The VLDB Journal (2006) 15(1): 1–20
DOI 10.1007/s00778-004-0144-2

REGULAR PAPER

Michail Vlachos · Marios Hadjieleftheriou · Dimitrios Gunopulos · Eamonn Keogh

Indexing Multidimensional Time-Series

Received: 3 May 2004 / Revised version: 13 October 2004 / Published online: 22 July 2005

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 an index structure that can support multiple distance measures. Our specific area of interest is the efficient retrieval and analysis of similar trajectories. Trajectory datasets are very common in environmental applications, mobility experiments, and 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.

Keywords Ensemble index · Longest common subsequence · Dynamic time warping · Trajectories · Motion capture

Edited by B. Ooi

M. Vlachos (✉)
IBM T.J. Watson Research Center, Hawthorne, NY, USA

M. Hadjieleftheriou · D. Gunopulos · E. Keogh
Computer Science Department, University of California, Riverside, CA, USA
E-mail: {marioh, eamonn, dg}@cs.ucr.edu

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 [9], motion capture [49], 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 [7, 47]. 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 multidimensional 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 [51]. In general, any dataset that involves the storage of multiattribute data sequences can be considered and treated as a set of multidimensional trajectories.

Some very important data mining tasks for multidimensional trajectories involve the discovery of objects that move similarly or closely follow certain 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 that 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 [52].

By incorporating the ability to allow warping in time as a requirement for our model our algorithms are instantly challenged with quadratic execution time. Nevertheless, to speed up 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. In addition to providing upper-/lower-bounding measures that guarantee *no false dismissals*, we also present similarity/distance approximations that can significantly reduce the index response time. 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 multidimensional 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 multidimensional 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 Sect. 2 related work is presented. In Sect. 3 we formalize the new similarity functions by extending the LCSS and DTW models. Section 4 analyzes the index construction, and Sect. 5 presents methods for producing efficient trajectory approximations. Section 6 describes the proposed prefiltering scheme and proves the claim of no false dismissals. Section 7 discusses various possible optimizations, and Sect. 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 Sect. 10 illustrates some real-world applications that can utilize our methodology. Finally, Sect. 11 concludes the paper and summarizes our contributions.

2 Related work

Work related to trajectory similarity can be found in the domain of time-series analysis, which mostly concentrates 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, 18, 53]. On the other hand, the model cannot deal well with outliers and is very sensitive to small distortions in the time domain [52]. 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 [29, 31]. 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 the comparison of specific time-series features (Landmarks [39] or signatures [16]). 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 by 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 making its adaptation challenging for large datasets. All work referenced above concentrates primarily on one-dimensional time series. Techniques that deal with multidimensional sequences appear in [28, 35]. 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, 6]. 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. DTW is also a ubiquitous tool in the biometric/surveillance community; it has been used for tracking time series extracted from video [20], classifying handwritten text [37, 46], and even fingerprint indexing [34]. Finally, perhaps the most familiar example of easily classifiable patterns that nevertheless cause problems to L_p metrics are electrocardiograms. The cardiological community has used DTW for several decades [50].

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, 54], 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 requires excessive disk space (about ten 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 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, 5, 10, 14]. In [52] 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 [30] is closest in spirit to our approach; however, it only addresses 1D 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 equilength 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.

Therefore, compared to the work in [30], this paper is enhanced in the following aspects: (i) support for multidimensional sequences, (ii) support for multiple measures, (iii) run-time setting of parameter delta, and (iv) better distance approximation through the use of variable-length MBRs for sequence approximation and with the introduction of upper bounds that improve the pruning power.

The biggest advantage of this work is that 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 extensions of the LCSS and DTW models for multiple dimensions.

3.1 LCSS model for multidimensional time-series

The original LCSS model refers to 1D sequences; therefore it needs to be extended to the multidimensional 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 us assume that measurements are taken at fixed and discrete time intervals. If this is not the case, then interpolation can be used [24, 40]. Let A and B be 2D time-series with sizes n and m, respectively, where $A = ((a_{x,1}, a_{y,1}), \ldots, (a_{x,n}, a_{y,n}))$ and $B = ((b_{x,1}, b_{y,1}), \ldots, (b_{x,m}, b_{y,m}))$. Also, let $\text{Head}(A) = ((a_{x,1}, a_{y,1}), \ldots, (a_{x,n-1}, a_{y,n-1}))$.

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

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

Constant δ controls the flexibility of matching in the time domain, and constant ϵ is the matching threshold in space. This LCSS model can be computed using dynamic programming. Its time complexity is in the order of $O(\delta(n + m))$, if

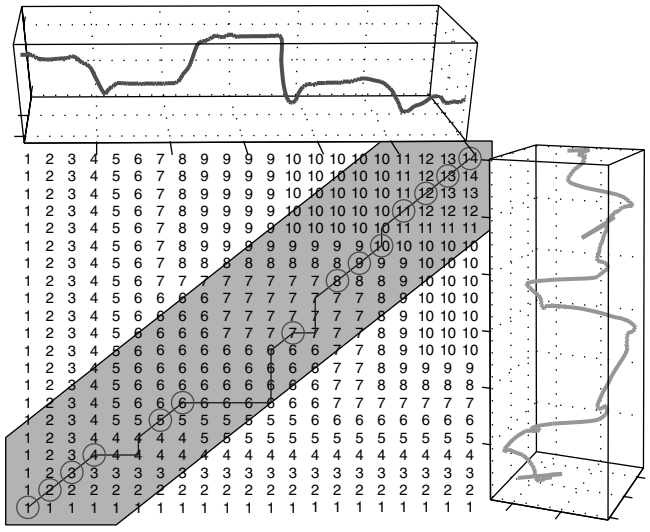

Fig. 1 Dynamic programming computation of LCSS. The matching windows in time are indicated by the *gray area* ($\delta = 5$)

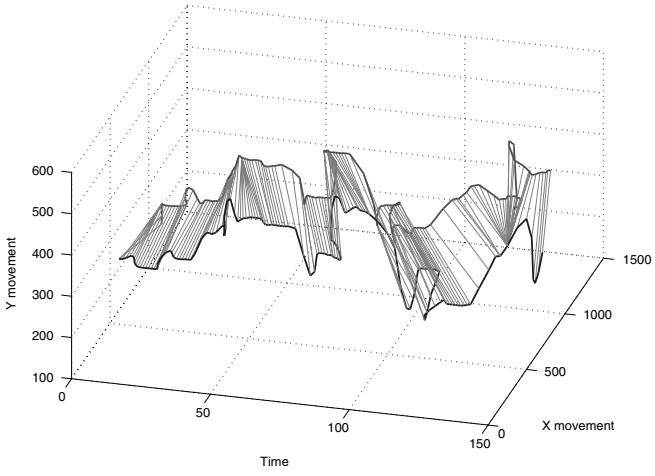

Fig. 2 DTW matching between two 2D trajectories

we only allow a matching window δ in time [14]. An instance of the dynamic programming execution between two trajectories is depicted in Fig. 1. The population of the array starts at the lower left corner, and the computed value at the top right corner provides the final similarity between the two trajectories. The value at position $[i, j]$ of the array increments by one the maximum of the adjacent array values ($[i-1, j]$, $[i, j-1]$,$[i-1, j-1]$), only if the values of trajectory A at time i and trajectory B at time j do not differ more than ϵ in all dimensions (two dimensions for this example). In the case when the matching window in time δ is also imposed, then only the elements of the array that are up to δ positions from the diagonal need to be calculated. In the example of Fig. 1 the gray region indicates the elements of the array that will be computed under a matching window of $\delta = 5$.

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 be normalized. One can derive a normalized distance based on the LCSS similarity as follows:

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

$$D_{\delta,\epsilon}(A, B) = 1 - \frac{\text{LCSS}_{\delta,\epsilon}(A, B)}{\min(n, m)}.$$

3.2 DTW model for multidimensional time-series

This section presents an extension of the original DTW function as described by Berndt and Clifford [8] for multidimensional trajectories. For simplicity we talk about 2D trajectories.

Definition 3 The DTW distance between 2D time-series A and B is defined as follows:

$$\begin{aligned}\text{DTW}(A, B) = L_p((a_{x,n}, a_{y,n}), (b_{x,m}, b_{y,m})) + \\ \min\{\text{DTW}(\text{Head}(A), \\ \text{Head}(B)), \text{DTW}(\text{Head}(A), B), \\ \text{DTW}(A, \text{Head}(B))\}.\end{aligned}$$

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 $O(\delta(n + m))$, similar to the LCSS. To meaningfully compare distances between sequences of different lengths, one can normalize the total DTW distance by the length of the warping path [44]. Figure 2 shows an example of a DTW matching between two trajectories.

3.3 Constraining the warping

Restricting the allowed warping length in time (parameter δ) can substantially speed up the execution of the dynamic programming algorithm for the DTW and the LCSS models. However, we will demonstrate (using real datasets) that restrained warping within a window δ offers more advantages:

1. For most datasets full-length warping is not required in order to achieve accurate matching. 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.

   Using the *camera Mouse* program [9], we have obtained multiple instances of various words created by tracking human movement over time. For example, by tracking the movement of some human feature (e.g.,

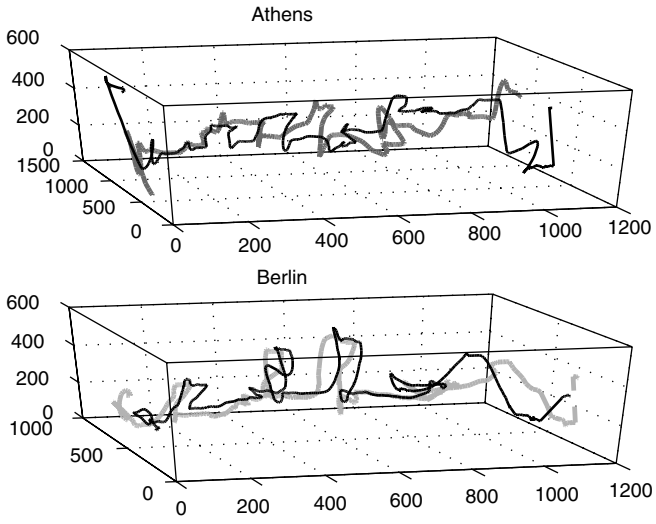

Fig. 3 Two words of the cameraMouse dataset (two instances each)

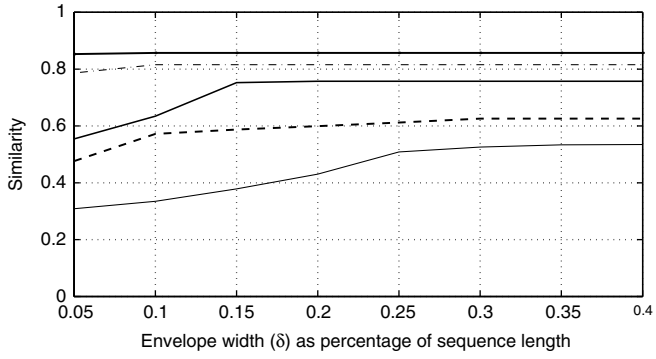

Fig. 4 Pairwise LCSS similarity over increasing warping windows

tip of nose), the user can utilize a "virtual keyboard" and "write" words. The program is targeted primarily for people with movement disabilities, providing an easier communication interface to the computer. In this manner, spatiotemporal sequences representing various words were collected (Fig. 3). In Fig. 4 one can observe the trend in the computed LCSS similarity, for increasing warping lengths, over five pairwise computations. It is apparent that the computed similarity follows the law of diminishing returns, since warping length larger than 20% does not change the computed sequence similarity.

Similar results have also been noted in Keogh, et al. [44] for other time-series datasets that span the areas of botany, video tracking, text recognition, etc. The authors also suggest that full warping can be futile since it typically does not yield better distance estimates. Using eight time-series datasets with known class labels, the authors searched for the optimal warping constraint and discovered that 2–8% warping provides the best classification accuracy.

2. In certain datasets extended warping not only yields no additional gains, but it also hurts accuracy. The following simple experiment was performed: we labeled each sequence of the test datasets using a "leave-one-out" one nearest neighbor classification scheme. Therefore, for each trajectory in the dataset, we find the closest match according to the LCSS similarity. The classification is considered correct only if the label of its nearest neighbor is the same as the original trajectory label. The total accuracy is recorded by repeating the experiments for all dataset sequences.

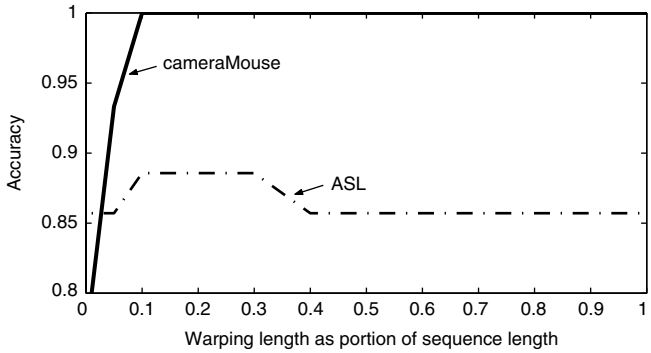

Fig. 5 "Leave-one-out" classification accuracy for two datasets. We observe that excessive warping can hurt the performance in certain datasets

Figure 5 illustrates the accuracy results obtained for the *camera Mouse* dataset and for 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% deteriorates the accuracy. It is apparent that excessive warping can distort the true trajectory distances and create artificial matches by allowing long and degenerate matching correspondences. Very similar results on the utility of tight constraints have been noted in [45].

The above discussion demonstrates that our choice of constraining the warping in time (parameter δ), besides improving the algorithm run time, is also a highly desirable feature for providing better similarity estimates and for improving the classification/clustering accuracy.

3.4 Parameter setting

In this section we will provide some general directions on how one can set the parameters for the LCSS model. Clearly, the values for δ and ϵ are dependent on the application and the dataset. For most datasets we had at our disposal, we discovered that setting δ to more than 10–20% of the trajectories length did not change the similarity estimate drastically. Furthermore, as demonstrated in the previous discussion, after some point the similarity stabilizes to a certain value. Therefore, the δ parameter can be first approximated by using the DTW measure (which uses just one parameter) and estimating the warping length, after which classification accuracy does not improve.

Subsequently the parameter ϵ can be estimated by probing a few values on the discovered δ parameter and recording

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

the one that provides the best class separation. In our experiments we used values in the range of 10–25% of the standard deviation of the examined trajectories, which yielded good and intuitive results.

Even though the setting of the second parameter ϵ may seem to reduce the usability of the LCSS model, we prefer to view it as an asset of the method, because it provides additional spatial filtering abilities that no other measure can provide. For example, a possible query at an airport watch tower could be: "Find all planes that followed a similar route as plane X and were always within a radius of ϵ." Such queries can be easily answered using the LCSS model only.

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 the given query. This can be accomplished by discovering a close match to the query as early as possible. A fast prefiltering 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 Fig. 6 and was also successfully used in [30, 54]. There are certain preprocessing steps that need to be followed:

1. The sequences are segmented into MBRs, which are stored in an R-tree.

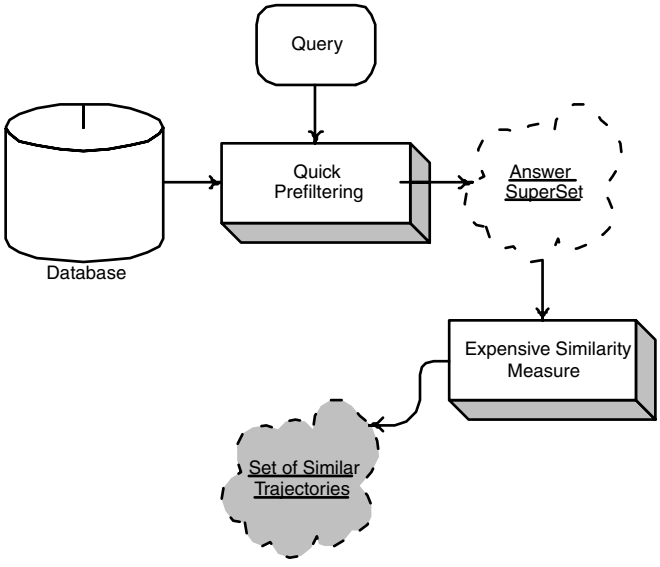

Fig. 6 Very distant trajectories can be discarded by using a fast prefiltering step. For the remaining trajectories an expensive, but accurate, similarity measure can be used

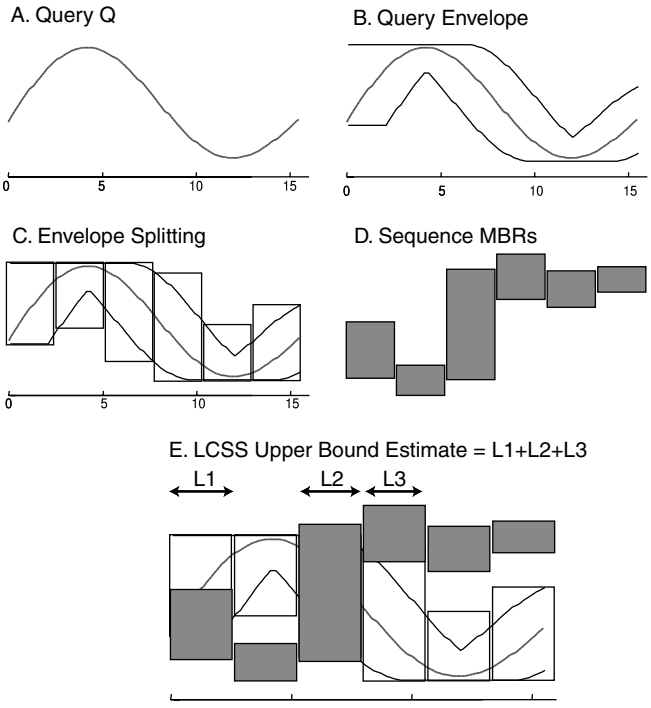

Fig. 7 One-dimensional illustration of the proposed approach. The query is extended into a bounding envelope, which in turn is approximated with a set of MBRs. Overlap between the query and the index MBRs suggests areas of possible matching and yield candidate trajectories

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 sequences.

These steps are illustrated in Fig. 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 1D sequence $A = (a_{x,1}, \ldots, a_{x,n})$. Ignoring for now the parameter ϵ, we would like to perform a very fast LCSS_δ 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 and after time i. The envelope that includes all these points defines the areas of possible matching. Nothing outside this envelope can ever 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 extend ϵ above and below the original envelope in space (Fig. 8). The notion of a bounding envelope can be trivially extended to

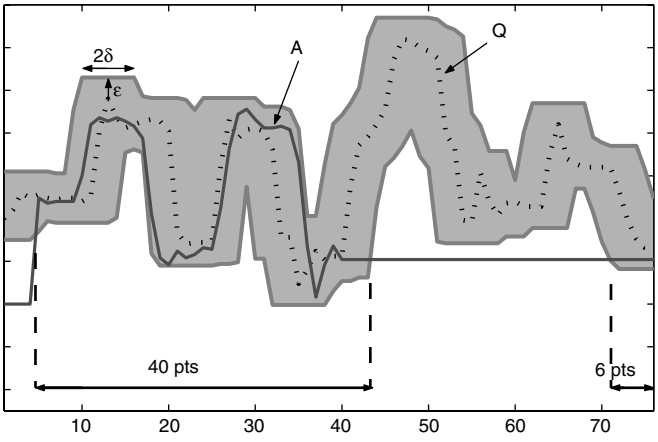

Fig. 8 The *minimum bounding envelope* (MBE) within δ in time and ϵ in space of a sequence. Nothing that lies outside this envelope can ever be matched

more dimensions. For instance, MBE(δ, ϵ) for a 2D trajectory $Q = ((q_{x,1}, q_{y,1}), \ldots, (q_{x,n}, q_{y,n}))$ covers the following area:

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

where for dimension d at time i:

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

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

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

For example, in Fig. 8 the LCSS similarity between MBE_Q and sequence A is 46. This value represents an overestimate or *upper bound* of the LCSS 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 sequences Q and A the following holds: $D_{\delta,\epsilon}(\text{MBE}_Q, A) \leq D_{\delta,\epsilon}(Q, A)$.*

Proof (Sketch). $D_{\delta,\epsilon}(\text{MBE}_Q, A) = 1 - \frac{\text{LCSS}_{\delta,\epsilon}(\text{MBE}_Q, A)}{\min(|Q|,|A|)}$; therefore, it is sufficient to show that: $\text{LCSS}_{\delta,\epsilon}(\text{MBE}_Q, A) \geq \text{LCSS}_{\delta,\epsilon}(Q, A)$. 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 appear 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 were originally defined only for 1D time-series (and are explained below for the 1D 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 four-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.

The lower-bounding function introduced by Yi et al. [54] (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.

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

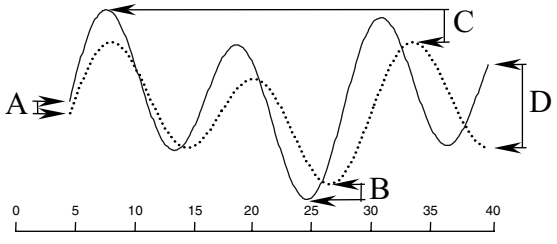

Fig. 9 A visual intuition of the lower-bounding measure introduced by Kim et al. The maximum of the squared difference between the first (**a**), last (**d**), minimum (**b**), and maximum points (**c**) of the two sequences is returned as the lower bound

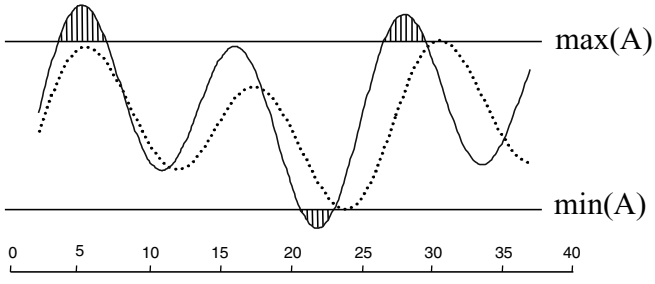

Fig. 10 A visual intuition of the lower-bounding measure introduced by Yi et al. The sum of the squared length of the *gray lines* represents the minimum of the corresponding points' contribution to the overall DTW distance and thus can be returned as the lower-bounding measure

is to embed the sequences into Euclidean space such that the distance structure of the original space is approximately preserved. Then, a traditional multidimensional 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, we first construct the MBE of the query Q, described by a low and high envelope (as defined above but setting $\epsilon = 0$). In contrast 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:

$$\text{DTW}(\text{MBE}_Q, A) = \sqrt{\sum_{d=1}^{D}\sum_{i=1}^{n}\begin{cases}(a_{d,i} - \text{EnvHigh}_{d,i})^2 & \text{if } a_{x,i} > \text{EnvHigh}_{d,i} \\ (a_{d,i} - \text{EnvLow}_{d,i})^2 & \text{if } a_{x,i} < \text{EnvLow}_{d,i} \\ 0 & \text{otherwise}\end{cases}}.$$

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 Fig. 11. This lower bound could be seen as a generalization for multidimensional time series of the technique proposed in [30].

It can be proven that this distance *lower bounds* the actual time warping distance. A proof for the 1D case appeared in [30]. The extension to multiple dimensions is straightforward. This measure has since been extended and used by several researchers, including [46, 55].

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

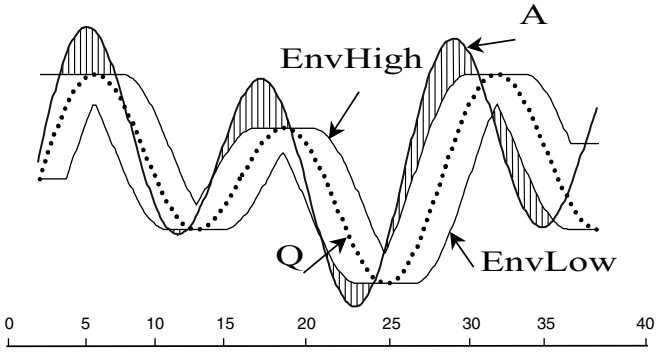

Fig. 11 An illustration of the lower-bounding function for DTW. The original sequence Q (*dotted*) is enclosed in a *minimum bounding envelope*. The squared sum of the distances from every part of sequence A not falling within MBE_Q to the nearest orthogonal edge of MBE_Q is returned as the lower bound

In the last two sections we 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 multidimensional R-tree. We will show how to compute the upper/lower bounds using only the MBRs that provide the trajectory approximations.

5 MBR generation

Given a multidimensional 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 MBE minimizes the expected similarity approximation error.*

Proof (Sketch) By minimizing the volume of the MBRs containing the bounding envelope, we minimize the number of intersecting MBRs; thus a tighter similarity approximation is obtained.

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. *Equisplit*. 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 Fig. 12 for an example).
3. *Random split*. With this approach the time positions where the trajectory is segmented into MBRs are produced randomly. As we shall see in our experiments, this approach performs competitively with equisplit for a number of datasets [41].
4. *Greedy split*. The greedy approach is our implementation of 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 $O(n \log n)$. A sketch of the method is shown

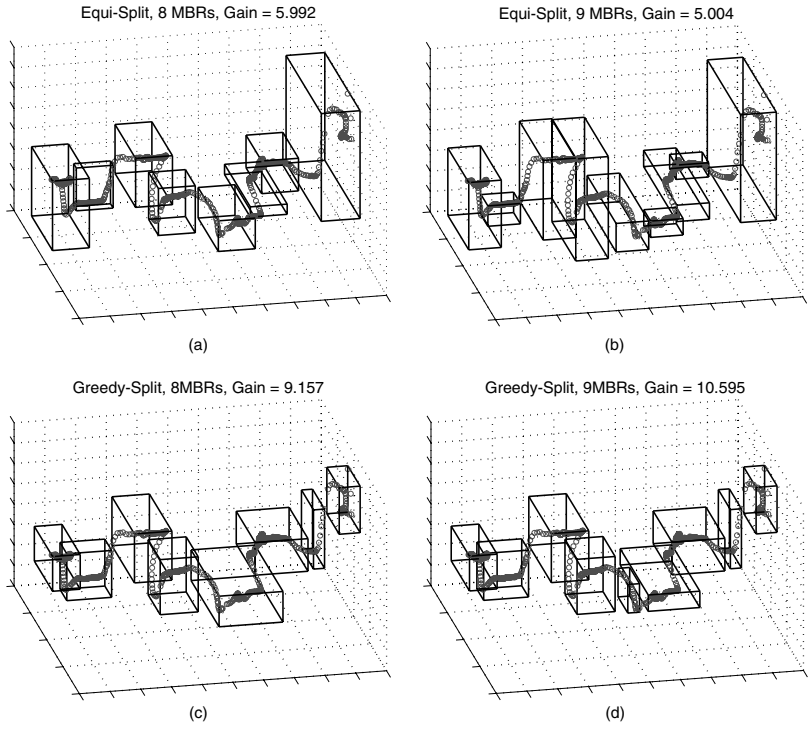

Fig. 12 **a** Eight MBRs produced using equisplit . The volume gain over having only one MBR is 5.992. **b** Segmenting into nine MBRs decreases the volume gain to 5.004. Thus disk space is wasted without providing a better approximation of the trajectory. **c** Eight MBRs using greedy split. The volume gain over having only one MBR is 9.157. **d** Every additional split will yield better space utilization. Segmentation into nine MBRs increases volume gain to 10.595

in Algorithm 1. Alternatively, instead of assigning the same number of MBRs to all objects, according to our space requirements we can assign a total of K MBRs to be distributed among all objects. This method can provide better results since we assign more MBRs 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 $O(K + N \log N)$, for a total of N objects [26].

In Fig. 12 we examine how additional MBRs affect the volume consumption of the equisplit and greedy split methods. To this end, we compute the volume gain, which conveys how much space reduction is achieved when using k MBRs compared to using only one MBR (larger volume gain suggests better space utilization). For this specific example, when choosing equisplit, the use of nine instead of eight MBRs actually *increases* space utilization, while for the greedy split each additional MBR leads to a reduction in the total volume consumption. A similar greedy algorithm is used for splitting the MBE of the query trajectory Q.

Algorithm 1 Greedy split

Input: Trajectory T, integer k denoting the number of final MBRs.
Output: A set of MBRs that cover T.
1: **for** $0 \leq i < n$ **do**
2: Compute the volume of the MBR produced by merging T_i and T_{i+1}.
3: Store the results in a priority queue.
4: **end for**
5: **While** $\#MBRs > k$ **do**
6: Using the priority queue merge the pair of consecutive MBRs that yield the smallest increase in volume.
7: Delete the two merged MBRs and insert the new one in the priority queue.
8: **end while**

6 Quick pruning of dissimilar trajectories

Suppose that we have an index with 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 kth $bestSoFar$ match. DTW can be handled by reversing the signs of the inequalities and utilizing the relevant *distance* estimates between the trajectory

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

```
Box Env = constructMBE_{δ,ε}(Q)
Vector V_Q = CreateMBRs(Env)
        //V_Q contains a number of boxes
Priority queue PQ ← ∅  // Keep trajectories sorted according
        to similarity estimates
for each box B in V_Q do
    V = I. intersectionQuery (B)  // V contains all trajectory
        MBRs that intersect with B
    if Method–Exact then PQ ← computeL-
        SimilarityEstimates(V, B)
          // upper-bound
    else PQ ← computeV-SimilarityEstimates(V, B)
          // approximate
end for
BestSoFar = 0; Best ← ∅
while PQ not empty do
    E ← PQ.top
    if E.estimate < BestSoFar then break
    else
        D = computeLCCS_{δ,ε}(Q, E) exact
        if D > BestSoFar then BestSoFar = D;
          Best ← E
end while
Report Best
```

MBRs. Next, we will apply these similarity (or distance) estimates between the MBRs when utilizing the LCSS or the 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, we first present several operators.

Each trajectory T can be decomposed into a number of MBRs. The ith 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 MBRs $M_{P,i}$ and $M_{R,j}$, belonging to objects P and R, respectively. In what follows, the notation $||Intersection||_d$ denotes the intersection or overlap of the two MBRs in dimension d.

1. $\bigcap_t^{(c)}(M_{P,i}, M_{R,j}) = ||Intersection||_t$, ($c$ stands for "complete")
   where $M_{R,j}.x_l \le M_{P,i}.x_l \le M_{R,j}.x_h$ and
   $M_{R,j}.x_l \le M_{P,i}.x_h \le M_{R,j}.x_h$ and
   $M_{R,j}.y_l \le M_{P,i}.y_l \le M_{R,j}.y_h$ and
   $M_{R,j}.y_l \le M_{P,i}.y_h \le M_{R,j}.y_h$
   or similarly by rotating $M_{R,j} \rightleftharpoons M_{P,i}$
   Therefore, this operator computes the time intersection of two MBRs 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")
   The operator returns the time intersection between two MBRs when one does *not* fully contain the other in the spatial dimensions.
3. $\bigcap_V(M_{P,i}, M_{R,j}) = ||Intersection||_t \cdot ||Intersection||_x \cdot ||Intersection||_y$
   The final operator provides the common volume intersection.

Using these operators we derive 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 $L_{t,\text{complete}}$, 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. 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 Fig. 13, top right).
   - The list $L_{t,\text{partial}}$, an element $L(P)$ of which is defined as:

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

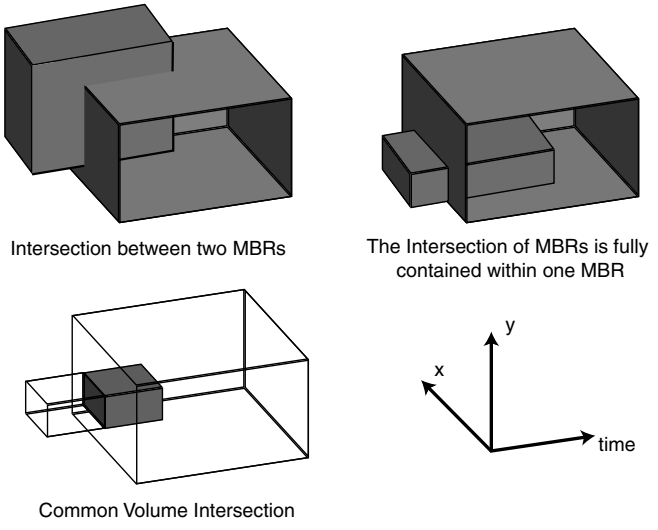

Fig. 13 *Top left*: Intersection recorded in list $L_{t,\text{partial}}$. *Top right*: Intersection recorded in list $L_{t,\text{complete}}$. *Bottom left*: Percentage of volume intersection kept in L_V

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 Fig. 13, top left). Regarding a query Q for any trajectory P, the sum of $L_{t,\text{complete}}(P)+L_{t,\text{partial}}(P)$ will provide an *upper bound* on the similarity of P and Q:

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

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

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

2. *Approximate estimates (V-similarity estimate).* This second estimate is based on the intersecting volume of the MBRs. This type of estimate is stored in list L_V:
   - Any element $L_\text{V}(P)$ 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.

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

where $\|M\|_\text{V}$ denotes the volume of MBR M and $\|M\|_t$ its length on the time dimension.

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 a 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 lower and upper bounds on the actual distance by evaluating the distance of the query MBE and the trajectory MBRs. The operators we define between the MBRs can be considered extensions of [48].

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

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

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

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 [30, 55]. The `MAXDIST` operator is defined as:

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

where

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

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

Sequences with lower bounds larger than the smallest computed upper bound can be pruned. With this additional prefiltering step the number of actual executions of the exact DTW distance is significantly reduced. In our experiments we have observed an additional 10–15% speedup in

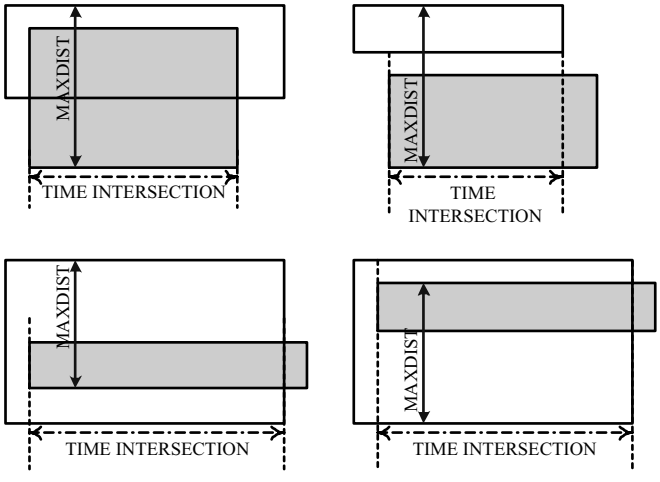

Fig. 14 Operator `MAXDIST` between two MBRs

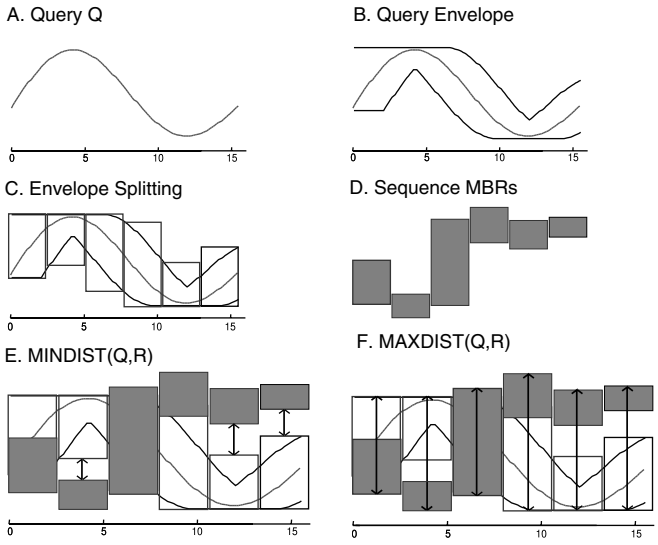

Fig. 15 A visual intuition of the DTW indexing technique (the 1D case is shown for clarity). The original query (**a**) is enclosed in a minimum bounding envelope (**b**) like the LCSS approach. The MBE is split into its MBRs using equisplit or greedy split (**c**). The MBRs of candidate sequences are 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 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 Fig. 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 (Fig. 16). That is:

$$Q_{i,\mathrm{MBR}(j)} = \frac{100 \times |Q_i \mathrm{points}|}{|\mathrm{MBR}(j)\mathrm{points}|}, \quad \text{for } i = 1, \ldots, 4\,.$$

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

A second optimization applies only to the LCSS estimates and explains the separation of the *L-estimate* into two lists. This variation can be used for providing approximate results with very little probability of false dismissals. It is based on the observation that the estimates stored in list

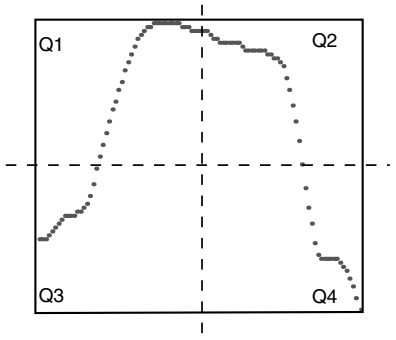

Fig. 16 As an additional optimization one can store with every MBR the percentage of points for each quadrant (projected in two dimensions for clarity)

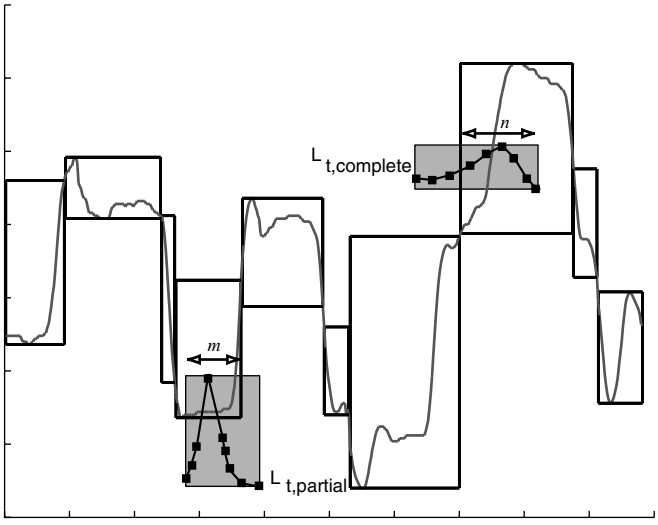

Fig. 17 The estimate stored in list $L_{t,\mathrm{partial}}$ can greatly overestimate the actual similarity. On the other hand, the estimate in $L_{t,\mathrm{complete}}$ represents a more accurate match

$L_{t,\mathrm{partial}}$ significantly overestimate the LCSS similarity. We depict this with an example: In Fig. 17 we see the two different types of MBR intersections in time. For the intersection stored in list $L_{t,\mathrm{partial}}$, 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 that is recorded in the list $L_{t,\mathrm{complete}}$, we can be sure that there are n points that 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 $L_{t,\mathrm{partial}}$. Although now it is possible to miss the best match to the query, we are going to find a close match faster because the new estimates will adhere closely to the original similarity (even though they provide no guarantee of no false dismissals).

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 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, and 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. Thus we can start by using a query with $\delta = 0$ (no bounding envelope) and increase it progressively in order to find more flexible matches (Fig. 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 already discovered a tight match to the query.

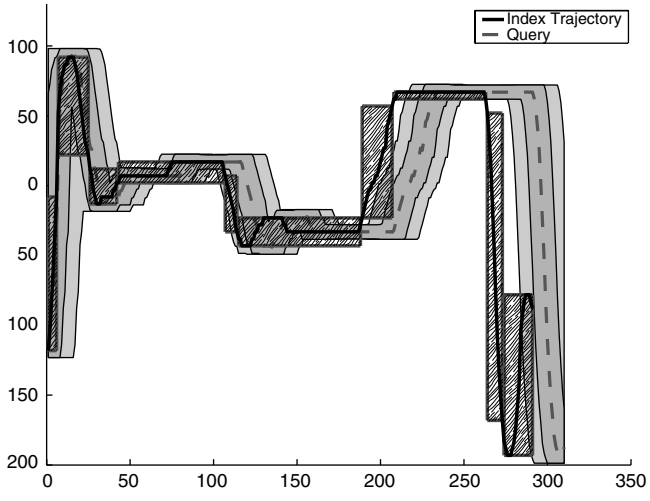

Fig. 18 By incorporating the bounding envelope on the query our approach can support Euclidean distance, constrained warping, or full warping. This is accomplished by progressively expanding the MBE

Therefore, our framework offers the unique advantage that multiple distance functions can be supported in a single index. The indexed sequences have been segmented into MBRs without any envelope applied on them and need not be adjusted again. For utilization of different measures on the index, the only aspects that change are:

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 the DTW.

9 Experimental evaluation

We will compare the effectiveness of the various MBR generation methods, and we will 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 were conducted on an AMD Athlon 1.4-GHz with 1 GB RAM and 60 GB 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 (equi, greedy, and random split). We have used eight datasets with diverse characteristics in order to provide objective results (Fig. 19). The majority of the datasets are real-world trajectories, extracted from video-tracking applications, marine mammal migration patterns, etc.

We evaluate the space consumption by calculating the "average volume gain" ($AvgVolGain$), which is defined as the percentage of volume saved when using i MBRs over the volume when using only one MBR (this ratio is denoted as $vGain$), normalized by the maximum gain provided over all methods. For completeness we evaluate the volume gain ratio over different numbers of trajectory splits.[2] Therefore, the average volume gain is computed by averaging the results of 20, 40, 60, 80, and 100 splits (number of split experiments $= |splitExp| = 5$). More concisely, the average gain $AvgGain_i$ for splitting method $i = equi, greedy, random$ is

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

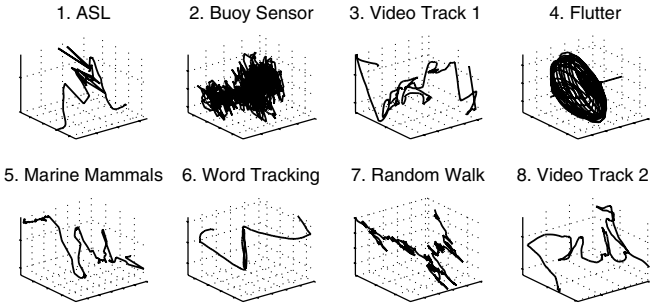

Fig. 19 Datasets used for testing the efficiency of various MBR generation methods

defined as:

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

$AvgVolGain$ 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 ran each experiment 100 times and averaged the results. In Fig. 20 we plot the average volume gain for all eight datasets. The greedy split algorithm produced MBRs that consumed at least half the space of equisplit. Equisplit offers slightly better results than producing MBRs at random. The volume gain of greedy split was not as dramatic only for the *buoy sensor* (dataset 2), which is a very busy and unstructured signal. This experiment validates our decision to use the greedy split method. Since the indexed MBR trajectories will take less space, we also expect tighter similarity estimates and therefore fewer false positives.

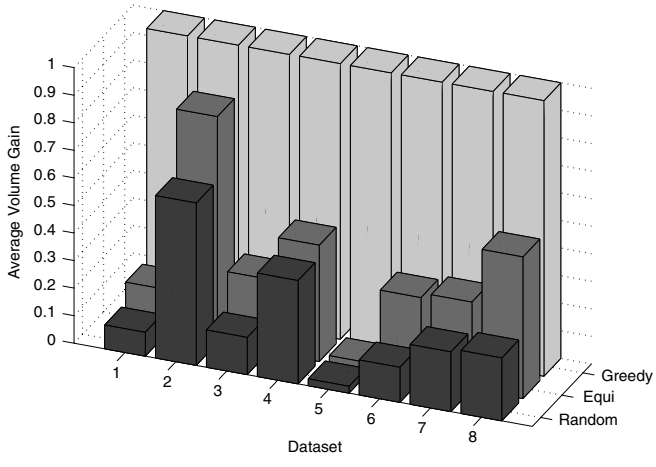

Fig. 20 The greedy split MBR generation algorithm presents the highest volume gain by producing MBRs that consume consistently less space over a number of datasets and for diverse numbers of generated MBRs

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 the best of our knowledge, this paper introduces the first upper-bounding technique for LCSS, which is why we only report results for the equisplit and greedy split methods. For the DTW we compare our lower bounds with LB-Kim and LB-Yi. These lower bounds originally referred to 1D 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 two to three 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 equisplit method. 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 to the actual distance.

9.3 Index performance

We test the performance of our index for the LCSS using the upper bound and the approximate similarity estimates and compare 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 100 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 *best-SoFar*. We call this *optimistic* linear scan. *Pessimistic* linear scan is the one that 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 multidimensional datasets. To this end, we utilized the aggregation of our eight real datasets as seeds for generating more trajectory variations. We created multiple copies of the original trajectories by incorporating the following features:

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

	LCSS				*DTW*					
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
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**

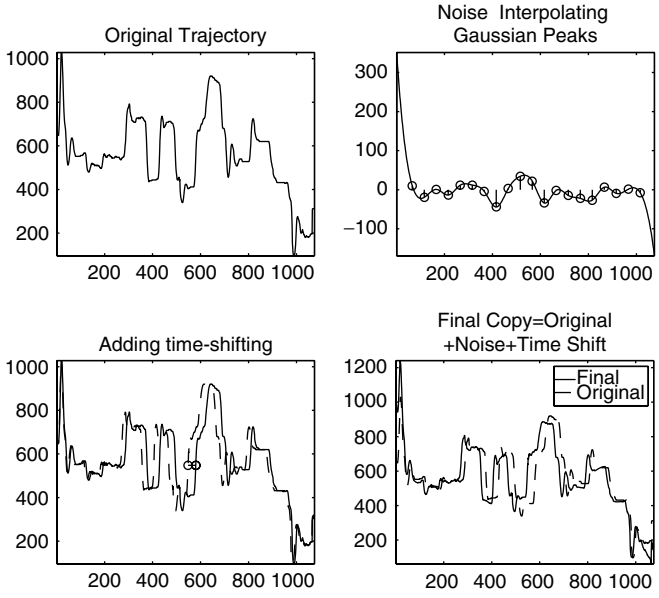

Fig. 21 Creation of multiple copies from the real datasets by adding smooth variations in the pattern and time shifting at random positions

– 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 in unnatural sequences 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 were able to create the smooth variations that existed in the original datasets (Fig. 21).

For our scalability experiments we utilized datasets with cardinalities ranging from 2^{10}–2^{16} trajectories. Taking into consideration that the average trajectory size is around 500 points, this resulted in a database with more than 16 million 2D points. The trajectories were *normalized* by subtracting the average value in each direction of movement.

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 $1/4$ of the query standard deviation.

- *Dataset size:* In Fig. 22 we can observe how the performance of the index scales with the database size (for various warping lengths). 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 five times faster than the linear scan. In Fig. 24 we also demonstrate the pruning power of the index as a true indicator of the index efficacy (since it is not biased by the implementation details). Using the index we perform two to five times fewer LCSS computations than the linear scan. We observe similar speedup when using the DTW as the distance function in Fig. 26.
- *Parameter δ:* The experiments record the performance for warping lengths ranging from 5 to 20% of the query's length. Increasing δ values signify larger bounding envelopes around the query, which results in larger search space and less accurate similarity estimates. While the graphs suggest that an index cannot be useful under full warping, this does not detract from the usefulness of the index, because, as we mentioned earlier, full warping might simply be futile since setting $delta$ to 10–20% of the query's length is typically sufficient for most datasets and applications.
- *Number of splits:* Even though a greater number of MBRs for each trajectory implies better volume utilization, more MBRs also lead to increased I/O cost. When in the figures we refer to $x\%$ splits, this means that we have assigned a total of $x/100(\sum_{i=1}^{n}(|T_i|))$ splits for all sequences T_i (where $|T_i|$ denotes the sequence length). In our figures we provide the 5% split scenario for the

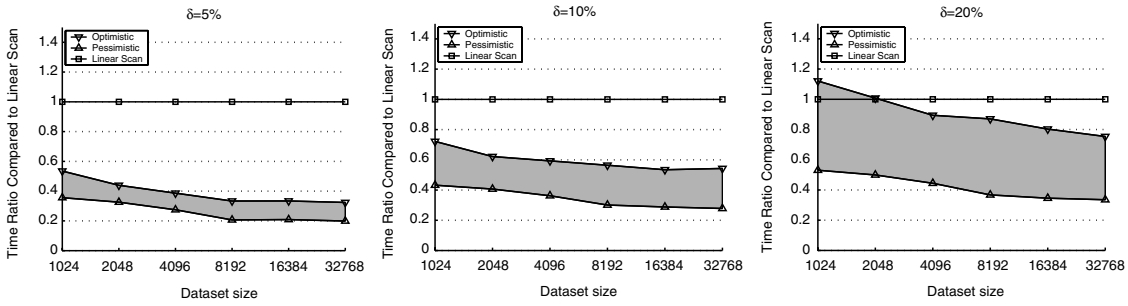

Fig. 22 Index performance for LCSS. For small warping windows the index can be up to five times faster than sequential scan without compromising accuracy. The *gray regions* indicate the range of potential speedup

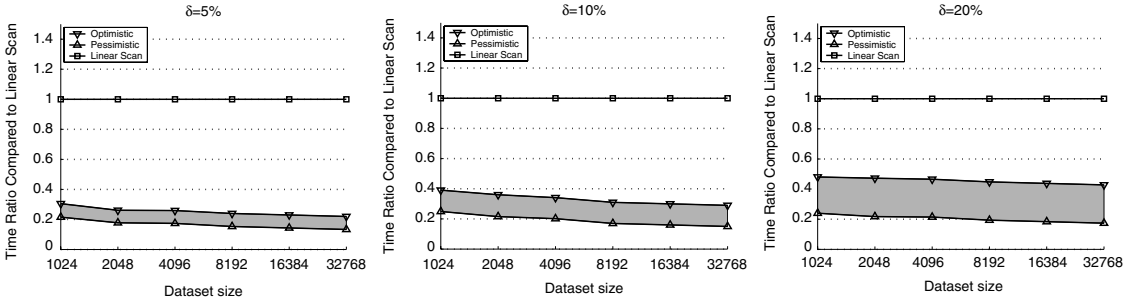

Fig. 23 Using the approximate similarity estimates for the LCSS, the response time can be more than seven times faster

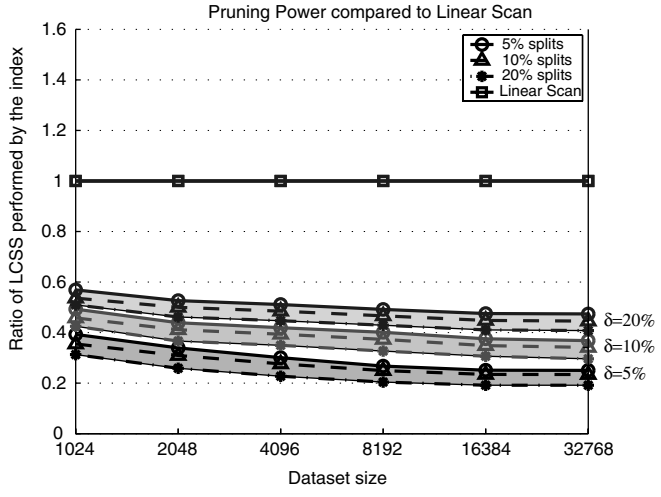

Fig. 24 Each *gray band* indicates (for a certain warping window δ) the percentage of LCSS computations conducted by the index compared to linear scan

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 space overhead of the index for the 5% splits does not exceed one quarter of the original trajectory disk size (Fig. 27). Therefore, a small increase in the storage space can lead to a significant reduction in the search response time.

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; the results are presented in Fig. 23. We observe that with 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 the question arises as to 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

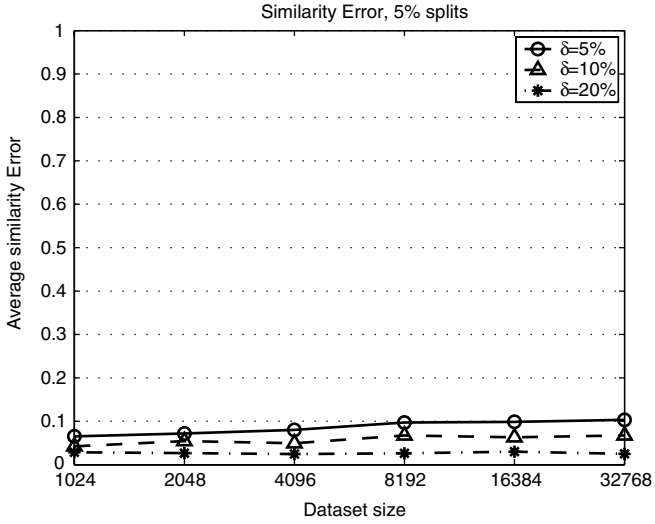

Fig. 25 Using the V-similarity estimate we can retrieve answers faster with very high accuracy. The LCSS similarity is very close (2–10%) to the exact answer returned by the sequential scan

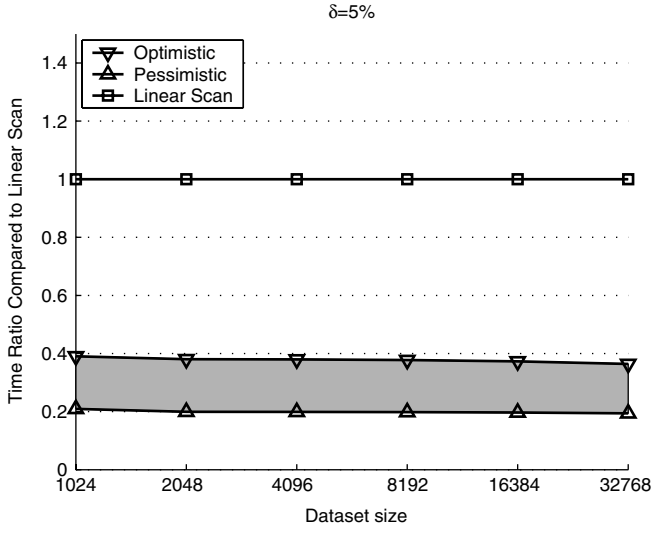

Fig. 26 Index performance using DTW as the distance measure ($\delta = 5\%$). We can observe up to five times faster speedup

$|q|$. Therefore, the *average similarity error* (ASE) is:

$$\text{ASE} = \frac{1}{|q|}\sum_{i=1}^{|q|}(|\text{BestMatch}_{\text{index}} - \text{BestMatch}_{\text{exhaustive}}|)\,.$$

The results are shown in Fig. 25. We notice that the nearest neighbor results returned by the index using the V-similarity estimates are typically within 5% of the LCSS similarity returned by the sequential scan of the data.

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

10 Test cases

In this final section we showcase the usefulness of the proposed similarity measures and indexing scheme on several real-world applications.

10.1 Automatic transcription of handwritten manuscripts

The Library of Congress maintains more than 54 million manuscripts, and there is an imperative need for preserving these historical documents (Fig. 28). Storing these manuscripts in image format is unrealistic, since the conversion to text alone would result in more than 20 terabytes of data. Therefore, there is increasing interest in automatically transcribing these documents. However, optical character

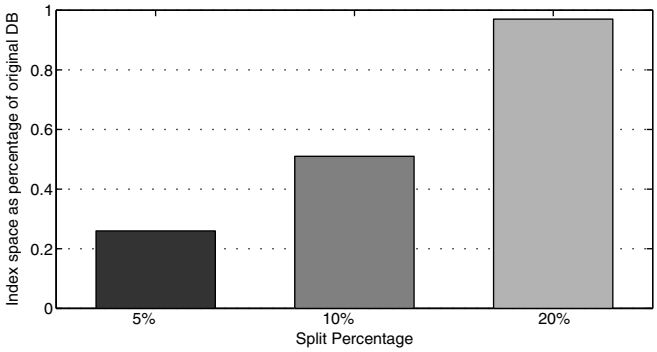

Fig. 27 Index size compared to dataset size

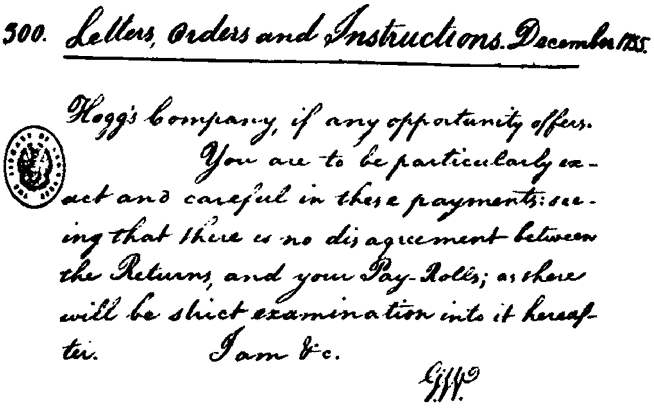

Fig. 28 Part of a George Washington manuscript in the Library of Congress

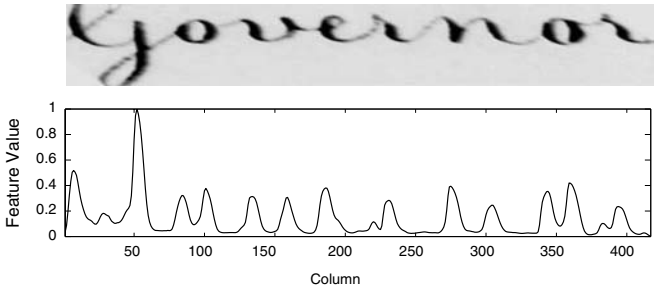

Fig. 29 *Top*: The word "governor" extracted from a George Washington manuscript. *Bottom*: One possible time-series feature extracted from every column of the image (sum of intensity values)

recognition techniques (OCR) cannot be fruitfully utilized in this case due to the handwritten nature of these documents and 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 [46]. 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.

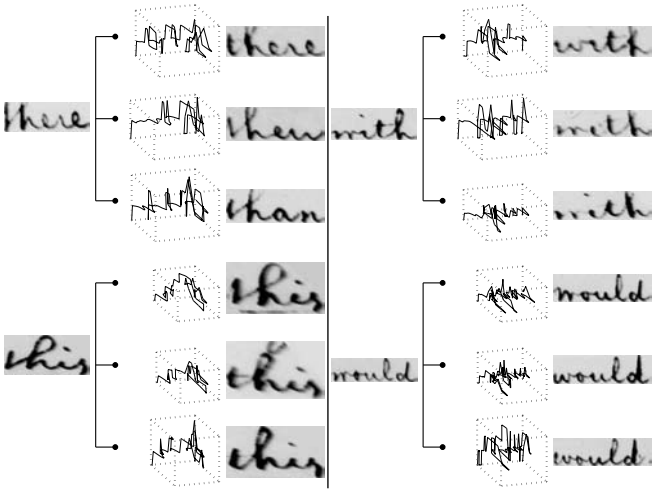

Fig. 30 Manuscript word annotation. 3NN matches based on the extracted time-series features, utilizing the multidimensional DTW as the distance measure

Such features can include the sum of intensity values per column, the ink/paper transitions per column, etc. (Fig. 29). (iv) Manual annotations of a word subset are produced. (v) Classification of the remaining words based on the annotated ones is performed.

The features that are extracted are stored as multidimensional time series. After the image subset is annotated, the remaining images can be classified/annotated using multidimensional distance (similarity) measures of DTW (or LCSS). In Fig. 30 we see the results of a 3NN search for a variety of words, produced using the techniques presented 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 reflect the use of the DTW as the distance function, but we obtained similar results with the LCSS. 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 3D 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 growing demand for content-based retrieval of motion sequences without using annotations or other metadata. 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.).

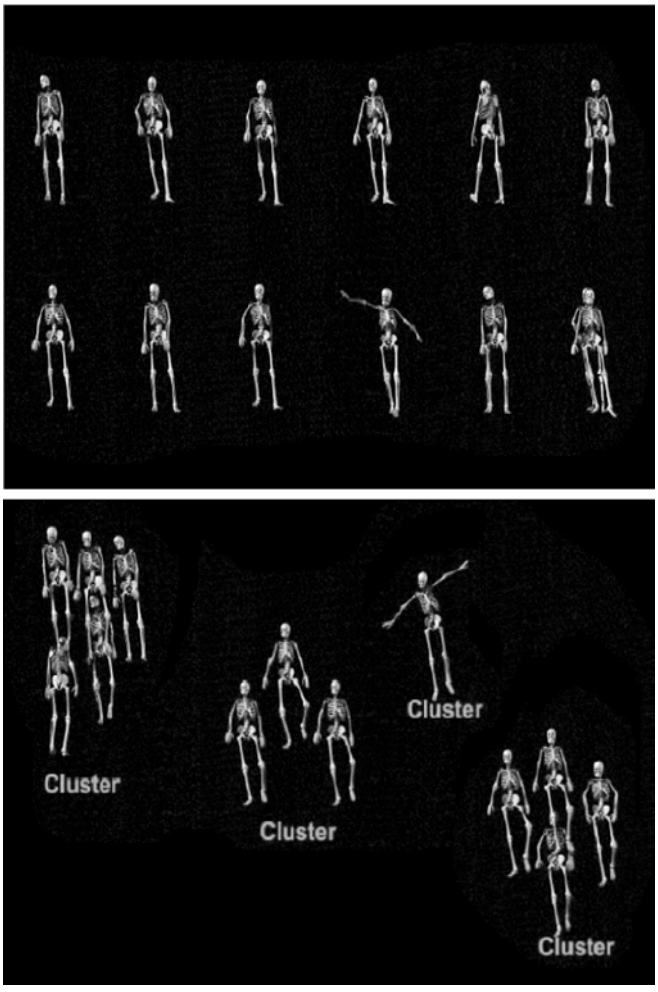

Fig. 31 Some matching results in the motion-capture database for the query: "Find all walking motions." The final results can be grouped into similar motions using a hierarchical clustering algorithm and presented to the animator in a more meaningful manner

The first step in 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. For maximum usability, the mocap matching engine must provide fast response to user queries over extended unlabeled mocap sequences while allowing for spatial and temporal deviations in the returned matches.

Using the techniques described in this paper, each motion is split up into multidimensional 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 Fig. 31. 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.

11 Conclusions

In this paper we have presented an external memory indexing method for discovering similar multidimensional time-series. The most striking novelty of our approach is the presentation of an indexing scheme that 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 nonmetric functions, as well as a tool for understanding better their strengths and weaknesses.

Acknowledgements We would like to thank Margrit Betke for providing 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 word dataset.

The research of Dimitrios Gunopulos was supported by NSF (Grants 9984729, 0220148, 0330481). The research of Eamonn Keogh was supported by NSF Career Award IIS-0237918-001.

References

1. Aach, J., Church, G.: Aligning gene expression time series with time warping algorithms. Bioinformatics **17**, 495–508 (2001)
2. Agrawal, R, Faloutsos, C, Swami, A: Efficient similarity search in sequence databases. In: Proc. of the 4th FODO, pp. 69–84 (October 1993)
3. Agrawal, R., Lin, K., Sawhney, H.S., Shim, K.: Fast similarity search in the presence of noise, scaling and translation in time-series databases. In: Proc. of VLDB, pp. 490–501 (September 1995)
4. Arikan, O., Forsyth, D.: Interactive motion generation from examples. In: Proc. of ACM SIGGRAPH (2002)
5. Bollobás, B., Das, G., Gunopulos, D., Mannila, H.: Time-series similarity problems and well-separated geometric sets. In: Proc. of the 13th SCG (1997)
6. Bar-Joseph, Z., Gerber, G., Gifford, D., Jaakkola, T., Simon, I.: A new approach to analyzing gene expression time series data. In: Proc. of 6th RECOMB, pp. 39–48 (2002)
7. Barbara, D.: Mobile computing and databases—a survey. In: IEEE TKDE, pp. 108–117 (January 1999)
8. Berndt, D., Clifford, J.: Using dynamic time warping to find patterns in time series. In: Proc. of AAAI-94 Workshop of SIGKDD (1994)
9. Betke, M., Gips, J., Fleming, P.: The camera mouse: visual tracking of body features to provide computer access for people with severe disabilities. IEEE Trans. Neural Syst. Rehabil. Eng. **10**(1) (2002)
10. Bozkaya, T., Yazdani, N., Ozsoyoglu, M.: Matching and indexing sequences of different lengths. In: Proc. of the CIKM (1997)
11. Cardle, M., Vlachos, M., Brooks, S., Keogh, E., Gunopulos, D.: Fast motion capture matching with replicated motion editing. In: 29th ACM SIGGRAPH, Sketches and Applications (2003)

12. Chu, K., Wong, M.: Fast time-series searching with scaling and shifting. ACM Principles of Database Systems, pp. 237–248 (June 1999)
13. Chudova, D., Gaffney, S., Mjolsness, E., Smyth, P.: Translation-invariant mixture models for curve clustering. In: Proc. of 9th SIGKDD, pp. 79–88 (2003)
14. Das, G., Gunopulos, D., Mannila, H.: Finding similar time series. In: Proc. of the 1st PKDD Symposium, pp. 88–100 (1997)
15. de Boor, C.: A Practical Guide to Splines. Springer, Berlin Heidelberg New York (1978)
16. Faloutsos, C., Jagadish, H., Mendelzon, A., Milo, T.: Signature technique for similarity-based queries. In: SEQUENCES 97 (1997)
17. Faloutsos, C., Lin, K.-I.: FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In: Proc. of ACM SIGMOD, pp. 163–174 (May 1995)
18. Faloutsos, C., Ranganathan, M., Manolopoulos, I.: Fast subsequence matching in time series databases. In: Proc. of ACM SIGMOD, pp. 419–429 (1994)
19. Gaffney, S., Smyth, P.: Trajectory clustering with mixtures of regression models. In: Proc. of ACM SIGKDD, pp. 63–72 (1999)
20. Gavrila, D., Davis, L.: 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. Ge, X., Smyth, P.: Deformable markov model templates for time-series pattern matching. In: Proc. of ACM SIGKDD (2000)
22. Goldin, D., Kanellakis, P.: On similarity queries for time-series data. In: Proc. of Principles and Practice of Constraint Programming (September 1995)
23. Gollmer, K., Posten, C.: Detection of distorted pattern using dynamic time warping algorithm and application for supervision of bioprocesses. On-Line Fault Detection and Supervision in Chemical Process Industries (1995)
24. Grumbach, S., Rigaux, P., Segoufin, L.: Manipulating interpolated data is easier than you thought. In: Proc. of VLDB, pp. 156–165 (2000)
25. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proc. of ACM SIGMOD, pp. 47–57 (1984)
26. Hadjieleftheriou, M., Kollios, G., Tsotras, V., Gunopulos, D.: Efficient indexing of spatiotemporal objects. In: Proc. of 8th EDBT, pp. 251–268 (2002)
27. Jagadish, H.V., Mendelzon, A.O., Milo, T.: Similarity-based queries. In: Proc. of the 14th ACM PODS, pp. 36–45 (1995)
28. Kahveci, T., Singh, A., Gürel, A.: Similarity searching for multi-attribute sequences. In: Proc. of SSDBM (2002)
29. Kahveci, T., Singh, A.K.: Variable length queries for time series data. In: Proc. of IEEE ICDE, pp. 273–282 (2001)
30. Keogh, E.: Exact indexing of dynamic time warping. In: Proc. of VLDB, pp. 406–417 (2002)
31. Keogh, E., Chakrabarti, K., Mehrotra, S., Pazzani, M.: Locally adaptive dimensionality reduction for indexing large time series databases. In: Proc. of ACM SIGMOD, pp. 151–162 (2001)
32. Keogh, E., Kasetty, S.: On the need for time series data mining benchmarks: a survey and empirical demonstration. In: Proc. of SIGKDD, pp. 102–111 (2002)
33. Kim, S., Park, S., Chu, W.: An index-based approach for similarity search supporting time warping in large sequence databases. In: Proc. of IEEE ICDE, pp. 607–614 (2001)
34. Kovács-Vajna, Z.: A fingerprint verification system based on triangular matching and dynamic time warping. IEEE Trans Pattern Anal Mach Intell, **22**(11), (2000)
35. Lee, S.-L., Chun, S.-J., Kim, D.-H., Lee, J.-H., Chung, C.-W.: Similarity search for multidimensional data sequences. In: Proc. of IEEE ICDE, pp. 599–608 (2000)
36. Levenshtein, V.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics—Doklady **10**(10), 707–710 (1966)
37. Munich, M., Perona, P.: Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In: 7th International Conference on Computer Vision, pp. 108–115 (1999)
38. Park, S., Chu, W., Yoon, J., Hsu, C.: Efficient searches for similar subsequences of different lengths in sequence databases. In: Proc. of IEEE ICDE, pp. 23–32 (2000)
39. Perng, S., Wang, H., Zhang, S., Parker, D.S.: Landmarks: A new model for similarity-based pattern querying in time series databases. In: Proc. of IEEE ICDE, pp. 33–42 (2000)
40. Pfoser, D., Jensen, C.S.: Capturing the uncertainty of moving-object representations. Lecture Notes in Computer Science, vol 1651 (1999)
41. Pratt, K.B.: Locating patterns in discrete time-series. Master's thesis (2001)
42. Qu, Y., Wang, C., Wang, X.: Supporting fast search in time series for movement patterns in multiple scales. In: Proc. of ACM CIKM, pp. 251–258 (1998)
43. Rafiei, D., Mendelzon, A.: Querying time series data based on similarity. In: IEEE Trans. Knowl. Data Eng., **12**(5), 675–693 (2000)
44. Ratanamahatana, C.A., Keogh, E.: Everything you know about dynamic time warping is wrong. In: 3rd Workshop on Mining Temporal and Sequential Data (SIGKDD), 2004
45. Ratanamahatana, C.A., Keogh, E.: Making time-series classification more accurate using learned constraints. In: Proc. of SIAM International Conference on Data Mining (SDM) (2004)
46. Rath, T., Manmatha, R.: Word image matching using dynamic time warping. Tec Report MM-38. Center for Intelligent Information Retrieval, University of Massachusetts Amherst (2002)
47. Roddick, J.F., Hornsby, K.: Temporal, spatial and spatio-temporal data mining. TSDM (2000)
48. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: Proc. of ACM SIGMOD (1995)
49. Shimada, M., Uehara, K.: Discovery of correlation from multi-stream of human motion. Discovery Sci., pp. 290–294 (2000)
50. Strik, H., Boves, L.: Averaging physiological signals with the use of a dtw algorithm. In: SPEECH'88, 7th FASE Symposium, Book 3, pp. 883–890 (1988)
51. Valdes-Perez, R.E., Stone, C.A.: Systematic detection of subtle spatio-temporal patterns in time-lapse imaging: II. Particle migrations. Bioimaging **6**(2), 71–78 (1998)
52. Vlachos, M., Kollios, G., Gunopulos, D.: Discovering similar multidimensional trajectories. In: Proc. of IEEE ICDE, pp. 673–684 (2002)
53. Yi, B.-K., Faloutsos, C.: Fast time sequence indexing for arbitrary Lp norms. In: Proc. of VLDB, pp. 385–394 (2000)
54. Yi, B.-K., Jagadish, H.V., Faloutsos, C.: Efficient retrieval of similar time sequences under time warping. In: Proc. of IEEE ICDE, pp. 201–208 (1998)
55. Zhu, Y., Shasha, D.: Warping indexes with envelope transforms for query by humming. In: Proc. of ACM SIGMOD, pp. 181–192 (2003)