} \\ 0, & otherwise \end{cases}$$

and $\bigcap_t$ is an operator that computes the intersection of two MBRs in the time dimension. The overall distance between the MBRs underestimates the true distance of the trajectories, thus no false dismissals are introduced. Using the MBRs we can also calculate upper-bound estimates on the distance, something that has not been exploited in previous work [31, 53]. The `MAXDIST` operator is defined as:

$$MAXDIST(Q,P) = \sqrt{\sum_{2 \le k \le d} Q \bigcap_t R \times x_d^2}$$

$$where \quad x_d = \begin{cases} max(h_{Q,d}, h_{P,d}) - min(l_{Q,d}, l_{Q,d}) \\ \quad if \quad (h_{Q,d} > h_{P,d} \quad \& \quad l_{Q,d} > l_{P,d}) \\ \quad or \quad (h_{P,d} > h_{Q,d} \quad \& \quad l_{P,d} > l_{Q,d}) \\ max \begin{cases} max(h_{Q,d}, h_{P,d}) - max(l_{Q,d}, l_{Q,d}) \\ min(h_{Q,d}, h_{P,d}) - min(l_{Q,d}, l_{Q,d}) \end{cases} \\ \quad otherwise \end{cases}$$

Examples of the operator for various MBR positions are shown in Figure 14.

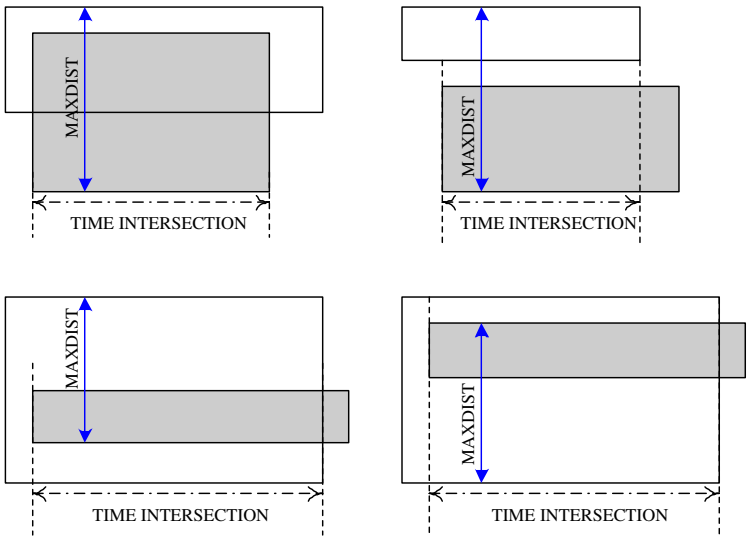

Figure 14: Operator `MAXDIST` between two MBRs.

Sequences with lower-bounds larger than the smallest computed upper-bound can be pruned. With this additional pre-filtering step the number of actual executions of the exact DTW distance are significantly reduced. In our experiments we have observed an additional 10-15% speedup in the total execution time when we also utilized the upper bounds.

A visual representation of the lower and upper bounding for the DTW case is shown in Figure 15.

7 Possible Index Optimizations

In a very large database we expect a significant number of the query MBRs to intersect only with a small area of the indexed MBRs. Therefore, as an optimization, with each MBR index entry we could store the percentage of points contained in each of the four quadrants Q_i of Q (Figure 16). That is:

$$Q_{i,MBR(j)} = \frac{100 * |Q_i \; points|}{|MBR(j) \; points|}, for \; i = 1 \ldots 4$$

This information can be used to compute a more accurate estimate of the actual distance between the query and the trajectory for the intersecting MBRs. This will increase the size of each entry of

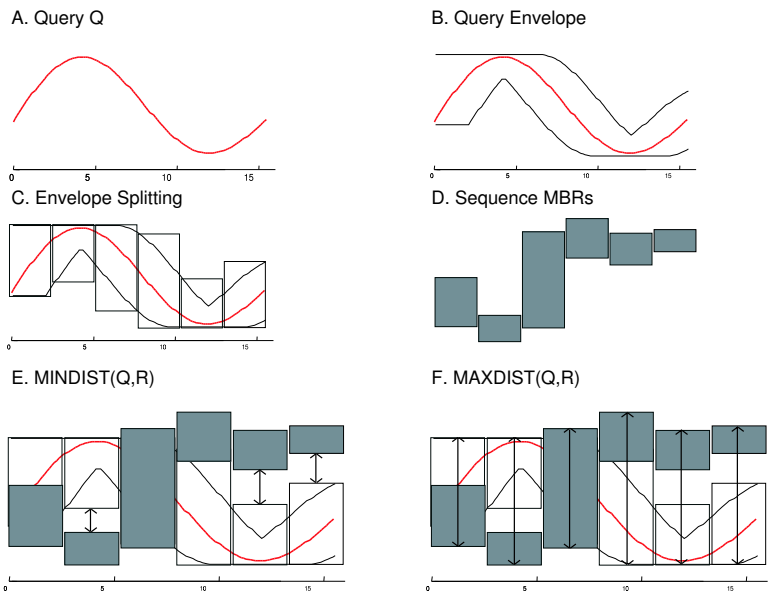

Figure 15: A visual intuition of the DTW indexing technique (the one-dimensional case is shown for clarity). The original query (A) is enclosed in a minimum-bounding envelope (B) like the LCSS approach. The MBE is split into its MBRs using Equi or Greedy-Split (C). The MBRs of candidate sequences are stored in the index (D). The minimum and maximum distance between the query and any sequence in the index can be quickly determined by examining the distances between the MBRs and the query bounding envelope, as represented by the arrows in (E) and (F).

the index nodes by 4 bytes (increasing also slightly the index size) and, thus, represents a middle ground in the case that we don't want to introduce more splits for the indexed trajectories (which will significantly increase the space consumption).

A second optimization justifies the separation of the *L-estimate* in two lists, and it could be used for producing approximate, but very good quality answers. It is based on the fact that the estimates stored in list $L_{t,partial}$, significantly overestimate the LCSS similarity. We depict this with an example: In Figure 17 we see the two different types of MBR intersections in time. For the intersection stored in list $L_{t,partial}$, even though there is only one point in the intersection of two MBRs, the similarity estimate will be increased by m which is the length of the intersection. However, for any intersection of length n in time, recorded in the list $L_{t,complete}$, we can be sure that there are n points which could *possibly* be matched.

Since the partial MBR intersections can be significantly misleading, if we would like to relax the accuracy of our method in favor of enhanced performance it is instructive to give a weight $0 < w_p < 1$ to all estimates in list $L_{t,partial}$. Although, now it is possible to miss the best match to the query, we are going to find a close match faster, because the estimates (even though not lower-bounding) will be tighter.

These optimizations were succinctly described here for the purpose of showing the great flexibility of our framework in balancing retrieval time and accuracy, with minimal or no adjustments in the index. In the experimental section we will only provide results on the approximate estimates based

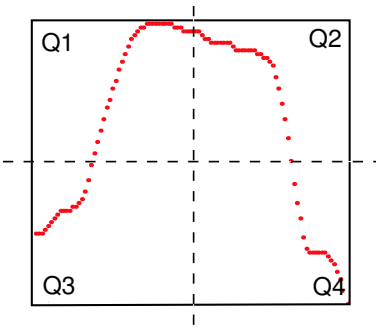

Figure 16: As an additional optimization one can store with every MBR the percentage of points for each quadrant (projected in two dimensions for clarity)

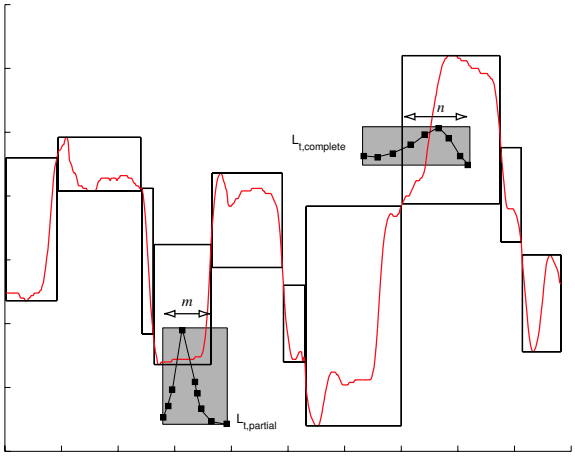

Figure 17: The estimate stored in list $L_{t,partial}$ can greatly overestimate the actual similarity. On the other hand, the estimate in $L_{t,complete}$ represents a more accurate match.

on the common volume intersection between the trajectory MBRs.

8 Supporting Multiple Measures

Here we will explain how the constructed index (based on the MBRs of the trajectories), can support without modification various distance measures, such as LCSS, DTW, as well as Euclidean distance.

The application of the *Minimum Bounding Envelope* only on the query suggests that user queries are not confined to a predefined and rigid matching window δ. The user can pose queries of variable warping in time. In some datasets, there is no need to perform warping, since the Euclidean distance performs well [32]. In other datasets, by using the Euclidean distance we can find quickly some very close matches, while using warping we can distinguish more flexible similarities. So, we can start by using a query with $\delta = 0$ (no bounding envelope), and increase it progressively in order to find more flexible matches (Figure 18).

Additionally, when the user desires to examine kNN similarity based on the DTW distance, it is instructive to pose a query with initial $\delta = 0$ for the following reasons:

1. For queries with constrained envelopes we expect to have faster index response times, since the query will intersect with fewer MBRs.

2. Since the computation of the Euclidean distance is cheap compared to DTW, by finding a good Euclidean match fast, we can prune dissimilar matches efficiently because we have discovered a tight match to the query.

Therefore, our framework offers the unique advantage that multiple distance functions can be supported in a single index. The indexed sequences have been segmented without any envelope applied on them and never have to be adjusted again. For different measures, the aspects that change are:

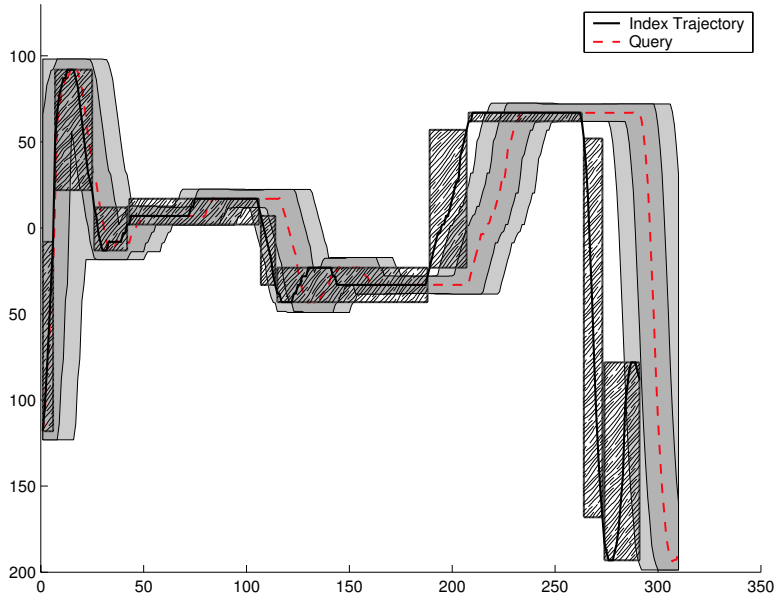

Figure 18: By incorporating the bounding envelope on the query our approach can support Euclidean distance, constrained warping, or full warping. This is accomplished by progressively expanding the MBE.

1. The creation of the query envelope (*splitting with or without envelope*)

2. The type of operation between query and indexed MBRs (*intersection or distance*)

For example, in order to pose queries based on Euclidean distance we follow these steps:

■ The query is segmented with no envelope applied on it.

■ The `MINDIST` and `MAXDIST` operators between MBRs for the Euclidean distance, are identical to the ones for DTW.

9 Experimental Evaluation

In this section we compare the effectiveness of the various MBR generation methods and we demonstrate the superiority of our lower-bounding technique for the DTW compared to other proposed lower-bounds. We describe the datasets we used and present comprehensive experiments regarding the index performance for the two similarity estimates. In addition, we evaluate the accuracy of the approximate estimates. All experiments conducted were run on an AMD Athlon 1.4 Ghz with 1GB RAM and 60GB of hard drive.

9.1 MBR Generation Comparison

The purpose of our first experiment is to test the space consumption of the proposed MBR generation methods. We have used eight datasets with diverse characteristics in order to provide objective results (Figure 19). The majority of the datasets are real world trajectories, extracted from video-tracking applications, marine mammals migration patterns etc.

We evaluate the space consumption by calculating the "Average Volume Gain" ($AvgVolGain$), which is defined as the percentage of volume when using i MBRs, over the volume when using only

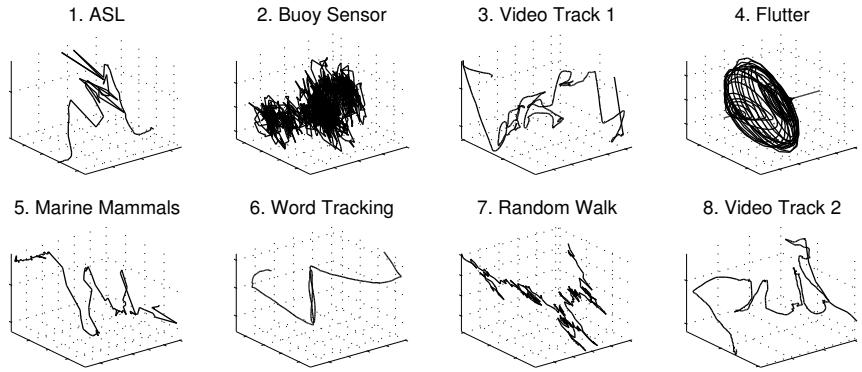

Figure 19: Datasets used for testing the efficiency of various MBR generation methods.

one MBR, normalized by the maximum gain provided over all methods. The result is averaged over the total number of experiments ($|splitExp|$) run with different number of splits [2] for each method. This number is usually 5, describing 20,40,60,80 and 100 splits. Specifically, the average gain $AvgGain_i$ for splitting method $i = equi, greedy, random$ is defined as:

$$AvgGain_i = \frac{1}{|splitExp|} * \sum_{split} \frac{vGain_{i,split}}{max_j(vGain_{j,split})}$$

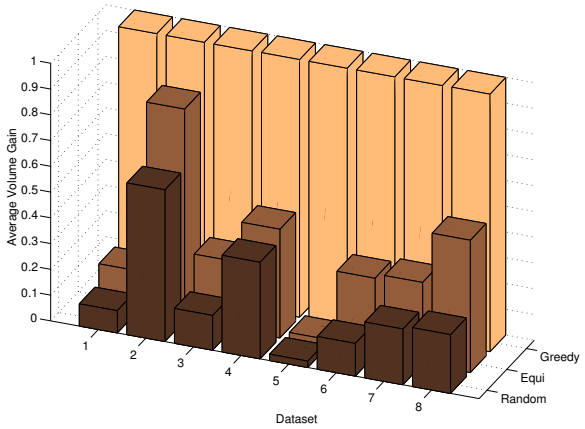

Figure 20: The Greedy-Split MBR generation algorithm presents the highest volume gain by producing MBRs that consume consistently less space over a number of datasets and for diverse number of generated MBRs

$AvgVolGain$ is a number between 0 and 1, where higher numbers indicate increased volume gain (or less space consumption) against the competitive methods. For the random split we run each experiment 100 times and we averaged the results. In Figure 20 we plot the average volume gain for all eight datasets. The Greedy-Split algorithm produced MBRs that consume at least half the space of Equi-Split. Equi-Split offers slightly better results than producing MBRs at random.

[2] By splits we mean the number of MBRs that approximate the trajectory

DATASET	$EQ_{s20,d5}$	$GR_{s20,d5}$	$EQ_{s40,d5}$	$GR_{s40,d5}$	$EQ_{s20,d5}$	$GR_{s20,d5}$	$EQ_{s40,d5}$	$GR_{s40,d5}$	LB-Kim	LB-Yi
	LCSS				**DTW**					
ASL	0.732	0.799	0.825	**0.856**	0.449	0.632	0.588	**0.756**	0.1873	0.2530
VT1	0.260	0.339	0.453	**0.511**	0.087	0.136	0.230	**0.266**	0.0838	0.1692
Marine	0.719	0.750	0.804	**0.814**	0.226	0.506	0.308	**0.608**	0.2587	0.4251
Word	0.627	0.666	0.761	**0.774**	0.311	0.361	0.466	**0.499**	0.0316	0.2116
Random	0.596	0.652	0.701	**0.741**	0.322	0.384	0.440	**0.491**	0.1389	0.2067
VT2	0.341	0.431	0.498	**0.569**	0.210	0.296	0.363	0.437	0.2100	**0.5321**

Table 1: Some indicative results of how close our similarity estimates are to the exact value (for 20 and 40 splits & $\delta = 5\%$). For all datasets the Greedy-Split approach provides the closest similarity estimates to the actual similarity.

The volume gain of Greedy-Split was smaller only for the *buoy sensor*, which is a very busy and unstructured signal. This experiment validates that our choice to use the Greedy-Split method was correct. Since the indexed MBR trajectories will take less space, we also expect tighter similarity estimates, therefore fewer false positives.

9.2 Tightness of Bounds

In table 1 we show how close our similarity estimates are (for LCSS and DTW) to the actual similarity between sequences. Numbers closer to 1 indicate higher similarity to the value returned by the exact algorithm. To our best knowledge, this paper introduces the first upper-bounding technique for LCSS, that is why we only report results for the equi-split and greedy-split method. For the Dynamic Time Warping we compare our lower bounds with LB-Kim and LB-Yi. These lower-bounds originally referred to 1-dimensional time-series but here we extended them for higher dimensions, in order to provide unambiguous results about the tightness of our estimates. Note that the previously proposed methods operate on the raw data. Our approach can still provide tighter estimates, while operating only on the trajectory MBRs. Using the raw data our experiments indicate that we are consistently 2-3 times better than the best alternative. However, since our index operates on the segmented trajectories we only report the results on the MBRs.

The results clearly indicate that the Greedy-Split method approximates the similarity consistently tighter than the Equi-Split. In table 1 only the results for $\delta = 5\%$ of the query's length are reported, but similar results are observed for increasing values of δ. It is evident from the table that using our method we can provide very tight lower bounds of the actual distance.

9.3 Index performance

We tested the performance of our index for LCSS using the upper-bound and the approximate similarity estimates, and compared it to the sequential scan. We also report results for DTW using the proposed lower and upper-bounds for pruning. The performance measure used is the total computation time required for the index and the sequential scan to return the nearest neighbor for the same one hundred queries. For the linear scan, one can also perform early termination of the LCSS (or the DTW) computation. Therefore, the LCSS execution can be stopped at the point where one is sure that the current sequence will not be more similar to the query than the

bestSoFar. We call this *optimistic* linear scan. *Pessimistic* linear scan is the one than does not reuse the previously computed similarity values and can be an accurate time estimate when the query match resides at the end of the dataset. We demonstrate the index performance relative to both types of linear scan, because this provides a realistic upper or lower-bound on the index speedup.

9.3.1 Dataset generation

In order to test the index scalability we needed to construct large realistic multi-dimensional datasets. To this end, we utilized the aggregation of our eight real datasets as seeds for generating more variations of them. We create multiple copies of the original trajectories by incorporating the following features:

- Addition of small variations in the original trajectory pattern
- Addition of random compression and decompression in time

Had we added random Gaussian noise to the original trajectory, this would have resulted to unnatural time series with large peaks at various positions. Therefore, the small variations in the pattern were added by interpolating *peaks* of Gaussian noise using splines [15]. In this manner we are able to create the smooth variations that existed in the original datasets (Figure 21).

The final dataset consisted of $2^{10} \ldots 2^{16}$ trajectories. Taking under consideration that the average trajectory size is around 500 points, this resulted to a database with more than 16 million 2-dimensional points. The trajectories have been *normalized* by subtracting the average value in each direction of movement.

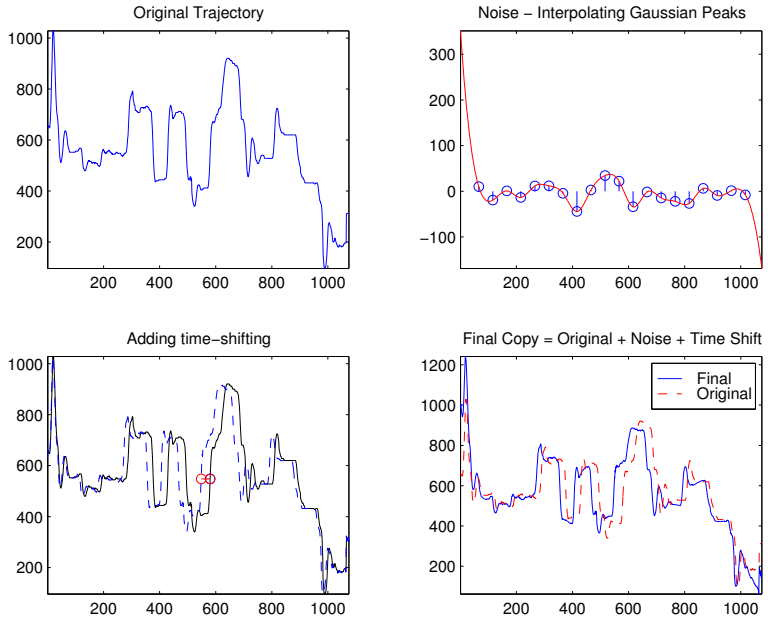

Figure 21: Creation of the multiple copies from the real datasets by adding smooth variations in the pattern and time shifting at random positions.

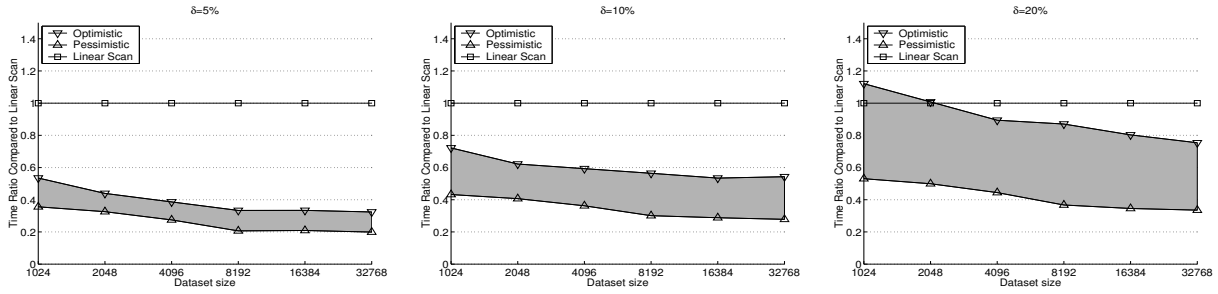

Figure 22: Index performance for LCSS. For small warping windows the index can be up to 5 times faster than sequential scan without compromising accuracy. The gray regions indicate the range of potential speedup.

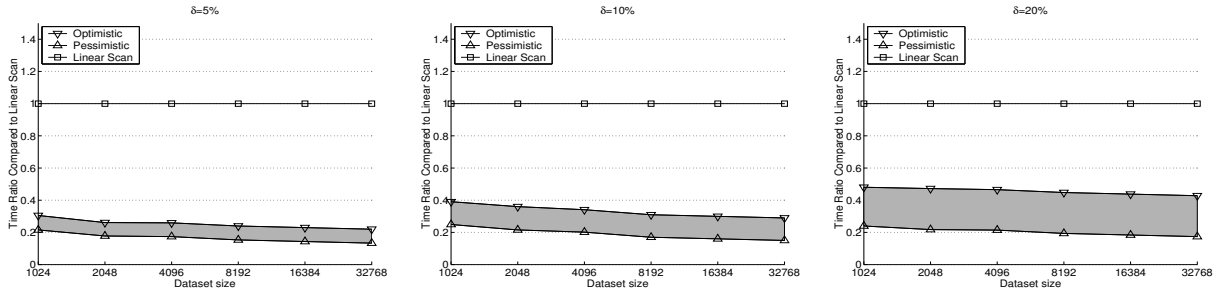

Figure 23: Using the approximate similarity estimates for the LCSS, the response time can be more than 7 times faster.

9.3.2 Results on the LCSS upper-bound estimates

The index performance is influenced be three parameters: the size of the dataset, the warping length δ (as a percentage of the query's length) and the number of trajectory MBRs. For all experiments the parameter ϵ (matching in space) was set to $std/4$ of the query, which provided good and intuitive results.

■ **Dataset size**: In figure 22 we can observe how the performance of the index scales with the database size (for various lengths of matching window). We record the index response time relative to both *optimistic* and *pessimistic* linear scan. Therefore, the gray region in the figures indicates the range of possible speedup. It is evident that the early termination feature of the sequential scan can significantly assist its performance. The usefulness of an index becomes obvious for large dataset sizes, where the quadratic computational cost dominates the I/O cost of the index. For these cases our approach can be up to 5 times faster than linear scan. In figure 24 we also demonstrate the pruning power of the index as a true indicator (not biased by any implementation details) about the efficacy of our index. Using the index we perform 2-5 times fewer LCSS computations than the linear scan. We observe similar speedup when using the DTW as the distance function in Figure 26.

■ **Parameter** δ: The index performance is better for smaller warping lengths. The experiments

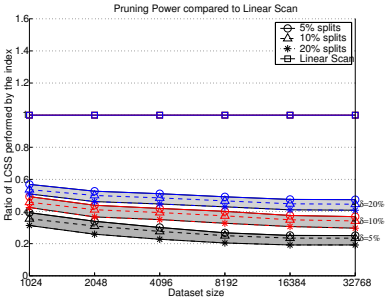

Figure 24: Each gray band indicates (for a certain warping window δ) the percentage of LCSS computations conducted by the index compared to linear scan.

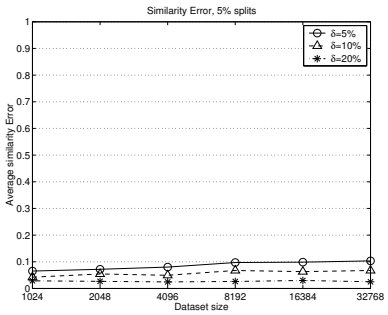

Figure 25: Using the V-similarity estimate we can retrieve answers faster with very high accuracy. The LCSS similarity is very close (2-10%) to the exact answer returned by the sequential scan.

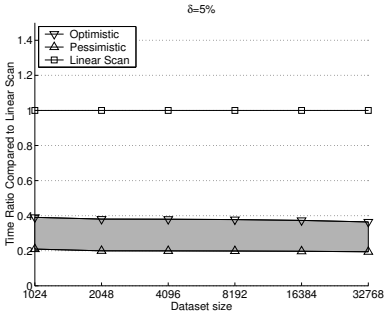

Figure 26: Index Performance using DTW as the distance measure. ($\delta = 5\%$). We can observe up to 5 times speedup.

record the performance for warping from 5% to 20% of the query's length. Increasing δ values signify larger bounding envelopes around the query, therefore larger space of search and less accurate similarity estimates. The graphs suggest that an index cannot be useful under full warping (when the data are normalized).

■ **Number of Splits**: Although greater number of MBRs for each trajectory implies better volume utilization, nonetheless more MBRs also lead to increased I/O cost. When we are referring to x% splits, it means that we have assigned a total of $100/x(\sum_{i=1}^{n}(||T_i||))$ splits, for all sequences T_i. In our figures we provide the 5% splits scenario for the MBRs, which offers better performance than 10% and 20% splits, since for the last two cases the I/O cost negates the effect of the better query approximation. The index space requirements for 5% splits is less than a quarter of the dataset size (Figure 27).

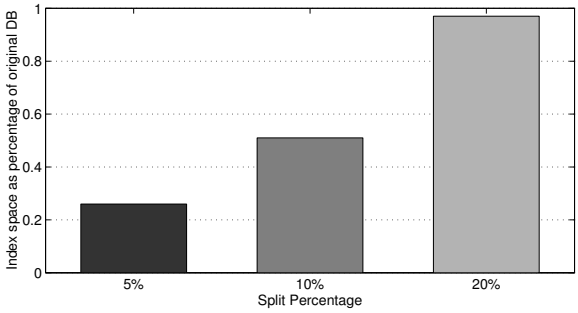

Figure 27: Index size compared to dataset size.

9.3.3 Results on the LCSS approximate estimates

Here we present the index performance when the volume intersections of the MBRs are used as estimates of the similarity and the results are shown in Figure 23. We observe that using this approximate similarity estimate our index performance is boosted significantly. The use of the *V-similarity* estimate leads to more tight approximations of the original similarity compared to the L-similarity estimate, however now we may miss the best match.

Naturally, comes the question of the quality of the results. We capture this by calculating the absolute difference between the similarity of the best match returned by the index and the best match found by the sequential scan for each query. Then, we average the results over a number of queries $|q|$. Therefore, the *Average Similarity Error* (ASE) is:

$$ASE = \frac{1}{|q|}\sum_{i=1}^{|q|}(|BestMatch_{index} - BestMatch_{exhaustive}|)$$

The results are shown in Figure 25. We can see that the similarity returned by the V-similarity estimate is approximately within 5% of the actual similarity (5% splits used).

By providing two similarity estimates for the LCSS, the user can decide the trade-off between expedited execution time and the quality of results. Since by using the latter estimator we can significantly increase the performance of the index, this is the approach we recommend for mining large datasets.

10 Test Cases

We present results on some real world applications, utilizing the proposed similarity measures and indexing scheme.

10.1 Automatic Transcription of Hand-written Manuscripts

The Library of Congress contains more than 54 million manuscripts and there is an imperative need for preserving these historical documents (Figure 28). Storing these manuscripts in image format is unrealistic, since only the conversion to text would results in more than 20 Terabytes of data. Therefore, there is an increasing interest to automatically transcribe these documents. However, optical character recognition techniques (OCR) cannot be fruitfully utilized in this case due to the handwritten nature of these documents and due to the degradations of the paper medium.

Recently, new techniques have been proposed that combine trends from pattern recognition and time-series similarity in order to achieve this difficult task [45]. The basic idea is that one can first identify the different words in a manuscript, manually annotate a subset of them, and then automatically classify the remaining words. The process involves the following steps: (i) From a manuscript image, images of the different words are extracted (word spotting); (ii) Writing variations (slant/skew/baseline) are corrected for the extracted words; (iii) Features for each image are extracted. Such features can include the sum of intensity values per column, or the ink/paper

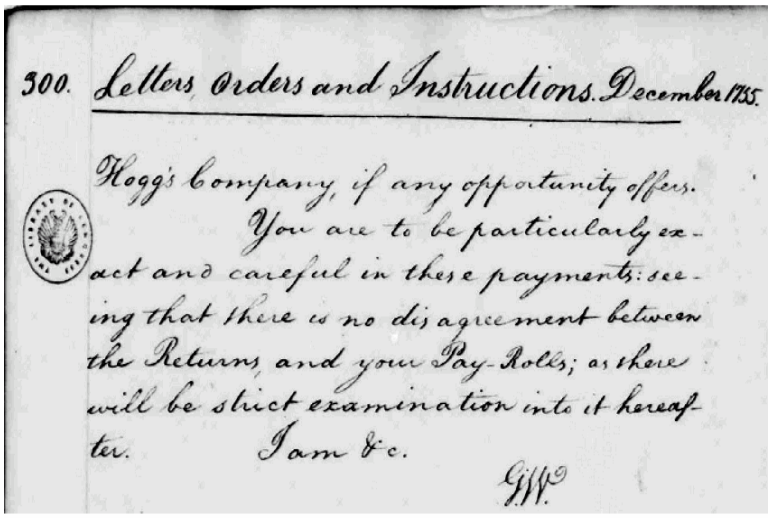

Figure 28: Part of a George Washington manuscript in the Library of Congress.

transitions per column etc. (Figure 29); (iv) Manual annotations of a word subset are produced; (v) Classification of the remaining words based on the annotated ones is performed.

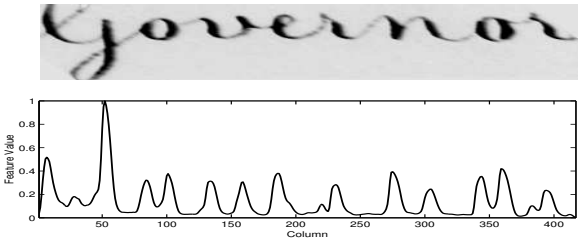

Figure 29: Top: The word 'governor' extracted from a George Washington manuscript. Bottom: One possible time-series feature extracted from every column of the image (sum of intensity values).

The features that are extracted are stored as multi-dimensional time-series. The annotation process can then use the multi-dimensional extensions of DTW or LCSS measures. In Figure 30 we see the results of a 3NN search for a variety of words, produced using the techniques analysed in this work. Next to each image we also depict the feature trajectory that described each word. For this experiment the two least correlated time-series features were utilized (out of the four that we had at our disposal).

These results used the DTW as the distance function, but we obtained similar results with the LCSS. We can observe that the outcome is very promising and even similarly looking words can be sufficiently discriminated and classified.

10.2 Similarity Search in Motion-Capture Data

The popularity of motion capture, or mocap, in CG movies and 3-dimensional computer games has motivated the development of many techniques to tackle the laborious problem of motion search and motion editing. The existence of large libraries of human motion capture has resulted in a

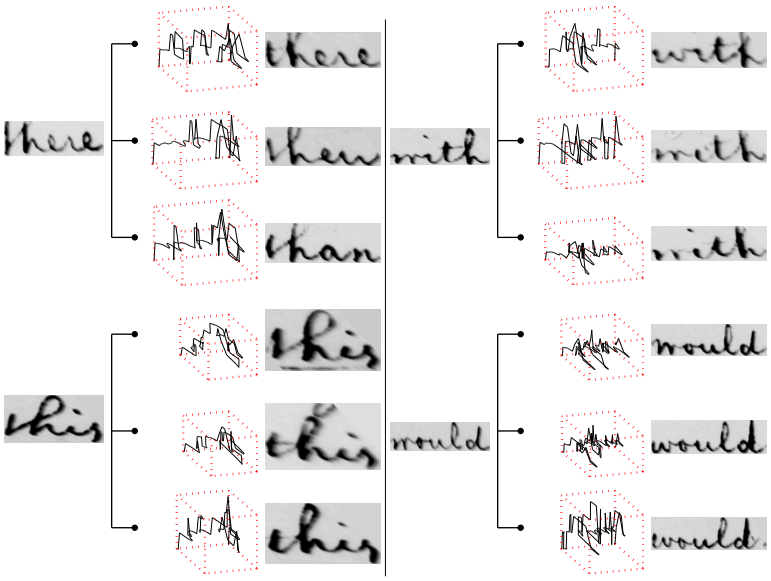

Figure 30: Manuscript word annotation. 3NN matches based on the extracted time-series features and utilizing multi-dimensional DTW as the distance measure.

growing demand for content-based retrieval of motion sequences without using annotations or other meta-data. Using the notions described in this paper, one can address efficiently the issue of rapidly retrieving perceptually similar occurrences of a particular motion in a long mocap sequence or unstructured mocap database for the purpose of replicating editing operations with minimal user-input. One or more editing operations on a given motion are made to affect all similar matching motions (for example change all walking motions into running motions etc.).

The first step of motion-editing consists of a similarity search portion, where using a query-by-example paradigm the animator first selects a particular motion, by specifying its start and end time, and the system searches for similar occurrences in a mocap database (Figure 31). For maximum usability, the mocap matching engine must provide fast response to user queries over extended unlabeled mocap sequences, whilst allowing for spatial and temporal deviations in the returned matches.

Using the techniques described in this paper, each motion is split up in multi-dimensional MBRs and stored in an index structure [11]. Support for LCSS and DTW similarity/distance measures caters for noisy motion with variations in the time axis. Therefore one can find matches independent of the speed at which the mocap data were recorded.

The animator has the crucial ability to interactively select the body areas utilized in the matching, so that, for example, all instances of a walking motion are returned, irrespective of the upper body motion. The results of such a query are shown in Figure 32. Finally, in the case where many potential matches exist, the query results can be clustered and presented in a more organized manner, allowing the animator to rapidly dismiss undesirable classes of matches.

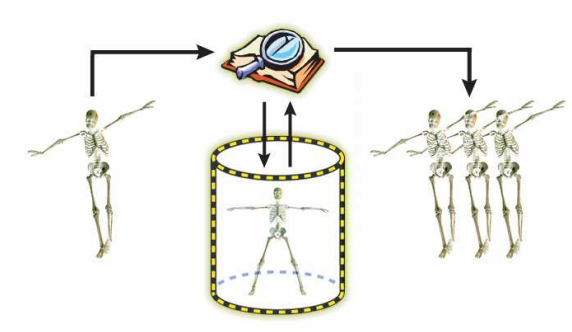

Figure 31: Matches to the user query motion are returned using an efficient search index, previously extracted from the motion database.

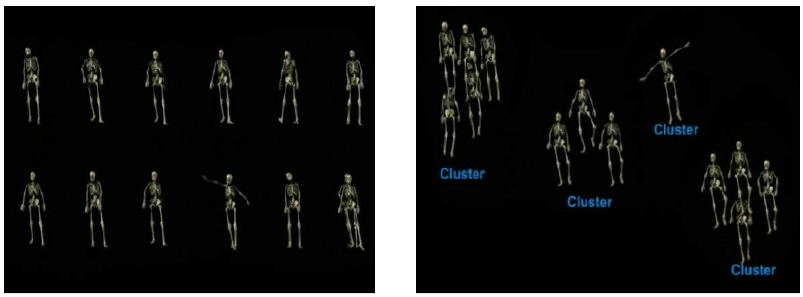

Figure 32: Some matching results in the motion-capture database for the query: "Find all walking motions". The final results can be grouped into similar motions using a hierarchical clustering algorithm and presented to the animator in a more meaningful manner.

11 Conclusions

In this paper we have presented an external memory indexing method for discovering similar multi-dimensional trajectories. The unique advantage of our approach is that it can accommodate multiple distance measures. The method guarantees no false dismissals and presents a significant execution speedup for LCSS and DTW compared to sequential scan. We have shown the tightness of our similarity estimates and demonstrated the usefulness of our measures in real world applications. We hope that our effort can act as a bridge between metric and non-metric functions, as well as a tool for understanding better their strengths and weaknesses.

Acknowledgements: We would like to thank Margrit Betke for providing us the *Video Track I and II* datasets, and Marc Cardle for providing his data and expertise on motion-capture. We are also obliged to T. Rath and R. Manmatha for kindly providing the manuscript words dataset.

References

[1] J. Aach and G. Church. Aligning gene expression time series with time warping algorithms. In *Bioinformatics, Volume 17*, pages 495–508, 2001.

[2] R. Agrawal, C. Faloutsos, and A. Swami. Efficient Similarity Search in Sequence Databases. *In Proc. of the 4th FODO*, pages 69–84, October 1993.

[3] R. Agrawal, K. Lin, H. S. Sawhney, and K. Shim. Fast Similarity Search in the Presence of Noise, Scaling and Translation in Time-Series Databases. *In Proc of VLDB*, pages 490–501, September 1995.

[4] O. Arikan and D. Forsyth. Interactive motion generation from examples. In *Proc. of ACM SIGGRAPH*, 2002.

[5] Z. Bar-Joseph, G. Gerber, D. Gifford, T. Jaakkola, and I. Simon. A new approach to analyzing gene expression time series data. In *Proc. of 6th RECOMB*, pages 39–48, 2002.

[6] D. Barbara. Mobile computing and databases - a survey. *IEEE TKDE*, pages 108–117, January 1999.

[7] D. Berndt and J. Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. *In Proc. of KDD Workshop*, 1994.

[8] M. Betke, J. Gips, and P. Fleming. The camera mouse: Visual tracking of body features to provide computer access for people with severe disabilities. In *IEEE Transactions on Neural Systems and Rehabilitation Engineering, Vol. 10, No. 1*, 2002.

[9] B. Bollobas, G. Das, D. Gunopulos, and H. Mannila. Time-Series Similarity Problems and Well-Separated Geometric Sets. *In Proc of the 13th SCG, Nice, France*, 1997.

[10] T. Bozkaya, N. Yazdani, and M. Ozsoyoglu. Matching and Indexing Sequences of Different Lengths. *Proc of the CIKM, Las Vegas*, 1997.

[11] M. Cardle, M. Vlachos, S. Brooks, E. Keogh, and D. Gunopulos. Fast motion capture matching with replicated motion editing. In *29th ACM SIGGRAPH, Sketches and Applications*, 2003.

[12] K. Chu and M. Wong. Fast Time-Series Searching with Scaling and Shifting. *ACM Principles of Database Systems*, pages 237–248, June 1999.

[13] Darya Chudova, Scott Gaffney, Eric Mjolsness, and Padhraic Smyth. Translation-invariant mixture models for curve clustering. In *Proc. of 9th SIGKDD*, 2003.

[14] G. Das, D. Gunopulos, and H. Mannila. Finding Similar Time Series. *In Proc. of the First PKDD Symp.*, pages 88–100, 1997.

[15] de Boor C. *A Practical Guide to Splines*. Springer-Verlag, 1978.

[16] C. Faloutsos, H.V. Jagadish, A. Mendelzon, and T. Milo. Signature technique for similarity-based queries. *In SEQUENCES 97*, 1997.

[17] C. Faloutsos and K.-I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *In Proc. ACM SIGMOD*, pages 163–174, May 1995.

[18] C. Faloutsos, M. Ranganathan, and I. Manolopoulos. Fast Subsequence Matching in Time Series Databases. *In Proceedings of ACM SIGMOD*, pages 419–429, May 1994.

[19] S. Gaffney and P. Smyth. Trajectory Clustering with Mixtures of Regression Models. *In Proc. of the 5th ACM SIGKDD, San Diego, CA*, pages 63–72, August 1999.

[20] D.M. Gavrila and L.S. Davis. Towards 3-d model-based tracking and recognition of human movement: a multi-view approach. In *Int. Workshop on Face and Gesture Recognition*, 1995.

[21] X. Ge and P. Smyth. Deformable markov model templates for time-series pattern matching. *In Proc ACM SIGKDD*, 2000.

[22] D. Goldin and P. Kanellakis. On Similarity Queries for Time-Series Data. *In Proceedings of CP '95, Cassis, France*, September 1995.

[23] K. Gollmer and C. Posten. Detection of distorted pattern using dynamic time warping algorithm and application for supervision of bioprocesses. In *On-Line Fault Detection and Supervision in Chemical Process Industries*, 1995.

[24] S. Grumbach, P. Rigaux, and Luc Segoufin. Manipulating interpolated data is easier than you thought. *In Proc of the 26th VLDB*, pages 156–165, 2000.

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

[26] Marios Hadjieleftheriou, George Kollios, Vassilis Tsotras, and Dimitrios Gunopulos. Efficient indexing of spatiotemporal objects. In *Proc. of 8th EDBT*, 2002.

[27] H. V. Jagadish, Alberto O. Mendelzon, and Tova Milo. Similarity-based queries. *In Proc. of the 14th ACM PODS*, pages 36–45, May 1995.

[28] T. Kahveci and A. K. Singh. Variable length queries for time series data. In *Proc. of IEEE ICDE*, pages 273–282, 2001.

[29] Tamer Kahveci, Ambuj Singh, and Aliekber Gurel. Similarity searching for multi-attribute sequences. In *Proc. of SSDBM*, 2002.

[30] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. of ACM SIGMOD*, pages 151–162, 2001.

[31] Eamonn Keogh. Exact indexing of dynamic time warping. In *Proc. of VLDB*, 2002.

[32] Eamonn Keogh and Shruti Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. In *Proc. of SIGKDD*, 2002.

[33] S. Kim, S. Park, and W. Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In *Proc. of 17th ICDE*, 2001.

[34] Z.M. Kovács-Vajna. A fingerprint verification system based on triangular matching and dynamic time warping. In *IEEE Transactions on PAMI, Vol. 22, No. 11*, 2000.

[35] S.-L. Lee, S.-J. Chun, D.-H. Kim, J.-H. Lee, and C.-W. Chung. Similarity Search for Multidimensional Data Sequences. *Proc. of ICDE*, pages 599–608, 2000.

[36] V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics - Doklady 10, 10*, pages 707–710, 1966.

[37] M. Munich and P. Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *7th International Conference on Computer Vision*, pages 108–115, 1999.

[38] S. Park, W. Chu, J. Yoon, and C. Hsu. Efficient Searches for Similar Subsequences of Different Lengths in Sequence Databases. *In Proc. of ICDE*, pages 23–32, 2000.

[39] S. Perng, H. Wang, S. Zhang, and D. S. Parker. Landmarks: a New Model for Similarity-based Pattern Querying in Time Series Databases. *In Proceedings of ICDE*, pages 33–42, 2000.

[40] D. Pfoser and C. S. Jensen. Capturing the uncertainty of moving-object representations. *Lecture Notes in Computer Science*, 1651, 1999.

[41] Kevin B. Pratt. Locating patterns in discrete time-series. Master's thesis, 2001.

[42] Y. Qu, C. Wang, and X.S. Wang. Supporting Fast Search in Time Series for Movement Patterns in Multiple Scales. In *Proc of the ACM CIKM*, pages 251–258, 1998.

[43] D. Rafiei and A. Mendelzon. Querying Time Series Data Based on Similarity. *IEEE Transactions on Knowledge and Data Engineering, Vol. 12, No 5.*, pages 675–693, 2000.

[44] Chotirat Ann Ratanamahatana and Eamonn Keogh. Making time-series classification more accurate using learned constraints. In *Proc. of SIAM International Conference on Data Mining (SDM),*, 2004.

[45] T. Rath and R. Manmatha. Word image matching using dynamic time warping. In *Tec Report MM-38. Center for Intelligent Information Retrieval, University of Massachusetts Amherst*, 2002.

[46] J. F. Roddick and K. Hornsby. *Temporal, Spatial and Spatio-Temporal Data Mining.* 2000.

[47] Mitsuomi Shimada and Kuniaki Uehara. Discovery of correlation from multi-stream of human motion. In *Discovery Science 2000.*

[48] H. Strik and L. Boves. Averaging physiological signals with the use of a dtw algorithm. In *SPEECH'88, 7th FASE Symposium, Book 3*, pages 883–890, 1988.

[49] R. E. Valdes-Perez and C. A. Stone. Systematic detection of subtle spatio-temporal patterns in time-lapse imaging ii. particle migrations. In *Bioimaging 6(2)*, pages 71–78, 1998.

[50] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In *Proc. of ICDE*, 2002.

[51] B.-K. Yi and C. Faloutsos. Fast Time Sequence Indexing for Arbitrary Lp Norms. *In Proceedings of VLDB, Cairo Egypt*, September 2000.

[52] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. *In Proc. of ICDE*, pages 201–208, 1998.

[53] Yunyue Zhu and Dennis Shasha. Query by humming: a time series database approach. In *Proc. of SIGMOD*, 2003.