Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures

Michail Vlachos   Marios Hadjieleftheriou   Dimitrios Gunopulos [†]   Eamonn Keogh

UC Riverside, {mvlachos, marioh, dg, eamonn}@cs.ucr.edu

ABSTRACT

Although 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, that 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 speed-up the computation of expensive similarity measures such as the LCSS and the DTW.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications, *Data Mining*

Keywords: Trajectories, Longest Common Subsequence, Dynamic Time Warping

1. INTRODUCTION

In this work we present an efficient and compact, external memory index for fast detection of similar trajectories. Trajectory data are prevalent in diverse fields of interest such as meteorology, GPS tracking, wireless applications, video tracking [5] and motion capture [18]. Recent advances in mobile computing, sensor and GPS technology have made it possible to collect large amounts of spatiotemporal data and there is increasing interest in performing data analysis tasks over such data [17]. In mobile computing, users equipped with mobile devices move in space and register their location at different time instances to spatiotemporal databases via wireless links. In environmental information systems, tracking animals and weather conditions is very common and large datasets can be created by storing locations of observed objects over time. Human motion data generated by tracking simultaneously various body joints are also multidimensional trajectories. In this field of computer graphics fundamental operations include the clustering of similar movements, leading to a multitude of applications such as interactive generation of motions [2]. Spatiotemporal data are also produced by migrating particles in biological sciences, where the focus can be on the discovery of subtle patterns during cellular mitoses [19]. In general, any dataset that involves storage of multiple streams (attributes) of data can be considered and treated as a multidimensional trajectory.

One very common task for such data is the discovery of objects that follow a certain motion pattern, for purposes of clustering or classification. The objective here is to efficiently organize trajectories on disk, so that we can quickly answer k-Nearest-Neighbors (kNN) queries. A frequent obstacle in the analysis of spatiotemporal data, is the presence of noise, which can be introduced due to electromagnetic anomalies, transceiver problems etc. Another impediment is that objects may move in a similar way, but at different speeds. So, we would like our similarity model to be robust to noise, support elastic and imprecise matches.

Choosing the Euclidean distance as the similarity model is unrealistic, since its performance degrades rapidly in the presence of noise and this measure is also sensitive to small variations in the time axis. We concentrate on two similarity models: the first is an extension of Dynamic Time Warping for higher dimensions. We note that DTW has been used so far for one-dimensional time series. Here we present a formulation for sequences of arbitrary dimensions. The second distance measure is a modification of the Longest Common Subsequence (LCSS), specially adapted for continuous values. Both measures represent a significant improvement compared to the Euclidean distance. However, LCSS is more robust than DTW under noisy conditions [20] as figure 1 shows. Euclidean matching completely disregards the variations in the time axis, while DTW performs excessive matchings, therefore distorting the true distance between sequences. The LCSS produces the most robust and intuitive correspondence between points.

By incorporating warping in time as a requirement to

[†] The research of this author was supported by NSF ITR 0220148, NSF CAREER 9907477, NSF IIS 9984729, and NRDRP

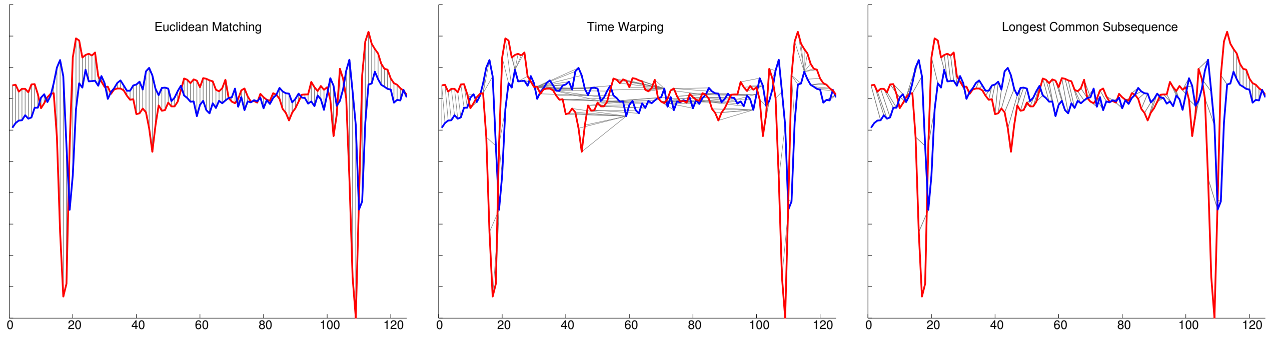

Figure 1: A lucid example about the quality matching of the LCSS compared to other distance functions. The Euclidean distance performs an inflexible matching, while the DTW gives many superfluous and spurious matchings, in the presence of noise.

our model, our algorithms are automatically challenged with quadratic execution time. Moreover, these flexible functions are typically non-metric, which makes difficult the design of indexing structures. To speed up the execution of a similarity function, one can devise a low cost, *upper bounding* function (since the LCSS model captures the similarity, which is inversely analogous to the distance). We utilize a fast prefiltering scheme that will return upper bound estimates for the LCSS similarity between the query and the indexed trajectories. In addition to providing similarity measures that guarantee *no false dismissals*, we also propose *approximate* similarity estimates that significantly reduce the index response time. Finally, we show that the same index can support other distance measures as well.

Our technique works by splitting the trajectories in multidimensional MBRs and storing them in an R-tree. For a given query, we construct a *Minimum Bounding Envelope* (MBE) that covers all the possible matching areas of the query under warping conditions. This MBE is decomposed into MBRs and then probed in the R-tree index. Using the index we can discover which trajectories could potentially be similar to the query. The index size is compact and its construction time scales well with the trajectory length and the database size, therefore our method can be utilized for massive datamining tasks.

The main contributions of the paper are:

■ We present the first external memory index for multidimensional trajectories, that supports multiple 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 in the design of an efficient indexing structure for the LCSS and the DTW.

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

2. RELATED WORK

There has been a wealth of papers that use an L_p distance family function to perform similarity matching for 1D time-series. Work on multidimensional sequences can be found in [14, 9]. However, they support only Euclidean distance, which, as mentioned in the introduction, cannot capture flexible similarities.

Although the vast majority of database/data mining research on time series data mining has focused on Euclidean distance, virtually all real world systems that use time series matching as a subroutine, use a similarity measure which allows warping. In retrospect, this is not very surprising, since most real world processes, particularly biological 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. Because of this, DTW is used for gene expression data mining [1, 3]. Dynamic Time Warping is a ubiquitous tool in the biometric/surveillance community. It has been used for tracking time series extracted from video [7], classifying handwritten text [16] and even fingerprint indexing [13].

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 the large computational cost has appeared in [12] and [21], where the authors use lower bounding measures to speed up the execution of DTW. However, the lower bounds can be loose approximations of the original distance, when the data are normalized. In [15] a different approach is used for indexing Time Warping, by using suffix trees. Nonetheless, the index requires excessive disk space (about 10 times the size of the original data).

The flexibility provided by DTW is very important, however its efficiency deteriorates for noisy data, since by matching all the points, it also matches the outliers distorting the true distance between the sequences. An alternative approach is the use of *Longest Common Subsequence* ($LCSS$), which is a variation of the edit distance. The basic idea is to match two sequences by allowing them to stretch, without rearranging the order of the elements but allowing some elements to be *unmatched*. Using the $LCSS$ of two sequences, one can define the distance using the length of this subsequence [6]. In [20] an internal memory index for the $LCSS$ has been proposed. It also demonstrated that while the $LCSS$ presents similar advantages to DTW, it does not share its volatile performance in the presence of outliers.

Closest in spirit to our approach, is the work of [10] which, however, only addresses 1D time-series. The author uses constrained DTW as the distance function, and surrounds the possible matching regions by a modified version of a Piecewise Approximation, which is later stored as equi-length MBRs in an R-tree. However, by using DTW, such an approach is susceptible to high bias of outliers. Also, the fixed MBR size (although 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 simply computed by calculating the MBR intersection areas. Since the index is not constructed for a specific warping window, the user can pose queries with variable warping length.

The purpose of this paper is to reconcile the best of both worlds. We provide a framework that can support in the *same* index, the LCSS, DTW and Euclidean distance functions. The only aspect that changes, is the different representation of the query for each distance measure.

3. DISTANCE MEASURES

In this section we present details of how the Dynamic Time Warping and the LCSS model can be extended to describe the similarity between trajectories.

3.1 Dynamic Time Warping for 2D trajectories

We describe an extension in 2D of the original DTW function as described by Berndt and Clifford [4]. Let A and B be two trajectories of moving objects with size n and m respectively, where $A = ((a_{x,1}, a_{y,1}), \ldots, (a_{x,n}, a_{y,n}))$ and $B = ((b_{x,1}, b_{y,1}), \ldots, (b_{x,m}, b_{y,m}))$. For a trajectory A, let $Head(A) = ((a_{x,1}, a_{y,1}), \ldots, (a_{x,n-1}, a_{y,n-1}))$.

Definition 1. *The Time Warping between 2-dimensional sequences A and B is:*

$$\begin{aligned} DTW(A,B) &= L_p((a_{x,n}, a_{y,n}), (b_{x,m}, b_{y,m})) + \\ &\quad \boldsymbol{min}\{DTW(Head(A), \\ &\qquad Head(B)), DTW(Head(A), B), \\ &\qquad DTW(A, Head(B))\} \end{aligned} \qquad (1)$$

where L_p is any p-Norm. Using dynamic programming and constraining the matching region within δ, the time required to compute DTW is $O(\delta(n+m))$. In order to represent an accurate relationship of distances between sequences with different lengths, the quantity in equation 1 is normalized by the length of the warping path. The extension to n dimensions is similar. In figure 2 we show an example of time warping for two trajectories.

3.2 LCSS model for 2D trajectories

The original LCSS model refers to 1D sequences, we must therefore extend it to the 2D case. In addition, the LCSS paradigm matches discrete values, however in our model we want to allow a matching, when the values are within a certain range in space and time (note that like this, we also avoid distant and degenerate matchings).

Definition 2. *Given an integer δ and a real number $0 < \epsilon < 1$, we define the $LCSS_{\delta,\epsilon}(A,B)$ as follows:*

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

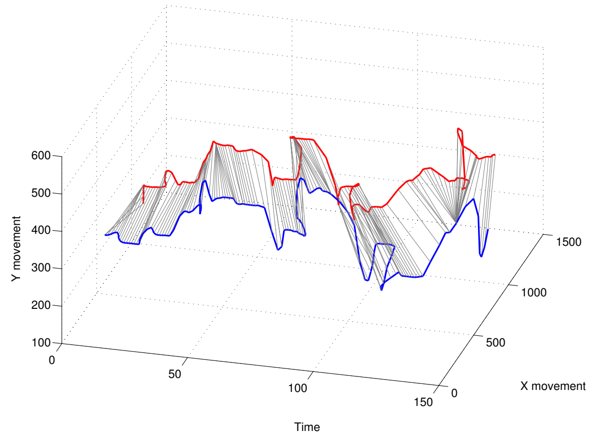

Figure 2: The support of flexible matching in spatiotemporal queries is very important. However, we can observe that Dynamic Time Warping matches all points (so the outliers as well), therefore distorting the true distance. In contrast, the LCSS model can efficiently ignore the noisy parts.

where sequences A and $Head(A)$ are defined similarly as before. The constant δ controls the flexibility of matching in time and constant ϵ is the matching threshold is space. The aforementioned $LCSS$ model has the same $O(\delta(n+m))$ computational complexity as the DTW, when we only allow a matching window δ in time [6].

The value of $LCSS$ is unbounded and depends on the length of the compared sequences. We need to normalize it, in order to support sequences of variable length. The distance derived from the LCSS similarity can be defined as follows:

Definition 3. *The distance $D_{\delta,\epsilon}$ expressed in terms of the LCSS similarity between two trajectories A and B is given by:*

$$D_{\delta,\epsilon}(A,B) = 1 - \frac{LCSS_{\delta,\epsilon}(A,B)}{\min(n,m)} \qquad (2)$$

4. INDEX CONSTRUCTION

Even though imposing a matching window δ can help speed up 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 our query. This can be accomplished by discovering a close match to our query, as early as possible. A fast pre-filtering step is employed that eliminates the majority of distant matches. Only for some qualified sequences will we execute the costly (but accurate) quadratic time algorithm. This philosophy has also been successfully used in [21, 10]. There are certain preprocessing steps that we follow:

1. The trajectories are segmented into MBRs, which are stored in an Rtree T.
2. Given a query Q, we discover the areas of possible matching by constructing its *Minimum Bounding Envelope* (MBE_Q).
3. MBE_Q is decomposed into MBRs that are probed in the index T.

4. Based on the MBR intersections, similarity estimates are computed and the exact LCSS (or DTW) is performed only on the qualified trajectories.

The above notions are illustrated in figure 3 and we explain in detail how they can be applied for the LCSS case in the sections that follow.

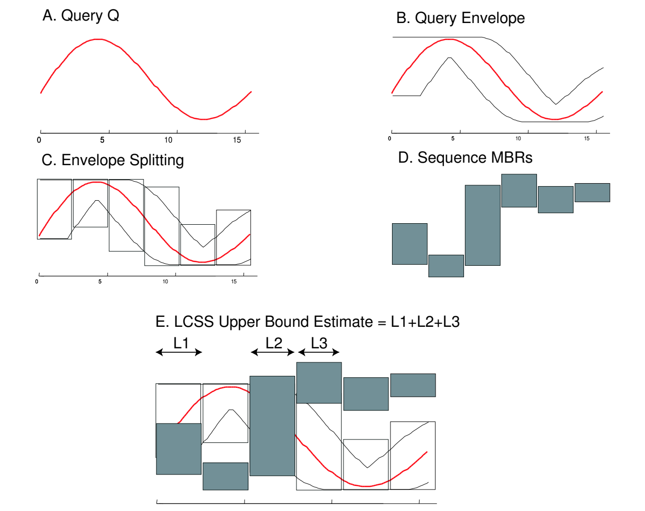

Figure 3: An example of our approach (in 1D for clarity); A query is extended into a bounding envelope, which in turn is also split into the resulting MBRs. Overlap between the query and the index MBRs suggest areas of possible matching.

4.1 Bounding the Matching Regions

Let us first consider a 1D time-series and let a sequence A be $(a_{x,1}, \dots, a_{x,n})$. Ignoring for now the parameter ϵ, we would like to perform a very fast $LCSS_\delta$ match between sequence A and some query Q. Suppore that we replicate each point Q_i for δ time instances before 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.

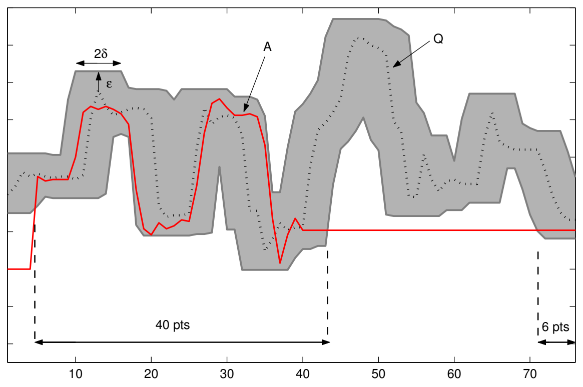

Figure 4: The *Minimum Bounding Envelope* (MBE) within δ in time and ϵ in space of a sequence. Everything that lies outside this envelope can never be matched.

We call this envelope, the *Minimum Bounding Envelope* (MBE) of a sequence. Also, once we incorporate the matching within ϵ in space, this envelope should extent ϵ above and below the original envelope (figure 4). The notion of the bounding envelope can be trivially extended in more dimensions, where $MBE(\delta, \epsilon)$ for a 2D trajectory $Q = ((q_{x,1}, q_{y,1}), \dots, (q_{x,n}, q_{y,n})$ covers the area between the following time-series:

$$EnvLow \leq MBE(\delta, \epsilon) \leq EnvHigh, \text{ where:}$$

$$\begin{cases} EnvHigh[i] = max(Q[j] + epsilon) & , |i - j| \leq \delta \\ EnvLow[j] = min(Q[j] - epsilon) & , |i - j| \leq \delta \end{cases}$$

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 4 the $LCSS$ similarity between MBE_Q and sequence A is 46, as indicated in the figure. This value represents an *upper bound* for the *similarity* of Q and A. We can use the MBE_Q to compute a *lower bound* on the *distance* between trajectories:

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. □

The previous lemma provides us with the power to create an index that guarantees no false dismissals. However, this lower bound refers to the raw data. In the sections that follow, we will 'split' the MBE of a trajectory, into a number of *Minimum Bounding Rectangles* (MBRs), to accommodate their storage into a multidimensional R-tree. We will show that the above inequality still holds between trajectory MBRs.

The MBR generation procedure is orthogonal to our approach, since any segmentation methodology can be applied to our framework. Therefore, the description of the potential MBR generation methods (and of our implementation choice) will be delayed until later.

5. 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 *upper bound* estimates of the similarity between the query and the indexed sequences using their MBRs.

Below we describe the algorithm to find the closest trajectory to a given query:

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

```
Box $Env = constructMBE_{\delta,\epsilon}(Q)$;
Vector $V_Q = CreateMBRs(Env)$;
  // $V_Q$ contains a number of boxes.
Priority queue $PQ \leftarrow \emptyset$;
  // $PQ$ keeps one entry per trajectory sorted
  // according to the similarity estimate

for each box $B$ in $V_Q$:
  $V$ = $I$.intersectionQuery($B$);
    // $V$ contains all trajectory MBRs that intersect with $B$.

  if Method == Exact: // upper bound
    $PQ \leftarrow$ computeL-SimilarityEstimates(V, B);
  else: // approximate
    $PQ \leftarrow$ computeV-SimilarityEstimates(V, B);

$BestSoFar = 0$; $Best \leftarrow \emptyset$;
while $PQ$ not empty:
  $E \leftarrow PQ.top$;
  if $E.estimate < BestSoFar$: break;
  else:
    $D = computeLCCS_{\delta,\epsilon}(Q, E)$; // exact
    if $D > BestSoFar$:
      $BestSoFar = D$; $Best \leftarrow E$;

Report $Best$;
```

The above algorithm can be adjusted to return the k-NN sequences, simply by comparing with the k^{th} $bestSoFar$ match. Next, we examine the possible similarity estimates. Some of them guarantee that will find the best match (they lower bound the original distance or upper bound the original similarity), while other estimates provide faster but approximate results.

5.1 Similarity Estimates

Here we will show how to compute estimates of the LCSS similarity, based on the geometric properties of the trajectory MBRs and their intersection. 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. Then we will use these operators to derive the estimates.

5.1.1 Estimates for the LCSS

Each trajectory T can be decomposed into a number of MBRs. The i_{th} 3D MBR of 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 3D MBRs $M_{P,i}$ and $M_{R,j}$, belonging to objects P and R, respectively:

1. $\bigcap_t^{(c)}(M_{P,i}, M_{R,j}) = ||Intersection||_t$,
   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

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

We can use upper bound or approximate estimates for the similarity:

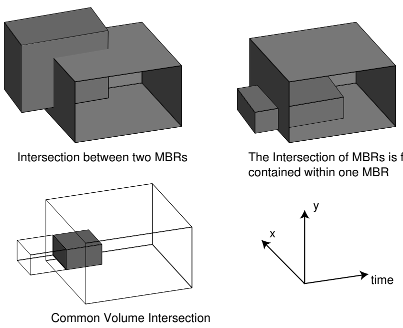
Intersection between two MBRs
The Intersection of MBRs is fully contained within one MBR
Common Volume Intersection
y
x
time

Figure 5: *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.

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 the total time that its MBRs intersected with the query's MBRs. We record into this list only the intersections, where a query MBR is fully contained in all spatial dimensions by a trajectory MBR (or vice versa -it is equivalent. See figure 5, top right).

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

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

This list records for each sequence the total intersection in time for those query MBRs that are not fully contained within the x,y dimensions by the trajectory MBRs (or vice versa. Figure 5, 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.

The reason for the distinction of the *L-similarity estimate* in two separate lists derives from the fact that the estimates stored in list $L_{t,partial}$ can significantly overestimate the LCSS similarity. If one wishes to relax the accuracy, in favor of enhanced performance, it is instructive to give a weight $0 < w_p < 1$ to all estimates in list $L_{t,partial}$. Even though now we *may* miss the best match to our query, we are going to find a close match in less time. This weighted approach is used when we are seeking for approximate, but very good quality answers, however it will not be explained further due to space limitations.

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 axis.

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 2. *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 to the original distance, we expect faster response time. Indeed, as we show in the experimental section, the index performance is boosted, while the error in similarity is frequently less then 5%.

5.2 Estimates for the DTW

When the distance function used is the Time Warping, using the index we obtain a *lower bound* of the actual distance. In this case we have the inverse situation from the LCSS; instead of calculating the degree of overlap between the MBRs of the indexed trajectories and the query, we evaluate the *distance* between the MBRs. The overall distance between the MBRs underestimates the true distance of the trajectories, and no false dismissals are introduced. Using the MBRs we can also calculate *upper bound* estimates on the distance, which hadn't been exploited in previous work [10, 22]. Sequences with lower bound larger than the smallest upper bound can be pruned. With this additional pre-filtering step we can gain on average an additional 10-15% speedup in the total execution time.

Due to space limitations only a visual representation of this approach is provided in figure 6.

6. MBR GENERATION

Given a multidimensional time-series (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. We can show the following lemma:

LEMMA 3. *Minimizing the volume of the Minimum Bounding Envelope, minimizes the expected similarity approximation error.*

Three different approaches are considered:

1. **k-Optimal**. We can discover the k MBRs of a sequence that take up the least volume, using a dynamic programming algorithm that requires $O(n^2k)$ time ([8]), where n is the length of the given sequence. Since this approach is not reasonable for large databases, we are motivated to consider approximate and faster solutions.

2. **Equi-Split**. This technique produces MBRs of fixed length l. It is a simple approach with cost linear in the length of a sequence. However, in pathological cases increasing the number of splits can result to larger space utilization,therefore the choice of the MBR length becomes a critical parameter (see figure 7 for an example).

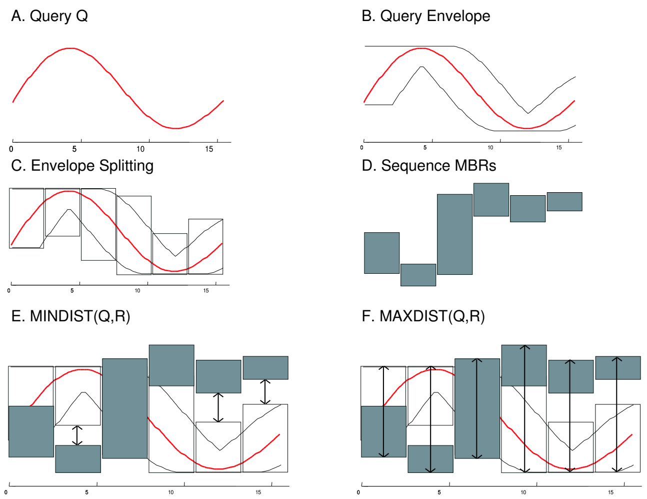

Figure 6: A visual intuition of the DTW indexing technique (the one-dimensional case is shown for clarity). The original query (A) is enclosed in a minimum-bounding envelope (B) like the LCSS approach. The MBE is split into its MBRs using equi or greedy split (fig. (C)). The candidate sequences in the database have their MBRs stored in the index (D). Between the query and any sequence in the index, the minimum and maximum distance can be quickly determined by examining the distance between the MBRs and the query's bounding envelope, as represented by the arrows in (E) and (F).

3. **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 $O(nlogn)$. We can see a sketch of the method in fig. 8. 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 can assign more splits for 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 ([8]).

```
Input: A spatiotemporal trajectory $T$ and an integer $k$ denoting the number of final MBRs.
■ For $0 \le i < n$ compute the volume of the MBR produced by merging $T_i$ and $T_{i+1}$. The results are stored in a priority queue.
■ While $\#MBRs < k$: 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.
Output: A set of MBRs that cover $T$.
```

Figure 8: The greedy algorithm for producing k MBRs that cover the trajectory T.

After a trajectory is segmented the MBRs can be stored in a 3D-Rtree. Using the greedy split each additional split will always lead to smaller (or equal) volume (figure 7). A similar greedy split algorithm is also used for splitting the MBE of the query trajectory Q.

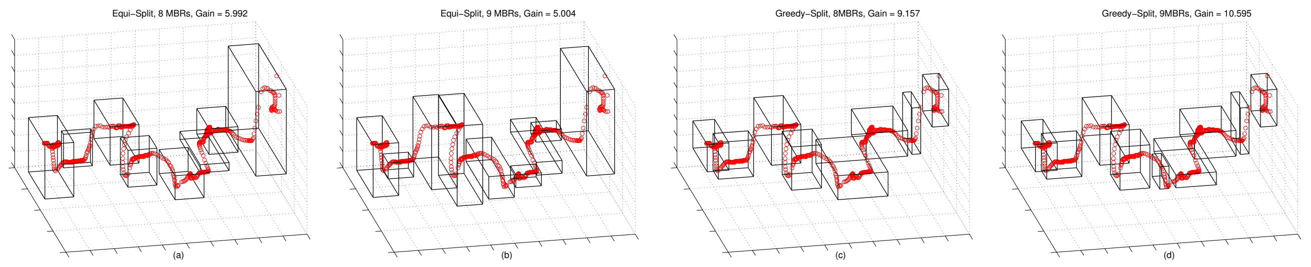

Figure 7: *(a):* 8 MBRs produced using equi-Split. The volume gain over having 1 MBR is 5.992. *(b):* Segmenting into 9 MBRs decreases the volume gain to 5.004. So, disk space is wasted without providing a better approximation of the trajectory. *(c):* 8 MBRs using greedy-Split. The volume gain over having 1 MBR is 9.157. *(d):* Every additional split will yield better space utilization. Segmentation into 9 MBRs increases volume gain to 10.595.

7. SUPPORTING MULTIPLE MEASURES

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 acceptably [11]. 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 9).

Therefore, our framework offers the unique advantage that multiple distance functions can be supported in a single index. The index 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, the creation of the query envelope and the type of operation between MBRs. In order to pose queries based on Euclidean distance we follow the steps:

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

■ The $minDist$ and $maxDist$ estimators for the Euclidean distance are derived by calculating the distance between the query and index MBRs, just like in the DTW case.

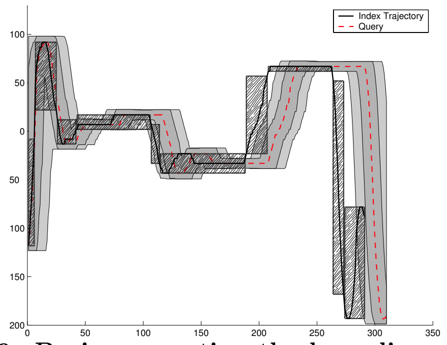

Figure 9: By incorporating the bounding envelope on the query, our approach can support Euclidean distance, constrained or full warping. This is accomplished by progressively expanding the MBE.

8. EXPERIMENTAL EVALUATION

In this section we compare the effectiveness of various splitting 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.

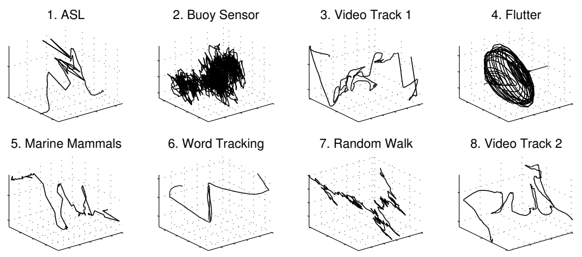

Figure 10: Datasets used for testing the efficiency of various MBR generation methods.

8.1 MBR Generation Comparison

The purpose of our first experiment is to test the space consumption of the presented MBR generation methods. We have used eight datasets with diverse characteristics, in order to provide objective results.

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 1 MBR, normalized by the maximum gain provided over all methods (for various number of splits).

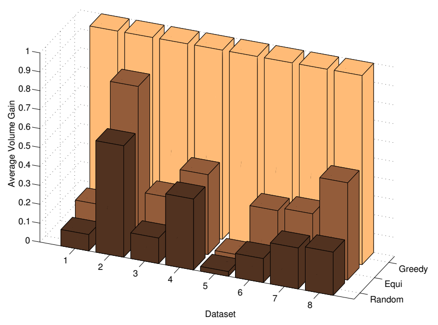

Figure 11: The greedy-split MBR generation algorithm presents the highest volume gain, by producing MBRs that consume consistently less space, over a number of datasets and for diverse number of generated MBRs

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.

$AvgVolGain$ is a number between 0 and 1, where higher numbers indicate increased volume gain (or less space consumption) against the competitive methods. In figure 11 we observe the average volume gain for the eight datasets. The greedy-split algorithm produced MBRs that took at least half the space, compared to equi-split. The equi-split offers slightly better results, than producing MBRs at random positions. The volume gain of greedy-split was less, 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.

8.2 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 the $LCSS$. For DTW there have been a few approaches to provide a lower bound of the distance; we refer to them as LB-Kim [12] and LB-Yi [21]. These lower bounds originally referred to 1D time-series; here we extend them in more 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 approach. However, since our index operates on the segmented time-series we only report the results on the MBRs.

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.

8.3 Matching Quality

We demonstrate the usefulness of our similarity measures in a real world dataset. The Library of Congress maintains thousands of handwritten manuscripts, and there is an increasing interest to perform automatic transcribing of these documents. Given the multiple variations of each word and due to the manuscript degradations, this is a particularly challenging task and the need for a flexible and robust distance function is essential.

We have applied the LCSS and DTW measures on word images extracted from a 10 page scanned manuscript. 4-dimensional time-series features have originally been extracted for each word. Here we maintain the 2 least correlated time-series features and treat each word as a trajectory. In figure 12 we observe the 3-KNN results using DTW for various word queries. The results are very good, showing high accuracy even for similarly looking words. Analogous results have been obtained using the $LCSS$.

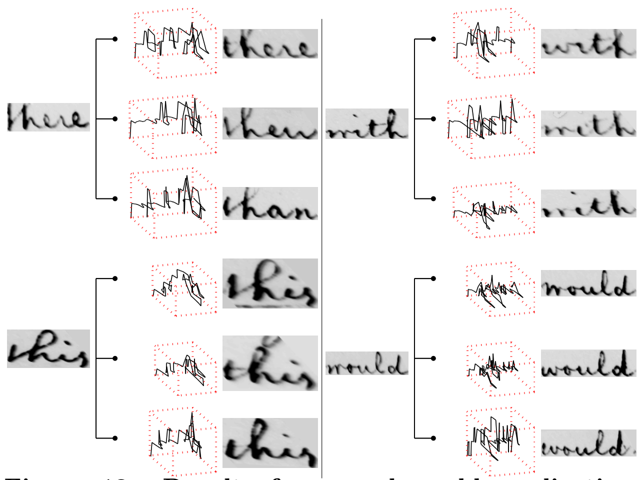

Figure 12: Results for a real world application. 3NN reported for each query, using Dynamic Time Warping to match features extracted from scanned manuscript words.

8.4 Index performance

We tested the performance of our index using the upper bound and the approximate similarity estimates, and compared it to the sequential scan. Because of limited space, the majority of the figures record the index performance using the $LCSS$ as a similarity measure. 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

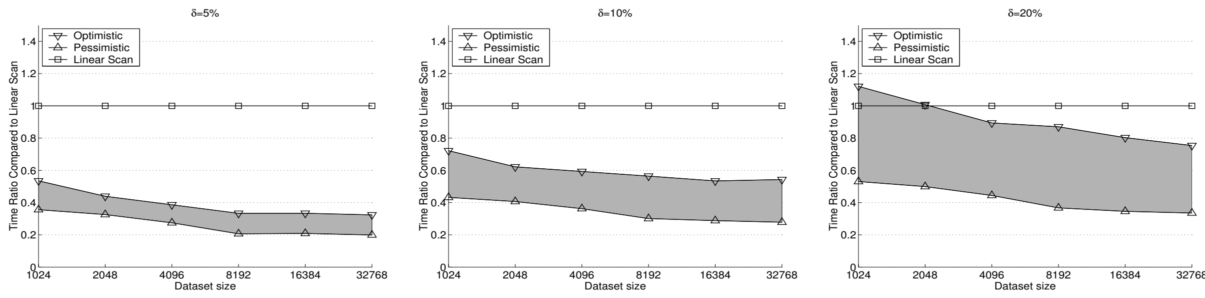

Figure 13: Index performance. For small warping windows the index can be up to 5 times faster than sequential scan without compromising accuracy. The gray regions indicate the range of potential speedup.

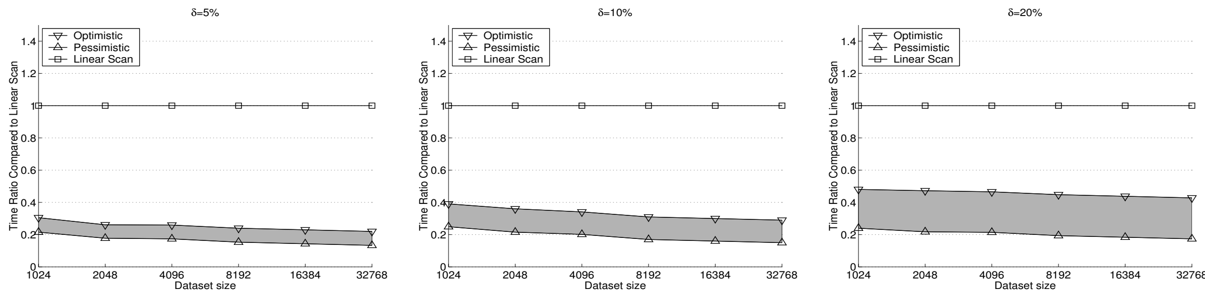

Figure 14: Using the approximate similarity estimates the response time can be more than 7 times faster.

both types of linear scan, because this provides a realistic upper or lower bound on the index speedup.

8.4.1 Dataset description

In order to test the index scalabilility we needed to construct large realistic multidimensional 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

The final dataset consisted of $2^{10} \dots 2^{16}$ trajectories. Taking under consideration that the average trajectory size is around 500 points, this resulted to a database with more than 16 million 2D points. The trajectories have been *normalized* by subtracting the average value in each direction of movement. All data and queries can be obtained by emailing the first author.

8.4.2 Results on the 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 $std/2$ of the query, which provided good and intuitive results.

■ **Dataset size**: In figure 13 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 15 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 17.

■ **Parameter** δ: The index performance is better for smaller warping lengths (parameter δ). The experiments 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 not 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.

8.4.3 Results on the 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 14. We observe that using this approximate similarity estimate, our index performance is boosted up. The use of the *V-similarity* estimate leads to more tight approximations of the original similarity compared to the L-similarity estimate, however

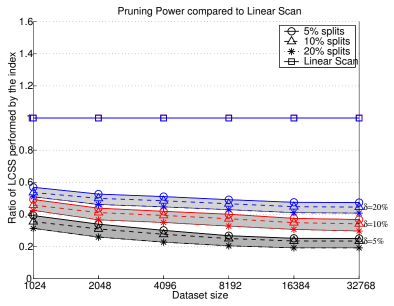

Figure 15: Each gray band indicates (for a certain warping window δ) the percentage of LCSS computations conducted by the index compared to linear scan.

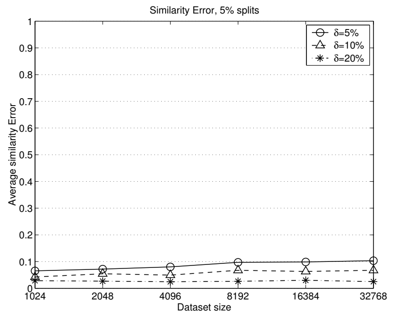

Figure 16: Using the V-similarity estimate, we can retrieve answers faster with very high accuracy. The LCSS similarity is very close (2-10%) to the exact answer returned by the sequential scan.

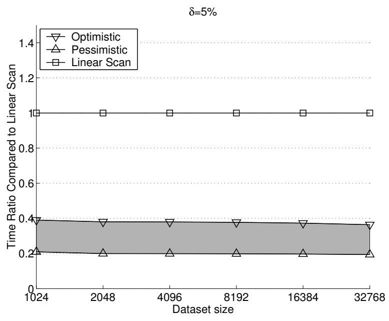

Figure 17: Index Performance using DTW as the distance measure. ($\delta = 5\%$). We can observe up to 5 times speedup.

now we may miss finding 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 16. We can see that the similarity returned by the V-similarity estimate is approximately within 5% of the actual similarity (5% splits used). Therefore, by providing two similarity estimates the user can decide for the trade-off between the 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.

9. CONCLUSIONS AND FUTURE WORK

In this paper we have presented an external memory indexing method for discovering similar multidimensional time-series. The unique advantage of our approach is that it can accommodate multiple distance measures. The method guarantees no false dismissals and depicts a significant execution speed up for the LCSS and DTW compared to sequential scan. We have shown the tightness of our similarity estimates and demonstrated the usefulness of our measures for challenging 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. In the future we plan to investigate the combination of several heuristics, in order to provide even tighter estimates.

Acknowledgements: We would like to thank Margrit Betke for providing us the *Video Track I and II* datasets. We also feel obliged to T. Rath and R. Manmatha for kindly providing the manuscript words dataset.

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

[3] 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.

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

[5] 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.

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

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

[8] M. Hadjieleftheriou, G. Kollios, V. Tsotras, and D. Gunopulos. Efficient indexing of spatiotemporal objects. In *Proc. of 8th EDBT*, 2002.

[9] T. Kahveci, A. Singh, and A. Gurel. Similarity searching for multi-attribute sequences. In *Proc. of SSDBM*, 2002.

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

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

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

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

[14] 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.

[15] 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.

[16] 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.

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

[18] M. Shimada and K. Uehara. Discovery of correlation from multi-stream of human motion. In *Discovery Science 2000.*

[19] 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.

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

[21] 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.

[22] Y. Zhu and D. Shasha. Query by humming: a time series database approach. In *Proc. of SIGMOD*, 2003.