
Machine Learning, 58, 301–334, 2005

Elastic Translation Invariant Matching of Trajectories

MICHAIL VLACHOS
IBM T.J. Watson Research Center, 19 Skyline Dr, Hawthorne, NY 10532

GEORGE KOLLIOS
Department of Computer Science, Boston University, 111 Cummington St, Boston, MA 02215

DIMITRIOS GUNOPULOS
Department of Computer Science and Engineering, University of California, Riverside, CA 92521

Editor: Eamonn Keogh

Abstract. We investigate techniques for analysis and retrieval of object trajectories. We assume that a trajectory is a sequence of two or three dimensional points. Trajectory datasets are very common in environmental applications, mobility experiments, video surveillance and are especially important for the discovery of certain biological patterns. Such kind of data usually contain a great amount of noise, that makes all previously used metrics fail. Therefore, here we formalize non-metric similarity functions based on the Longest Common Subsequence (LCSS), which are very robust to noise and furthermore provide an intuitive notion of similarity between trajectories by giving more weight to the similar portions of the sequences. Stretching of sequences in time is allowed, as well as global translating of the sequences in space. Efficient approximate algorithms that compute these similarity measures are also provided. We compare these new methods to the widely used Euclidean and Dynamic Time Warping distance functions (for real and synthetic data) and show the superiority of our approach, especially under the strong presence of noise. We prove a weaker version of the triangle inequality and employ it in an indexing structure to answer nearest neighbor queries. Finally, we present experimental results that validate the accuracy and efficiency of our approach.

Keywords: longest common sequence, time warping, time-series

1. Introduction

In this paper we investigate the problem of discovering similar trajectories of moving objects. The trajectory of a moving object is typically modeled as a sequence of consecutive locations in a multidimensional (generally two or three dimensional) Euclidean space. Such data types arise in many applications where the location of a given object is measured repeatedly over time. Examples include features extracted from video clips, animal mobility experiments, sign language recognition, motion-capture data, mobile phone usage, multiple attribute response curves in drug therapy, and so on.

Moreover, the recent advances in mobile computing, sensor and GPS technology have made it possible to collect large amounts of spatio-temporal data and there is increasing

interest to perform data analysis tasks over this data (Barbará, 1999). For example, in mobile computing, users equipped with mobile devices move in space and register their location at different time instants via wireless links to spatio-temporal databases. 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. Data analysis in such data include determining and finding objects that moved in a similar way or followed a certain motion pattern. An appropriate and efficient model for defining the similarity for trajectory data will be very important for the quality of the data mining tasks.

1.1. Robust distance metrics for trajectories

Typical trajectory data are obtained during a tracking procedure, with the aid of various sensors. Here also lies the main obstacle of such data; they may contain a significant amount of *outliers* (see figure 1) or in other words incorrect data measurements (unlike for example, stock data which contain no errors whatsoever).

One of our objectives is the automatic classification of trajectories using a Nearest Neighbor Classifier. It has been shown that the one nearest neighbor rule has asymptotic error rate that is at most twice the Bayes error rate (Duda & Hart, 1973). Therefore, the problem that must be addressed is the following: given a database $\mathcal{D}$ of trajectories and a query $\mathcal{Q}$ (not already in the database), we want to find the trajectory $\mathcal{T}$ that is closest to $\mathcal{Q}$. In order to answer this query, we have to: (i) define a realistic distance function and (ii) design an indexing scheme for answering the nearest neighbor query efficiently.

Previous approaches to model the similarity between time-series include the use of the *Euclidean* and the *Dynamic Time Warping (DTW)* distance, which however are relatively sensitive to noise. Distance functions that are robust to extremely noisy data will typically violate the triangular inequality. These functions achieve this by not considering the most

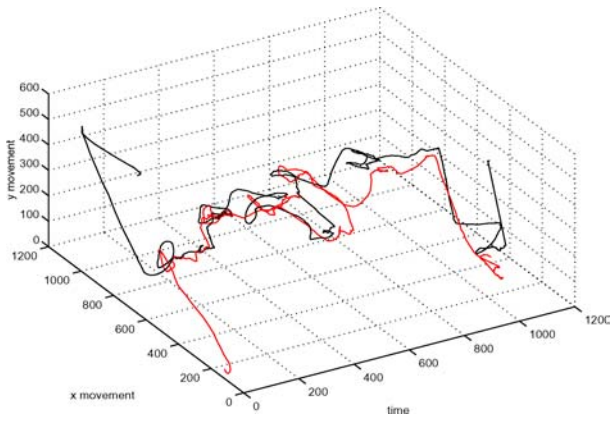

Figure 1. Examples of 2D trajectories. Two instances of video-tracked time-series data representing the word 'athens'. Start and ending contain many outliers.

dissimilar parts of the objects. However, they are very useful because they represent an accurate model of the human perception (Medin, Goldstone, & Gentner, 1993), since when comparing any kind of data (images, trajectories etc), we mostly focus on the portions that are similar and we are willing to pay less attention to regions of great dissimilarity.

For this kind of data we need distance functions that can address the following issues:

- *Different Sampling Rates or different speeds.* The time-series that we obtain, are not guaranteed to be the outcome of sampling at fixed time intervals. The sensors collecting the data may fail for some period of time, leading to inconsistent sampling rates. Moreover, two time series moving at exactly the similar way, but one moving at twice the speed of the other will result (most probably) to a very large Euclidean distance.
- *Similar motions in different space regions*. Objects can move similarly, but differ in the space they move. This can easily be observed in sign language recognition, if the camera is centered at different positions. If we work in Euclidean space, usually subtracting the average value of the time-series, will move the similar series closer.
- *Outliers.* Might be introduced due to anomaly in the sensor collecting the data or can be attributed to human 'failure' (e.g. jerky movement during a tracking process). In this case the Euclidean distance will completely fail and result to very large distance, even though this anomaly may be found in only a few points.
- *Different lengths.* Euclidean distance deals with time-series of equal length. In the case of different lengths we have to decide whether to truncate the longer series, or pad with zeros the shorter etc. In general its use gets complicated and the distance notion more vague.
- *Efficiency.* It has to be adequately expressive but sufficiently simple, so as to allow efficient computation of the similarity.

To cope with these challenges we use the Longest Common Subsequence (LCSS) model. The LCSS is a variation of the edit distance. The basic idea is to match two sequences by allowing them to stretch, without rearranging the sequence of the elements but allowing some elements to be *unmatched*. The advantages of the LCSS method are twofold:

(1) Some elements may be unmatched, where in Euclidean and DTW *all* elements must be matched, even the outliers.
(2) The LCSS model allows a more efficient approximate computation, as will be shown later (whereas in DTW you need to compute a costlier L_p Norm).

In figure 2 we can see the clustering produced by the *Euclidean* and *DTW* distance. The sequences represent data collected through a video tracking process. Originally they represent 2D series, but only one dimension is depicted here for clarity. The hierarchical clustering on the pairwise *DTW* distances, fails to distinguish the two classes of words, due to the great amount of outliers, especially in the beginning and in the end of the trajectories. Using the Euclidean distance we obtain even worse results because it cannot

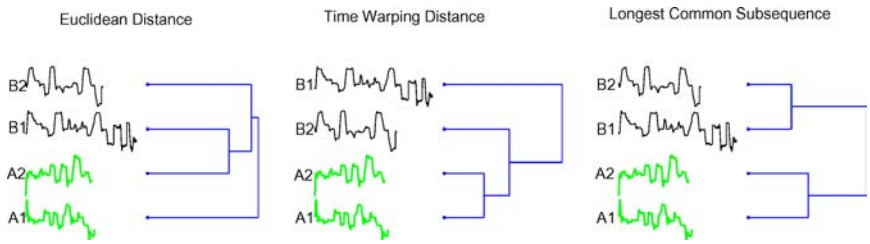

Figure 2. Hierarchical clustering of 2D series (displayed as 1D for clarity). Four video-tracked time-series representing 2 instances of the words *Athens*(A1 and A2) and *Boston* (B1 and B2) are used for this example. The presence of many outliers in the beginning and the end of the sequences leads to incorrect clustering for the Euclidean and the DTW distance. The *LCSS* focusing on the common parts achieves the correct clustering.

support compression in time. The *LCSS* produces the most intuitive clustering as shown in the same figure. Generally, the Euclidean distance is very sensitive to small variations in the time axis, while the major drawback of the *DTW* is that it has to pair all elements of the sequences (see also figure 3).

Therefore, we use the *LCSS* model to define similarity measures for trajectories. Nevertheless, a simple extension of this model into 2 or more dimensions is not sufficient, because (for example) this model cannot deal with parallel movements. Therefore, we extend it in order to address similar problems. In our similarity model we consider a set of translations in 2 or more dimensions and we find the translation that yields the optimal solution to the *LCSS* problem.

The rest of the paper is organized as follows. In Sections 2 and 3 we formalize the new similarity functions by extending the *LCSS* model, while in Section 4 we empirically

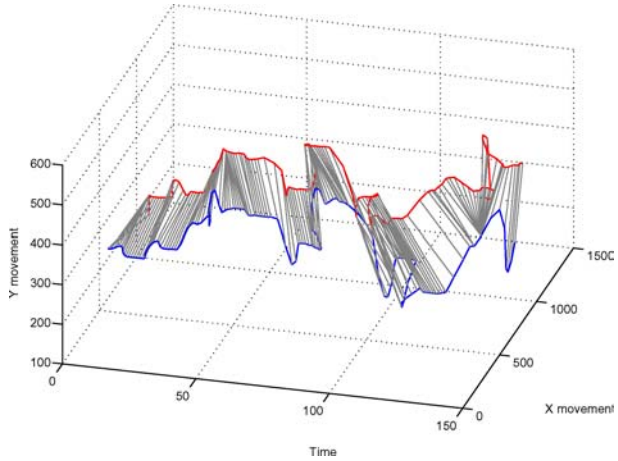

Figure 3. The support of flexible matching in spatio-temporal queries is very important. However, we can observe that Dynamic Time Warping matches all points (so the outliers as well), therefore distorting the true distance. In contrast, the LCSS model can efficiently ignore the noisy parts.

justify our choice for constraining the warping region. Section 5 demonstrates efficient algorithms to compute these functions and Section 6 elaborates on the indexing structure. Section 7 provides the experimental validation of the accuracy and efficiency of the proposed approach and Section 8 presents the related work. Finally, Section 9 concludes the paper.

2. Similarity measures

In this section we define similarity models that match the user's perception of similar trajectories. First we provide some definitions and then we proceed by presenting the similarity functions based on the appropriate models. We assume that objects are points that move on the (x, y)-plane and time is discrete. In Table 1 we summarize the notation used throughout the paper.

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 $Head(A)$ be the sequence $Head(A) = ((a_{x,1}, a_{y,1}), \ldots, (a_{x,n-1}, a_{y,n-1}))$.

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

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

Table 1. Notation used in the paper.

A (or B)	2-dimensional trajectories
$Head(A)$	trajectory A without its last element
A_i (or B_i)	(x, y) position at time i of trajectory A (or B)
δ	Allowed temporal matching *(maximum warping length)*
ϵ	Allowed spatial matching
$\mathcal{F}$	Set of translations in the x-y plane
$f_{c,d}$	x-translation by c, y-translation by d
$S1$	LCSS similarity for real values
$S2$	LCSS similarity with translation
$S3$	LCSS sigmoidal similarity
$D1$	Distance based on S1 similarity
$D2$	Distance based on S2 similarity

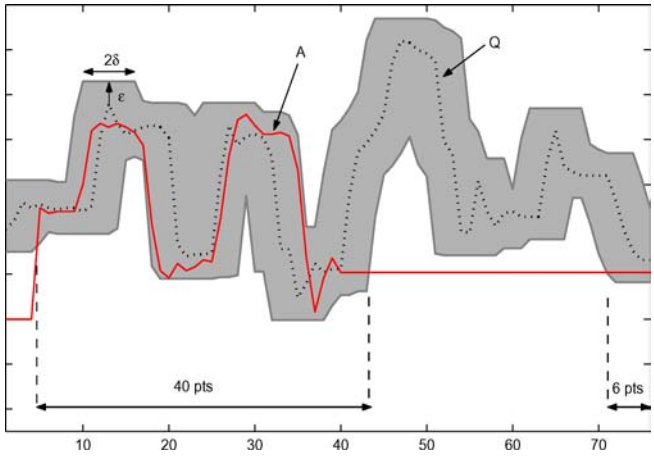

Figure 4. The notion of the *LCSS* matching within a region of δ & ϵ for a trajectory. The points of the 2 trajectories within the gray region can be matched by the *LCSS* function.

The constant δ controls how far in time we can go in order to match a given point from one trajectory to a point in another trajectory. The constant ϵ is the matching threshold (see figure 4).

The first similarity function is based on the *LCSS* and the idea is to allow time stretching. Then, objects that are close in space at different time instants can be matched if the time instants are also close.

Definition 2. We define the similarity function $S1$ between two trajectories A and B, given δ and ϵ, as follows:

$$S1(\delta, \epsilon, A, B) = \frac{LCSS_{\delta,\epsilon}(A, B)}{\min(n, m)}$$

The division by the length of the sequence in $S1$, serves the purpose of comparing the LCSS value between sequences of different lengths.

Now we use this $S1$ function to define another similarity measure that is more suitable for trajectories. First, we consider the set of translations. A translation simply shifts a trajectory in space by a different constant in each dimension. Let $\mathcal{F}$ be the family of translations. Then a function $f_{c,d}$ belongs to $\mathcal{F}$ if $f_{c,d}(A) = ((a_{x,1} + c, a_{y,1} + d), \ldots, (a_{x,n} + c, a_{y,n} + d))$. Next, we define a second notion of the similarity based on the above family of functions.

Definition 3. Given δ, ϵ and the family $\mathcal{F}$ of translations, we define the similarity function $S2$ between two trajectories A and B, as follows:

$$S2(\delta, \epsilon, A, B) = \max_{f_{c,d} \in \mathcal{F}} S1(\delta, \epsilon, A, f_{c,d}(B))$$

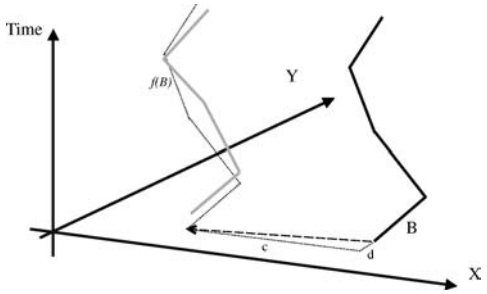

Figure 5. Translation of trajectory B.

So the similarity functions $S1$ and $S2$ range from 0 to 1. Therefore we can define the distance function between two trajectories as follows:

Definition 4. Given δ, ϵ and two trajectories A and B we define the following distance functions:

$$D1(\delta, \epsilon, A, B) = 1 - S1(\delta, \epsilon, A, B)$$

and

$$D2(\delta, \epsilon, A, B) = 1 - S2(\delta, \epsilon, A, B)$$

Note that $D1$ and $D2$ are *symmetric*. $LCSS_{\delta,\epsilon}(A, B)$ is equal to $LCSS_{\delta,\epsilon}(B, A)$ and the transformation that we use in $D2$ is a translation which preserves the symmetric property.

By allowing translations, we can detect similarities between movements that are parallel in space, but not identical. In addition, the *LCSS* model allows stretching and displacement in time, so we can detect similarities in movements that happen with different speeds, or at different times. In figure 5 we show an example where a trajectory B matches another trajectory A after a translation is applied. Note that the value of parameters c and d are also important since they give the distance of the trajectories in space. This can be useful information when we analyze trajectory data.

The similarity function S_2 is a significant improvement over the S_1, because: (i) now we can detect parallel movements, (ii) the use of *normalization* does not guarantee that we will get the best match between two trajectories. Usually, because of the significant amount of noise, the average value and/or the standard deviation of the time-series, that are being used in the normalization process, can be distorted leading to improper translations.

3. Sigmoidal matching function

Using the previous LCSS model we perform *unweighted* matching. Whenever the points of two trajectories lie within ϵ in space the similarity is increased by one. This however

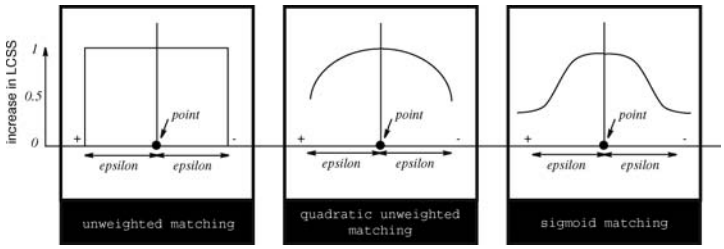

Figure 6. Unweighted and weighted matching options for LCSS model.

penalizes the points that were marginally outside the matching region (assigning to them a value of zero) and also makes the choice of ϵ an important issue. Therefore, one can devise a weighted matching scheme that will give more significance to closer points and less to distant points. The importance given to each matching point decreases gradually up to a certain cutoff distance, therefore efficiently pruning the outliers (which will be given a similarity of zero). Figure 6 depicts on the left the regular unweighted matching function presented in the previous sections. Possible weighted functions are also shown, while in the text that follows we delineate how the sigmoidal function can be used to enhance the matching quality of the *LCSS*.

Using the *LCSS* paradigm one can allow for time compression and decompression. However now for the matching function between two trajectories A and B we use a sigmoidal function and we set its matching width equal to $\min(\text{std}(A), \text{std}(B))$, where std denotes the standard deviation of a sequence's values. Now all points with distance greater than the matching threshold will not be matched (essentially the outliers), but the weighted approach will help to reveal and accurately capture even small motion discriminations between two sequences. A sigmoidal function has the form:

$$s(x) = 1/\left(1 + e^{-a(x-k-1)}\right), \quad x = 1 \ldots 2k+1$$

One can use different parameters for the variable a, in the evaluation of function $s(\cdot)$, in order to indicate where more matching weight should be given. In figure 7 we can observe the different degree of similarity assigned between two points, for different values of a. For our experiments we used a value of $a = 0.25$. The value of k essentially controls the accuracy (more bins evaluate more accurately the sigmoidal value).

The weighted distance between two 2D points $A_i = (a_{x,i}, a_{y,i})$, $B_j = (b_{x,j}, b_{y,j})$ belonging to sequences A and B, respectively, is given by the *SigmoidMatch* function:

$$\begin{aligned} &\textit{SigmoidMatch}(A_i, B_j) \\ &= \begin{cases} 0, & \text{if } |a_{p,i} - b_{p,i}| > \min(\text{std}(A_p), \text{std}(B_p)), \quad p = x, y \\ \frac{1}{2}\sum_{p=x}^{y} s\left(\lceil \dfrac{|(a_{p,i}, b_{p,i})|}{\min(\text{std}(A_p), \text{std}(B_p))} \cdot (2k+1) \rceil\right), & \text{otherwise} \end{cases} \end{aligned}$$

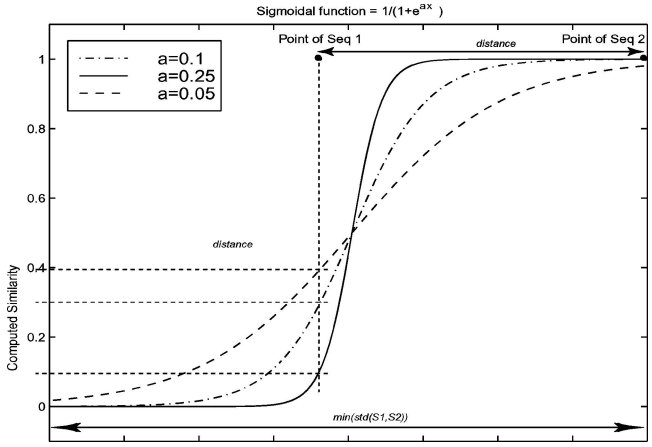

Figure 7. The *SigmoidMatch* function for different parameters and the weighted matching procedure.

where $A_x = (a_{x,1}, a_{x,2}, \ldots, a_{x,n})$, $A_y = (a_{y,1}, a_{y,2}, \ldots, a_{y,n})$ and similarly are defined sequences B_x, B_y.

The sigmoid based similarity function between two trajectories is now defined as:

Definition 5. Given an integer δ, we define the $SigmoidSim_\delta(A, B)$ as follows:

$$\begin{aligned} SigmoidSim_\delta(A, B) = {} & SigmoidMatch(A_n, B_m) + \\ & \max\{SigmoidSim(Head(A), Head(B)), \\ & \quad SigmoidSim(Head(A), B), \\ & \quad SigmoidSim(A, Head(B))\} \\ & \text{where, } \mid n - m \mid \leq \delta \end{aligned}$$

It should be noted, that under the new similarity definition we have essentially eliminated the matching parameter ϵ. The constant δ defines again the allowed flexibility for time compression and decompression. An example of the sigmoidal matching (disregarding the matching within δ) is depicted in figure 8, while the length normalized function is defined below.

Definition 6. We define the similarity $S3$ between two trajectories A and B, given δ, as follows:

$$S3(\delta, A, B) = \frac{SigmoidSim_\delta(A, B)}{\min(n, m)}$$

In the experiments we will demonstrate the enhanced accuracy that can be achieved using the sigmoidal based similarity function, especially for noisy data.

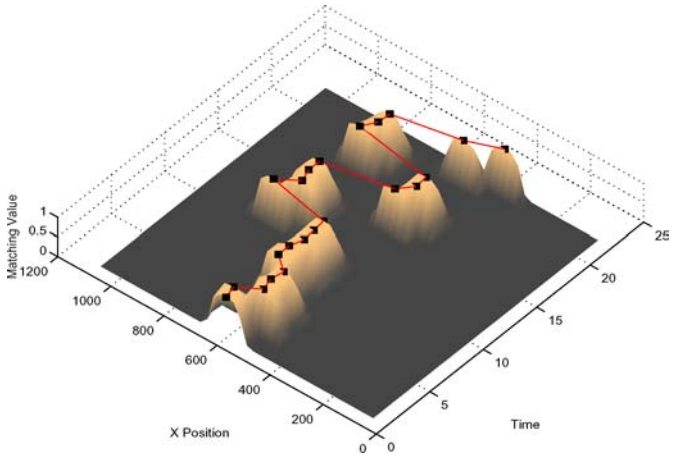

Figure 8. Sigmoidal matching of a sequence (without the time matching within δ). The similarity value at each point of another sequence, declines up to a certain cut-off point.

The advantages of the model S3 are summarized below:

1. Only one parameter is required (parameter δ)
2. More accurate similarities can be captured against the unweighted distance models (that utilize the ϵ matching threshold).

4. Constraining the warping

In all our similarity models we have used a constrained warping window within δ. This significantly improves the run time execution of the algorithm, as will be shown in the next section. However, we will demonstrate (using real datasets as examples), that we have more to gain by constricting the matching window in time:

1. In most datasets there is no need to perform full length warping. In practice time warping around 20% of the sequences' length, proves sufficient for most datasets and applications. We can usually observe convergence in the pairwise similarity between sequences after a certain warping window. Therefore, allowing for larger windows, would not yield changes in the similarity, at the expense of prolonged execution time.

   Using the *Camera Mouse* program (Gips, Betke, & Fleming, 2000), we have obtained multiple instances of various words created by tracking a human feature over time, while utilizing a 'virtual keyboard'. In this manner, we had at our disposal spatiotemporal sequences (figure 9), each one describing a different word. In figure 10 we can observe how the LCSS similarity progresses, over increasing warping windows for various pairwise computations. It is apparent that there are diminishing returns for extensive

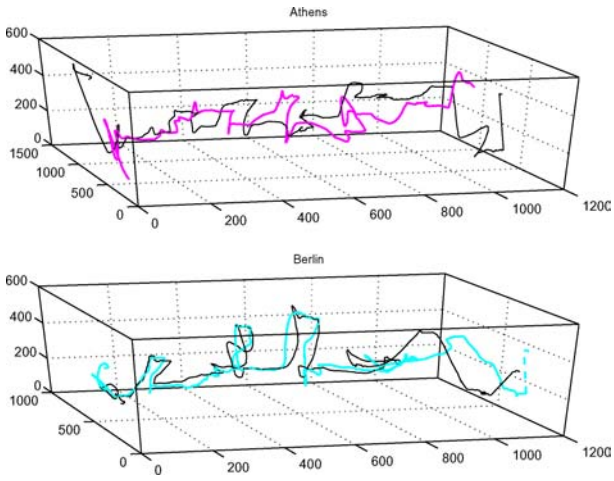

Figure 9. Two words of the cameraMouse dataset (two instances each).

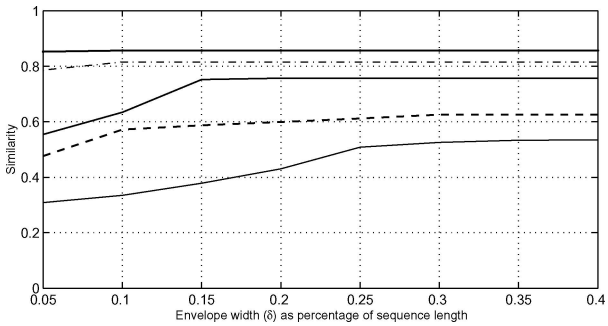

Figure 10. Five pairwise LCSS similarity computations for increasing warping windows (parameter δ). The similarity does not change for warping more than 25% of the sequence's length.

warping, since time warping more than 20% of the sequence's length, rarely changes the computed sequence similarity.

Similar behavior has also been observed by Keogh and Ratanamahatana (2004) for time-series datasets, that span in the areas of botany, video-tracking, text recognition etc. The authors clearly suggest that full warping is simply futile, and does not yield better results (for their datasets ranges of 3–10% warping returned the best results).

2. For certain datasets, extended warping not only may not yield additional gain, but it also may hurt accuracy. We performed the following simple experiment; we tried to label each sequence of some test datasets, using a 'leave-one-out' 1-Nearest-Neighbor classification scheme. Therefore, each time-series was given the label of its nearest neighbor among the remaining sequences. Figure 11 is illustrative of the accuracy results that we have obtained for the *cameraMouse* dataset and for a subset of the *Australian Sign Language*

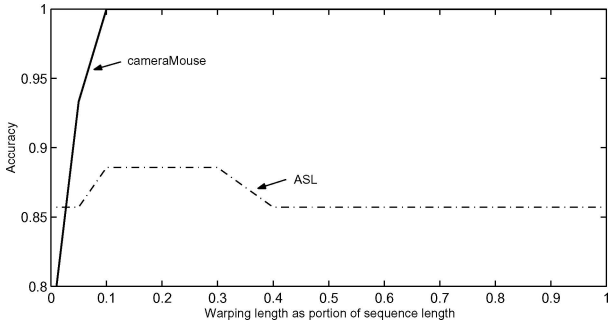

Figure 11. 'Leave-one-out' classification accuracy for two datasets. We observe that excessive warping can hurt the performance in certain datasets.

(ASL) dataset.[1] While for the *cameraMouse* extensive warping does not further assist the accuracy, in the case of the *ASL*, increasing the allowed warping in time to more than 30% actually penalizes the accuracy. This is easily explained, since excessive matching envelopes not only distort the true distance, but also 'force' instances of words from different classes to match with one another, by allowing long and degenerate matching correspondences. Again, similar results that support this claim have also been noted in Ratanamahatana and Keogh (2004) and Keogh and Ratanamahatana (2004).

This discussion denotes that our choice of constraining the warping in time (parameter δ), not only improves the algorithm run time, but it is also a highly desirable feature for providing better similarity estimates and for improving the classification/clustering accuracy.

5. Efficient algorithms to compute the similarity

5.1. *Computing the similarity function* $S1$

To compute the similarity functions $S1$, $S3$ we need to run an *LCSS* computation for the two sequences. The *LCSS* can be computed by a dynamic programming algorithm in $O(n^2)$ time. However we only allow matchings when the difference in the indices is at most δ, and this allows the use of a faster algorithm. The following lemma has been shown in Berndt and Clifford (1994) and Das, Gunopulos, and Mannila (1997).

Lemma 1. *Given two trajectories* A *and* B, *with* $|A| = n$ *and* $|B| = m$, *we can find the* $LCSS_{\delta,\epsilon}(A, B)$ *in* $O(\delta(n + m))$ *time.*

If δ is small, the dynamic programming algorithm is very efficient (figure 12). However, for some applications δ may need to be large. For that case, we can speed-up the above computation using random sampling.

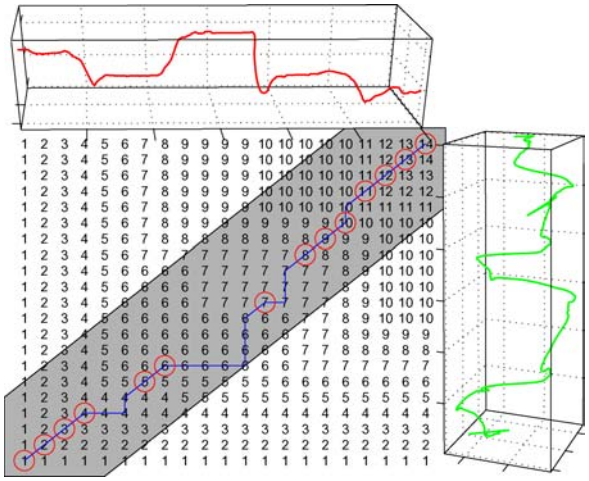

Figure 12. Solution of the LCSS problem for trajectories using dynamic programming. The matching region in time is indicated by the gray area ($\delta = 5$). For this example, the constrained warping solution is the same as the one provided when computing the whole array.

Random sampling has been effectively used in many scientific areas in order to speed up the execution of costly algorithms. Such applications include clustering (Guha, Rastogi, & Shim, 1998), or other data-mining applications (Kivinen & Mannila, 1994; Zaki et al., 1997), where in general we have to examine a large amount of data. By taking a sufficiently small amount of random samples from the original data, we can show that with high probability, the random sample that we collected preserves the properties (shape, structure, average value, etc) of the original population. This can be proven using several tail inequalities, such as the Chebyshev's inequality or the Chernoff bounds. In this work we use the Chernoff bounds.

Lemma 2. *Given two trajectories A and B with length n, two constants δ and ϵ, and a random sample of A, $|RA| = s$, we can compute an approximation of the $LCSS(\delta, \epsilon, A, B)$ such that the approximation error is less than β with probability at least $1 - \rho$, in $O(ns)$.*

Proof: The idea is that for each element a_i of RA we choose the elements from B that are within a range of δ from a_i (so, $b_i - \delta, \ldots, b_i + \delta$). Suppose RB is the set of all such elements from B and $|RB| = \min(2\delta, n)$. Then we compute the $LCSS(\delta, \epsilon, RA, RB)$ and let the result be g. The $LCSS$ is estimated as $g * n/s$. Now, let X_i be a random variable that is 1 if we find a match for a_i in $LCSS(\delta, \epsilon, RA, RB)$ and 0 otherwise. We can assume that all $X_1, X_2, \ldots, X_s$ are independent random variables. Also, let $LCSS(\delta, \epsilon, A, B) = m$, we have that $P(X_j = 1) = m/n$. Thus, the number of the sample points that match is given by the random variable $X = \sum_{j=1}^{s} X_j$. The expected value of X is $\mu = E[X] = E[X = \sum_{j=1}^{s} X_j] = \sum_{j=1}^{s} E[X_j] = \frac{sm}{n}$. Subsequently, using the Chernoff bound: $P[X < (1 - \beta)\mu] < e^{\frac{-\mu\beta^2}{2}}$ and demanding this probability to be smaller than p, we get that: $|s| = \frac{2n}{\beta^2 m} \log(\frac{1}{p})$ □

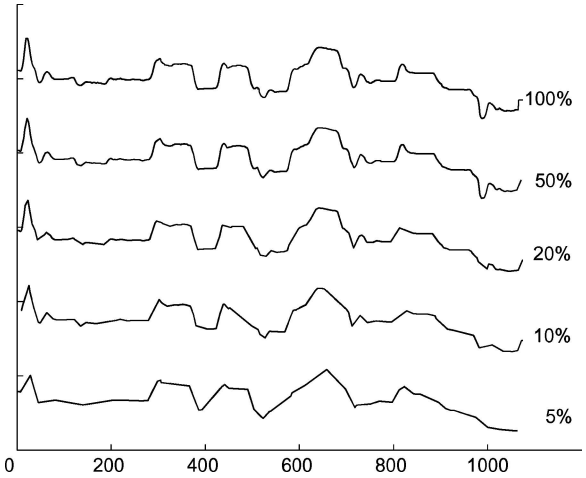

Figure 13. The general shape of a sequence remains the same even for down to 5% sampling.

The above proof states that if we want to be within 0.1 of the true similarity of the trajectories with a probability of 90% and the similarity between the examined trajectories is around 0.8 we should sample the value at 250 positions of the original trajectory. Notice that this number is independent of the length of the sequence. If we want to be able to capture accurately the similarity between less similar trajectories (e.g. with 0.4 similarity) then we should sample at 500 points. Therefore for long trajectories consisting of 10000 points, a sample of 5% is adequate. In practice, we can get very good results even for shorter trajectories and for sampling down to 10% of the original series length. In figure 13 we show how the general shape of the trajectory changes with the sample size. The original trajectory consists of 1074 2d points and in figure we show the x coordinate. The only distortions that we can observe on the sampled trajectory are noticed for samples lower than 10% of the trajectory length.

5.2. *Computing the similarity function S2*

We now consider the more complex similarity function $S2$. Here, given two sequences A, B, and constants δ, ϵ, we have to find the translation $f_{c,d}$ that maximizes the length of the longest common subsequence of A, $f_{c,d}(B)$ ($LCSS_{\delta,\epsilon}(A, f_{c,d}(B))$ over all possible translations.

Let the length of trajectories A and B be n and m respectively. Also assume that the translation f_{c_1,d_1} is the translation that, when applied to B, gives a longest common subsequence: $LCSS_{\delta,\epsilon}(A, f_{c_1,d_1}(B)) = a$, and it is also the translation that maximizes the length of the longest common subsequence:

$$LCSS_{\delta,\epsilon}(A, f_{c_1,d_1}(B)) = \max_{c,d\in\mathcal{R}} LCSS_{\delta,\epsilon}(A, f_{c,d}(B)).$$

The key observation is that, although there is an infinite number of translations that we can apply to B, each translation $f_{c,d}$ results in a longest common subsequence between A and $f_{c,d}(B)$, and there is a finite set of possible longest common subsequences. In this section we show that we can efficiently enumerate a finite set of translations, such that this set provably includes a translation that maximizes the length of the longest common subsequence of A and $f_{c,d}(B)$.

To give a bound on the number of transformations that we have to consider, we look at the projections of the two trajectories on the two axes separately.

We define the x projection of a trajectory $B = ((x_1, y_1), \ldots (x_m, y_m))$ to be the sequence of the values on the x-coordinate: $B_x = (b_{x,1}, \ldots, b_{x,m})$. A one dimensional translation f_c is a function that adds a constant to all the elements of a 1-dimensional sequence: $f_c(x_1, \ldots, x_m) = (x_1 + c, \ldots, x_m + c)$.

Considering the x projections of A and B, A_x and B_x respectively, we can show the following lemma:

Lemma 3. *Given trajectories A, B, if $LCSS_{\delta,\epsilon}(A, f_{c_1,d_1}(B)) = a$, then the length of the longest common subsequence of the one dimensional sequences A_x and $f_{c_1}(B_x) = (b_{x,1} + c_1, \ldots, b_{x,m} + c_1)$, is at least a: $LCSS_{\delta,\epsilon}(A_x, f_{c_1}(B_x)) \geq a$. Also, $LCSS_{\delta,\epsilon}(A_y, f_{d_1}(B_y)) \geq a$.*

Proof: The proof is clear from the fact that any locations that match for the two dimensional longest common subsequence must also match at the projection. □

Now, consider A_x and B_x. A translation by c', applied to B_x can be thought of as a linear transformation of the form $f(b_{x,i}) = b_{x,i} + c'$. Such a transformation will allow $b_{x,i}$ to be matched to all $a_{x,j}$ for which $|i - j| < \delta$, and $a_{x,j} - \epsilon \leq f(b_{x,i}) \leq a_{x,j} + \epsilon$.

It is instructive to view this as a stabbing problem; consider the $O(\delta(n + m))$ vertical line segments $((b_{x,i}, a_{x,j} - \epsilon), (b_{x,i}, a_{x,j} + \epsilon))$, where $|i - j| < \delta$ (figure 14).

These line segments are on a two dimensional plane, where on the x axis we put elements of B_x and on the y axis we put elements of A_x. For every pair of elements $[b_{x,i}, a_{x,j}]$ in A_x and B_x that are within δ positions from each other (and therefore can be matched by the

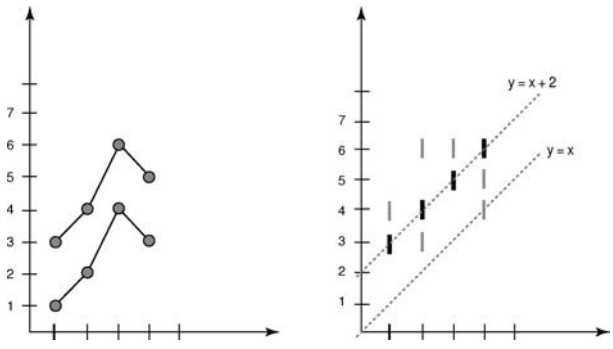

Figure 14. An example of two translations and the transformation into a stabbing problem.

LCSS algorithm if their values are within ϵ), we create a vertical line segment that is centered at the point $(b_{x,i}, a_{x,j})$ and extends ϵ above and below this point. Since each element in A_x can be matched with at most $2\delta + 1$ elements in b_x, the total number of such line segments is $O(\delta n)$.

Since a one dimensional translation $f_{c'}$ is a function of the form $f_{c'}(b_{x,i}) = b_{x,i} + c'$, it can be described on the Euclidean plane as a line of slope 1. After translating B_x by $f_{c'}$, an element $b_{x,i}$ of B_x can be matched to an element $a_{x,j}$ of A_x if and only if the line $f_{c'}(x) = x + c'$ intersects the line segment $((b_{x,i}, a_{x,j} - \epsilon), (b_{x,i}, a_{x,j} + \epsilon))$.

Therefore each line of slope 1 defines a set of possible matchings between the elements of sequences A_x and B_x. The number of intersected line segments is actually an upper bound on the length of the longest common subsequence because the ordering of the elements is ignored. However, two different translations can result to different longest common subsequences only if the respective lines intersect a different set of line segments. For example, the translations $f_0(x) = x + 0$ and $f_2(x) = x + 2$ in figure 14 intersect different sets of line segments and result to longest common subsequences of different length.

The following lemma gives a bound on the number of possible different longest common subsequences, by bounding the number of possible different sets of line segments that are intersected by lines of slope 1.

Lemma 4. *Given two one dimensional sequences A_x, B_x, there are $O(\delta(n + m))$ lines of slope* 1 *that intersect different sets of line segments.*

Proof: Let $f_{c'}(x) = x + c'$ be a line of slope 1. If we move this line slightly to the left or to the right, it still intersects the same number of line segments, unless we cross an endpoint of a line segment. In this case, the set of intersected line segments increases or decreases by one. There are $O(\delta(n + m))$ endpoints. A line of slope 1 that sweeps all the endpoints will therefore intersect at most $O(\delta(n + m))$ different sets of line segments during the sweep. □

In addition, we can enumerate the $O(\delta(n + m))$ translations that produce different sets of potential matchings by finding the lines of slope 1 that pass through the endpoints. Each such translation corresponds to a line $f_{c'}(x) = x + c'$. This set of $O(\delta(n + m))$ translations gives all possible matchings for a longest common subsequence of A_x, B_x. By applying the same process on A_y, B_y we can also find a set of $O(\delta(n + m))$ translations that give all matchings of A_y, B_y. To find the longest common subsequence of the sequences A, B we have to consider only the $O(\delta^2(n + m)^2)$ two dimensional translations that are created by taking the Cartesian product of the translations on x and the translations on y. Since running the LCSS algorithm takes $O(\delta(n + m))$ we have shown the following theorem:

Theorem 1. *Given two trajectories A and B, with $|A| = n$ and $|B| = m$, we can compute the $S2(\delta, \epsilon, A, B)$ in $O((n + m)^3\delta^3)$ time.*

5.3. *An efficient approximate algorithm*

Theorem 1 gives an exact algorithm for computing $S2$, but this algorithm runs in cubic time. In this section we present a much more efficient approximate algorithm. The key to our technique is that we can bound the difference between the sets of line segments, that are intersected by different lines of slope 1, based on how far apart the lines are.

Consider again the one dimensional projections A_x, B_x. Let us consider the $O(\delta(n+m))$ translations that result in different sets of intersected line segments. Each translation is a line of the form $f_{c'}(x) = x + c'$. Let us sort these translations by c'. For a given translation $f_{c'}$, let $L_{f_{c'}}$ be the set of line segments it intersects. The following lemma shows that 'neighboring' translations intersect similar sets of line segments.

Lemma 5. *Let $f_1(x) = x + c'_1, \ldots, f_N(x) = x + c'_N$ be the different translations for sequences A_x and B_x, where $c'_1 \leq \cdots \leq c'_N$. Then the symmetric difference $L_{f_i} \Delta L_{f_j} = |i - j|$.*

Proof: Without loss of generality, assume $i < j$. The difference between f_i and f_{i+1} is simply that we cross one additional endpoint. This means that we have to either add or subtract one line segment to L_{f_i} to get $L_{f_{i+1}}$. By repeating this step $(j - i)$ times we create L_{f_j}. □

We can now prove our main theorem:

Theorem 2. *Given two trajectories A and B, with $|A| = n$ and $|B| = m$, and a constant $0 < \beta < 1$, we can find an approximation $AS2_{\delta,\beta}(A, B)$ of the similarity $S2(\delta, \epsilon, A, B)$ such that $S2(\delta, \epsilon, A, B) - AS2_{\delta,\beta}(A, B) < \beta$ in $O((m + n)\delta^3/\beta^2)$ time.*

Proof: Let $a = S2(\delta, \epsilon, A, B)$. We consider the projections of A and B onto the x and y axes. There exists a translation f_i on the x axis only, such that L_{f_i} is a superset of the matches in the optimal *LCSS* of A and B. In addition, according to the previous lemma, there are $2b$ translations $(f_{i-b}, \ldots, f_{i+b})$ that have at most b different matchings from the optimal.

Therefore, if we use the translations f_{ib}, for $i = 1, \ldots, \lceil \frac{2\delta(n+m)}{b} \rceil$ in the ordering described above, we are within b different matchings from the optimal matching of A and B (figure 15). We can find these translations in $O(\delta(n+m)\log(n+m))$ time if we find and sort all the translations.

Alternatively, we can find these translations in $O(\frac{\delta(n+m)}{b}\delta(n+m))$ time if we run $\lceil \frac{2\delta(n+m)}{b} \rceil$ quantile operations. The same is true for A_y and B_y.

So we get a total of $(\frac{2\delta(m+n)}{b})^2$ pairs of translations in the (x, y) plane. Since there is one that is b away from the optimal in each dimension, there is one that is $2b$ away from the optimal in 2 dimensions. Setting $b = \frac{\beta(n+m)}{2}$ completes the proof. □

Given trajectories A, B with lengths n, m respectively, and constants δ, β, ϵ, the approximation algorithm works as follows:

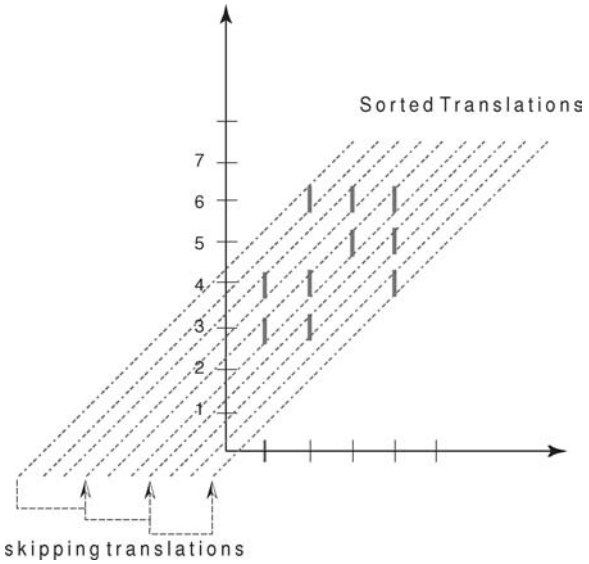

Figure 15. If we can afford to be within a certain error range then we don't have to try *all* translations.

1. Using the projections of A, B on the two axes, find the sets of all different translations on the x and y axis.
2. Find the $i\frac{\beta(n+m)}{2}$-th quantiles for each set, $1 \leq i \leq \frac{4\delta}{\beta}$.
3. Run the $LCSS_{\delta,\epsilon}$ algorithm on A and B, for each of the $(\frac{4\delta}{\beta})^2$ pairs of translations.
4. Return the highest result.

6. Indexing trajectories for similarity retrieval

Even though the approximation algorithm for the distance D2 significantly reduces the computational cost over the exact algorithm, it can still prove costly when one is interested in similarity search on massive trajectory databases.

Here we will show how to utilize the tree produced by a hierarchical clustering algorithm in order to efficiently answer nearest neighbor queries.

The major obstacle in providing an indexing scheme for the distance function $D2$ is that it is not a metric, since it does not obey the triangle inequality. This makes the use of traditional indexing techniques difficult. Indeed, it is easy to construct examples where we have trajectories A, B and C, where $D2(\delta, \epsilon, A, C) > D2(\delta, \epsilon, A, B) + D2(\delta, \epsilon, B, C)$. Such an example is shown in figure 16, where $D2(\delta, \epsilon, A, B) = D2(\delta, \epsilon, B, C) = 0$ (since the similarity is 1), and $D2(\delta, \epsilon, A, C) = 1$ (because the similarity within ϵ in space is zero).

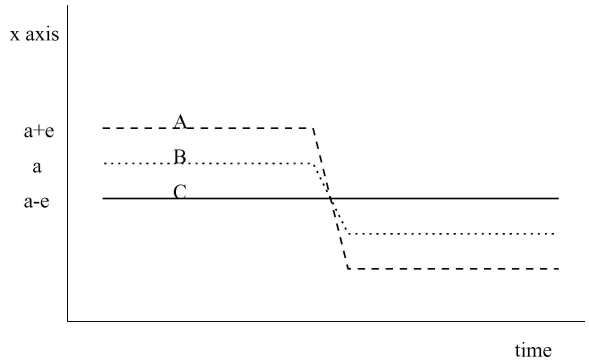

Figure 16. An example where the triangle inequality does not hold for the $D2$ distance.

We can however prove a weaker version of the triangle inequality, which can help us avoid examining a large portion of the database objects. First we define:

$$LCSS_{\delta,\epsilon,\mathcal{F}}(A, B) = \max_{f_{c,d}\in\mathcal{F}} LCSS_{\delta,\epsilon}(A, f_{c,d}(B))$$

Clearly, $D2(\delta, \epsilon, A, B) = 1 - \frac{LCSS_{\delta,\epsilon,\mathcal{F}}(A,B)}{\min(|A|,|B|)}$ (as before, $\mathcal{F}$ is the set of translations). Now we can show the following lemma:

Lemma 6. *Given trajectories A, B, C:*

$$LCSS_{\delta,2\epsilon,\mathcal{F}}(A, C) \geq LCSS_{\delta,\epsilon,\mathcal{F}}(A, B) + LCSS_{\delta,\epsilon,\mathcal{F}}(B, C) - |B|$$

where $|B|$ is the length of sequence B.

Proof: Clearly, if an element of A can match an element of B within ϵ, and the same element of B matches an element of C within ϵ, then the element of A can also match the element of C within 2ϵ. It follows that:

$$\begin{aligned} LCSS_{\delta,2\epsilon,\mathcal{F}}(A, C) &= \sharp \text{ points in B that match A and C within } \epsilon \\ &\geq (\sharp \text{ points in B}) - (\sharp \text{ B points not matching A}) \\ &\quad - (\sharp \text{ B points not matching C}) = |B| \\ &\quad -(|B| - LCSS_{\delta,\epsilon,\mathcal{F}}(A, B)) - (|B| - LCSS_{\delta,\epsilon,\mathcal{F}}(B, C)) \\ &= LCSS_{\delta,\epsilon,\mathcal{F}}(A, B) + LCSS_{\delta,\epsilon,\mathcal{F}}(B, C) - |B|. \end{aligned}$$

□

6.1. *Indexing structure*

We first partition all the trajectories into sets according to their length, so that the longest trajectory in each set is at most a times the shortest (typically we use $a = 2$.) We apply a hierarchical clustering algorithm on each set, and we use the tree that the algorithm produced as follows:

For every node C of the tree we store the medoid (M_C) of the cluster represented by this node. The medoid is the trajectory that has the minimum distance (or maximum LCSS) from every other trajectory in the cluster: $\max_{v_i \in C} \min_{v_j \in C} LCSS_{\delta,\epsilon,\mathcal{F}}(v_i, v_j, e)$. However, keeping only the medoid is not enough since we need to have a way to efficiently prune part of the tree during the search procedure. Namely, given the tree and a query sequence Q, we want to examine whether to follow the subtree that is rooted at C. However, from the previous lemma we know that for any sequence B in C:

$$LCSS_{\delta,\epsilon,\mathcal{F}}(B, Q) < |B| + LCSS_{\delta,2\epsilon,\mathcal{F}}(M_C, Q) - LCSS_{\delta,\epsilon,\mathcal{F}}(M_C, B)$$

or in terms of distance:

$$D2(\delta, \epsilon, B, Q) = 1 - \frac{LCSS_{\delta,\epsilon,\mathcal{F}}(B, Q)}{\min(|B|, |Q|)} > 1 - \frac{|B|}{\min(|B|, |Q|)} - \frac{LCSS_{\delta,2\epsilon,\mathcal{F}}(M_C, Q)}{\min(|B|, |Q|)} + \frac{LCSS_{\delta,\epsilon,\mathcal{F}}(M_C, B)}{\min(|B|, |Q|)}$$

In order to provide an upper bound on the similarity (or a lower bound on the distance) we have to maximize the expression $|B| - LCSS_{\delta,\epsilon,\mathcal{F}}(A, B)$. Therefore, for every node of the tree along with the medoid we have to keep the trajectory r_c that maximizes this expression. If the length of the query is smaller than the shortest length of the trajectories we are currently considering we use that, otherwise we use the minimum and maximum lengths to obtain an approximate result.

6.2. *Searching the index tree for nearest trajectories*

We assume that we search an index tree that contains trajectories with minimum length $\min l$ and maximum length $\max l$. For simplicity we discuss the algorithm for the 1-Nearest Neighbor query, where given a query trajectory Q we try to find the trajectory in the set that is the most similar to Q.

The search procedure takes as input a node N in the tree, the query Q and the distance to the closest trajectory found so far (figure 17). For each of the children C, we check if the child is a trajectory or a cluster. In case that it is a trajectory, we just compare its distance to Q with the current nearest trajectory. If it is a cluster, we check the length of the query and we choose the appropriate value for $\min(|B|, |Q|)$. Then we compute a lower bound L to the distance of the query with any trajectory in the cluster and we compare the result with the distance of the current nearest neighbor *mindist*. We need to examine this cluster only if L is smaller than *mindist*.

```
Procedure Search(N, Q, mindist)
for each children C of N do {
    if C is a leaf { /* singleton cluster */
        if (D2(C, Q) < mindist) {
            mindist = D2(C, Q);
            NNtraj = C;
            return;
        }
    } else { /* non-leaf, cluster with multiple trajectories */
        if (|C| < minl) {
            K =|C|
        } else if (|C| > maxl) {
            K = |r_C|
        } else {
            K = minl
        }
        L = 1 − (|r_C| + LCSS_{δ,2ε,F}(M_C, r_C) − LCSS_{δ,2ε,F}(M_C, Q)) / K
        if (mindist > L)
            Search(C, Q, mindist)
        else
            return;
}
```

Figure 17. Search procedure for nearest neighbor queries.

In our scheme we use an approximate algorithm to compute the $LCSS_{\delta,\epsilon,\mathcal{F}}$. Consequently, the value of $\frac{LCSS_{\delta,\epsilon,\mathcal{F}}(M_C,B)}{\min(|B|,|Q|)}$ that we compute can be up to β times higher than the exact value. Therefore, since we use the approximate algorithm of Section 3.2 for indexing trajectories, we have to subtract $\frac{\beta * \min(|M_C|,|B|)}{\min(|B|,|Q|)}$ from the bound we compute for $D2(\delta, \epsilon, B, Q)$. Note that we don't need to worry about the other terms since they have a negative sign and the approximation algorithm always underestimates the *LCSS*.

7. Experimental evaluation

We implemented the proposed approximation and indexing techniques as they are described in the previous sections and here we present experimental results evaluating our techniques. We describe the datasets and then we continue by presenting the results. The purpose of our experiments is twofold: first, to evaluate the efficiency and accuracy of the approximation algorithm presented in Section 5 and the sigmoidal matching function of Section 3. Second, to evaluate the indexing technique that we discussed in the previous section. Our experiments were run on a PC AMD Athlon at 1 GHz with 1 GB RAM and 60 GB hard disk.

7.1. Time and accuracy experiments

Here we present the results of some experiments using the approximation algorithm to compute the similarity function $S2$. Our dataset consists of marine mammals' satellite

Table 2. Similarity values between two sequences from the *SEALS* dataset.

			Approximate for K tries			
δ	ϵ	Exact	4	9	25	49
2	0.25	0.316	0.1846	0.22	0.253	0.273
2	0.5	0.571	0.410	0.406	0.510	0.521
4	0.25	0.387	0.196	0.258	0.306	0.323
4	0.5	0.612	0.488	0.467	0.563	0.567
6	0.25	0.408	0.250	0.313	0.357	0.367
6	0.5	0.653	0.440	0.4912	0.584	0.591

tracking data.[2] It contains of sequences recording the geographic locations of various marine animals (dolphins, sea lions, whales, etc) tracked over different periods of time, that range from one to three months (*SEALS* dataset). The length of the trajectories is close to 100. Examples have been shown in figure 1.

In Table 2 we show the computed similarity between a pair of sequences in the *SEALS* dataset. We run the exact and the approximate algorithm for different values of δ and ϵ and we report here some indicative results. K is the number of times the approximate algorithm invokes the *LCSS* procedure (that is, the number of translations (c, d) that we try). We can observe that for $K = 25$ and 49 we get very good results. We got similar results for synthetic datasets. Also, in Table 3 we report the running times to compute the similarity measure between two trajectories of the same dataset. The approximation algorithm uses again from 4 to 49 different runs. The running time of the approximation algorithm is much faster even for $K = 49$.

One can notice from the experimental results that the running times of the approximation algorithm is not proportional to the number of runs (K). This is achieved by reusing the results of previous translations and terminating early the execution of the current translation, if it is not going to yield a better result. The main conclusion of the above experiments is

Table 3. Running time (sec) of the exact and approximate LCSS model.

			Approximate for K tries			
δ	ϵ	Exact	4	9	25	49
2	0.25	17.705	0.0012	0.0014	0.0017	0.0022
2	0.5	17.707	0.0012	0.0014	0.00169	0.0022
4	0.25	32.327	0.0016	0.0018	0.0022	0.00281
4	0.5	32.323	0.0015	0.0018	0.0023	0.00280
6	0.25	90.229	0.0017	0.00191	0.00231	0.0031
6	0.5	90.232	0.0017	0.00193	0.0023	0.0031

that the approximation algorithm can provide a very tractable time vs accuracy trade-off for computing the similarity between two trajectories, when the similarity is defined using the *LCSS* model.

7.2. Classification using the approximation algorithm

We compare the clustering performance of our method to the widely used Euclidean and DTW distance functions. Specifically:

1. The Euclidean distance is only defined for sequences of the same length (and the length of our sequences vary considerably). We tried to offer the best possible comparison between every pair of sequences, by sliding the shorter of the two trajectories across the longer one and recording their minimum distance.
2. For DTW we modified the original algorithm in order to match both x and y coordinates. In both DTW and Euclidean we normalized the data before computing the distances (z-normalization: subtract mean, divide by standard deviation). While the z-normalization is optimal under the Euclidean distance (Goldin & Kanellakis, 1995), this does not necessarily hold under warping conditions. However, it still provides more intuitive results than working with un-normalized sequences. Our method does not need any normalization, since it evaluates the similarity under a diverse set of translations, that guarantee the matching quality.
3. For LCSS we used a randomized version with and without sampling, and for various values of δ. The time and the correct clusterings represent the average values of 15 runs of the experiment. This is necessary due to the randomized nature of our approach.

7.2.1. Determining the values for δ & ϵ. The values we used for δ and ϵ are clearly 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 yield significant improvement. Furthermore, after some point the similarity stabilizes to a certain value. Therefore, the δ parameter can be first approximated using the DTW measure (which uses just one parameter) and estimating the warping length, after which classification accuracy does not get improved.

Subsequently the parameter ϵ can be estimated by probing a few values on the discovered δ parameter and recording 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. Nevertheless, when we use the index the value of ϵ has to be the same for all pairs of trajectories.

Even though the setting of the second parameter ϵ may seem to reduce the usability of the $S1$ and $S2$ measures, it is actually a significant asset of the LCSS method. This is because it provides additional spatial filtering abilities. For example, a possible query at an airport watch tower, can be: "Find all planes that took a similar route as plane X, and were always within a radius of ϵ". Such queries can be easily answered only using the LCSS model.

7.2.2. Experiment 1—Video tracking data. The 2D time series obtained represent the X and Y position of a human tracking feature (e.g. tip of finger). In conjunction with a *"spelling program"* the user can *"write"* various words (Gips, Betke, & Fleming, 2000; Betke, Gips, & Fleming, 2002). We used 3 recordings of 5 different words. The data correspond to the following words: '*athens*', '*berlin*', '*london*', '*boston*', '*paris*'. The average length of the series is around 1100 points. The shortest one is 834 points and the longest one 1719 points.

To determine the efficiency of each similarity measure we perform a simple clustering experiment. We take all instances for each pair of words (i.e. $N = 3 * 2$ sequences) and we compute the distance matrix between all pairs of sequences using $N^2/2$ distance operations. This experiment is performed for each all pairs of words (that is, for this dataset we perform the experiment $5 * 4/2 = 10$ times).

A hierarchical clustering is run on the distance matrix of each similarity measure, with the expectation to produce a correct partition into 2 classes (based on our knowledge of which word each trajectory actually represents). While at the lower levels of the dendrogram the clustering is subjective, if the top level provides an accurate division into two classes, then the clustering is considered correct, otherwise not. Using IR terminology, in this experiment we essentially estimate the 'precision' of each method.

We have performed the clustering experiment using single, complete and average linkage. However, since the best results for every distance function are produced using the *complete linkage*, we report only the results for this approach (Table 4). The same experiment is conducted with the rest of the datasets. Results have been also collected for various sequence sample sizes and values of δ (as a percentage of the original series length).

The results with the Euclidean distance have many classification errors and the DTW has some errors, too. For the LCSS the only real variations in the clustering are for sample sizes $s \leq 10\%$. Still the average incorrect clusterings for these cases were constantly less than one (<0.7). For 15% sampling or more, there were no errors.

Table 4. Results using the video tracking data for various sizes of sample s and δ.

Distance function	Time (sec)	Correct clusterings (out of 10) complete linkage
Euclidean	34.96	2
DTW	237.641	8
LCSS:		
$s = 5\%, \delta = 20\%$	2.733	9.800
$s = 10\%, \delta = 20\%$	8.041	9.933
$s = 15\%, \delta = 20\%$	16.173	10
$s = 20\%, \delta = 20\%$	28.851	10
$s = 25\%, \delta = 20\%$	45.065	10
$s = 30\%, \delta = 20\%$	65.203	10
$s = 40\%, \delta = 20\%$	113.583	10
$s = 60\%, \delta = 20\%$	266.753	10
$s = 100\%, \delta = 20\%$	728.277	10

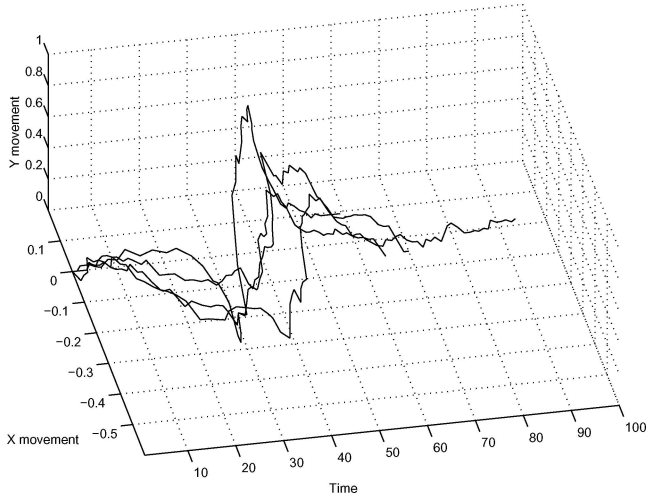

Figure 18. Four recordings of the word '*norway*' in the Australian Sign Language. The graph depicts the x & y position of the writer's hand.

7.2.3. Experiment 2—Australian sign language dataset (ASL). The dataset[3] consists of various parameters (such as the X ,Y, Z hand position, azimuth etc) tracked while different writers sign one the 95 words of the ASL. These series are relatively short (50-100 points). We used only the X and Y parameters and collected 5 recordings of the following 10 words: '*Norway*', '*cold*', '*crazy*', '*eat*', '*forget*', '*happy*', '*innocent*', '*later*', '*lose*', '*spend*'. This is the experiment conducted also in Keogh and Pazzani (2000) (but there only one dimension was used). Examples of this dataset can be seen in figure 18.

The performance of the LCSS in this experiment is similar to the DTW (*DTW* recognized correctly 20 clusters and *LCSS* recognized 21 clusters). This is expected since this dataset does not contain excessive noise and furthermore the data seem to be already normalized and rescaled within the range $[-1 \ldots 1]$. Therefore in this experiment we used also the similarity function $S1$ (no translation), since the use of translations was not going to provide any further improvement (see figure 20). Sampling is only performed down to 75% of the series length (these trajectories are already short). As a consequence, even though we don't gain much in accuracy, our execution time is comparable to the Euclidean (without performing any translations). This is easily explained, since the computation of the L_2 Norm is more computationally intensive, than the simple range comparison that is used in our approach.

7.2.4. Experiment 3—ASL with added noise. We added noise at every sequence of the ASL at a random starting point and for duration equal to the 15% of the series length (see figure 19). The noise was added using the function: $\langle \vec{x}_{\text{noise}}, \vec{y}_{\text{noise}} \rangle = \langle \vec{x}, \vec{y} \rangle + randn * rangeValues$, where *randn* produces a random number, chosen from a normal distribution with mean zero and variance one, and *rangeValues* is the range of values on X or Y coordinates. In this last experiment we wanted to see how the addition of noise would

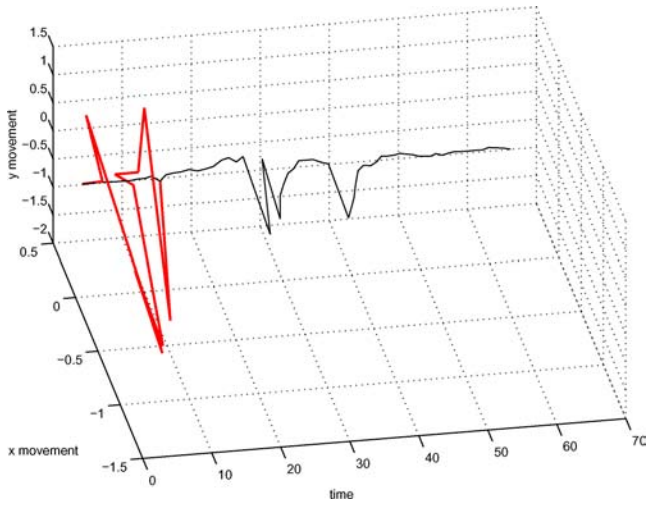

Figure 19. Instance of the ASL dataset after the addition of noise.

affect the performance of the various distance functions. Again, the running time is the same as with the original ASL data.

The LCSS proves to be more robust than the Euclidean and the DTW under noisy conditions (Table 5, figures 20 and 21). The Euclidean again performed poorly, recognizing only 5 clusters, the DTW recognized 7 and the LCSS (S1 and S2) up to 14 clusters (almost as many recognized by the Euclidean *without* any noise!). The sigmoidal based similarity function (S3), proved to be the most robust to noise recognizing 17 clusters.

7.2.5. Experiment 4—Comparison of sigmoidal matching to various Lp-Norm based functions. In the previous experiment we verified the high resilience to noise of the sigmoidal matching function. It has also been noted that the use of Norms lower than L_1 typically provide better performance for noisy datasets (because extreme values don't dominate the

Table 5. Results for ASL data and ASL with added noise for the Euclidean, DTW and the LCSS based distance functions. The sigmoidal matching function S3 provides the best clustering results.

Distance	Time (sec)	Correct clusterings (out of 45) ASL	Correct clusterings (out of 45) ASL with noise
Euclidean	2.271	15	5
DTW	9.112	20	7
S1	2.703	21	14
S2	51.320	21	14
S3	3.034	23	17

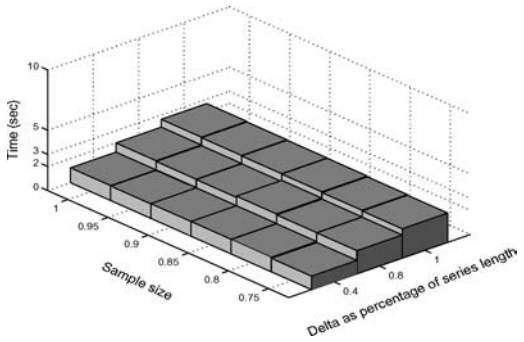

Figure 20. *ASL data*: Time required to compute the pairwise distances of the 45 combinations (same for ASL and ASL with noise).

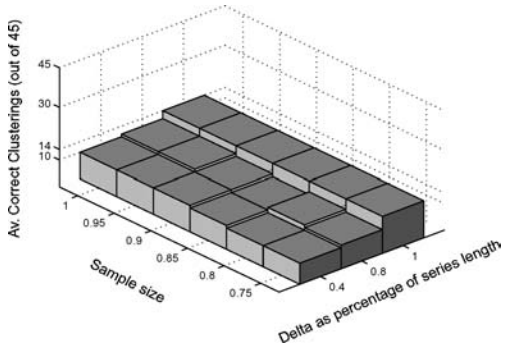

Figure 21. *Noisy ASL data*: The correct clusterings of the LCSS method using complete linkage.

total distance (Aggarwal, Hinneburg, & Keim, 2001)). Here we provide a comprehensive comparison of the sigmoidal function to the L_p Norms, as well as to the Dynamic Time Warping distance that utilizes different L_p-Norms as the base distance function.

For this experiment we use a smaller instance of the ASL dataset comprised of 7 classes of objects, with 5 instances per class (*ASLsmall*). Now the possible class pairings for our experiment are $7 * 6/2 = 21$. This dataset is also corrupted with noise, in a similar fashion as in the previous experiment and the new dataset is called *ASLsmallNoise*. The distorted part of the sequence can be considered non-recoverable.

Figures 22 and 23 depict the number of correct clusters (out of 21) discovered by the various distance functions. We observe that the use of the lower Norms improves the clustering accuracy, however the sigmoidal function (denoted as *SigmoidSim*) performs consistently better. The dataset with the additional noise shows the true potential of our method. Under the strong presence of noise, the sigmoidal function is almost twice as accurate than the other distance functions (figure 23), with comparable accuracy performance as the $L_{0.1}$-Norm on the same dataset without any noise.

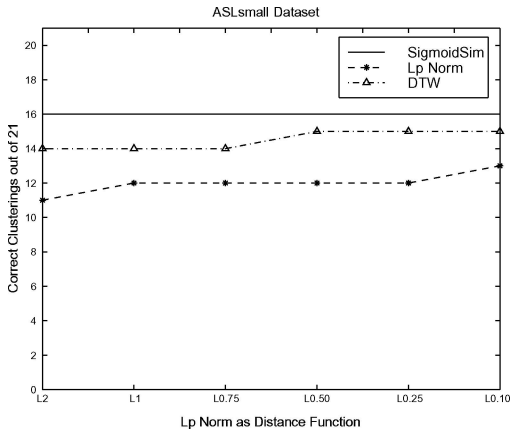

Figure 22. Clustering accuracy comparison between the sigmoidal matching function and various L_p-norms. We observe that the use of smaller norms can improve the clustering results.

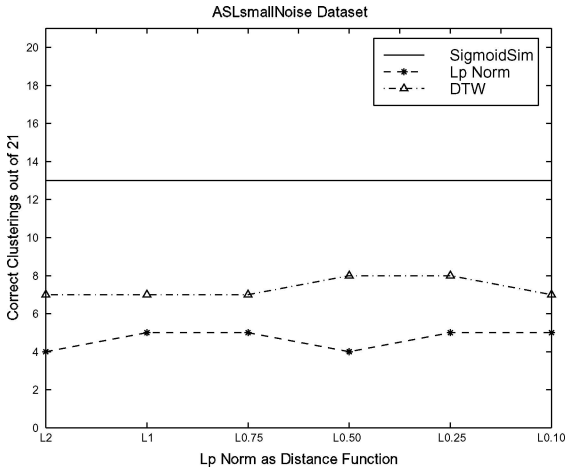

Figure 23. Under the presence of noise, the *SigmoidSim* is noticably superior to the other alternatives (depicting almost twice as high accuracy).

7.3. Evaluating the quality and efficiency of the indexing technique

In this part of our experiments we evaluated the efficiency and effectiveness of the proposed indexing scheme. We performed tests over datasets of different sizes and different number of clusters. To generate large realistic datasets, we used real trajectories (from the $SEALS$ and ASL datasets) as "seeds" to create larger datasets that follow the same patterns. To perform tests, we used queries that do not have exact matches in the database, but on the other hand are similar to some of the existing trajectories. For each experiment we run 100 different queries and we report the averaged results.

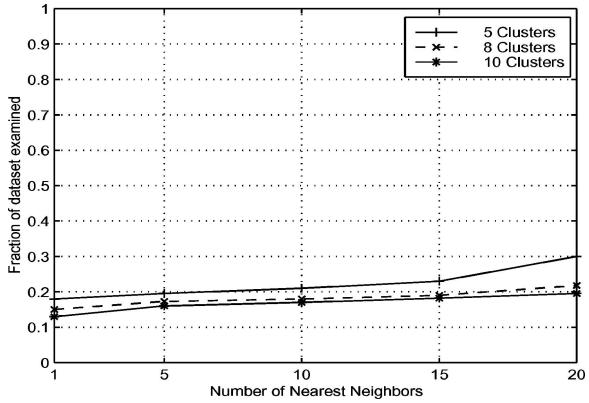

Figure 24. Performance for increasing number of Nearest Neighbors.

We have tested the index performance for different number of clusters in a dataset consisting of a total of 2000 trajectories. We executed a set of K-Nearest Neighbor (K-NN) queries for K 1, 5, 10, 15 and 20 and we plot the fraction of the dataset that has to be examined in order to guarantee that we have found the best match for the K-NN query. Note that in this fraction we included the medoids that we check during the search since they are also part of the dataset.

In figure 24 we show some results for K-Nearest Neighbor queries. We used datasets with 5, 8 and 10 clusters. As we can see the results indicate that the algorithm has good performance even for queries with large K. We also performed similar experiments where we varied the number of clusters in the datasets. As the number of clusters increased the performance of the algorithm improved considerably. This behavior is expected and it is similar to the behavior of recent proposed index structures for high dimensional data (Chakrabarti & Mehrotra, 2000; Beyer et al., 1999; Goldstein & Ramakrishnan, 2000). On the other hand if the dataset has no clusters, the performance of the algorithm degrades, since the majority of the trajectories have almost the same distance to the query. This behavior follows again the same pattern of high dimensional indexing methods (Beyer et al., 1999; Weber, Schek, & Blott, 1998).

The last experiment evaluates the index performance, over sets of trajectories with increasing cardinality. We indexed from 1000 to 16000 trajectories. The pruning power of the inequality is evident in figure 25. As the size of the database increases, we can avoid examining a larger fraction of the database.

8. Related work

Most of the related work on time-series has concentrated on the use of some metric L_p Norm. The Norm distance between two n-dimensional vectors $\bar{x}$ and $\bar{y}$ is defined as $L_p(\bar{x}, \bar{y}) = (\sum_{i=1}^{n}(|x_i - y_i|)^p)^{\frac{1}{p}}$. For $p = 2$ it is the well known Euclidean distance and for $p = 1$

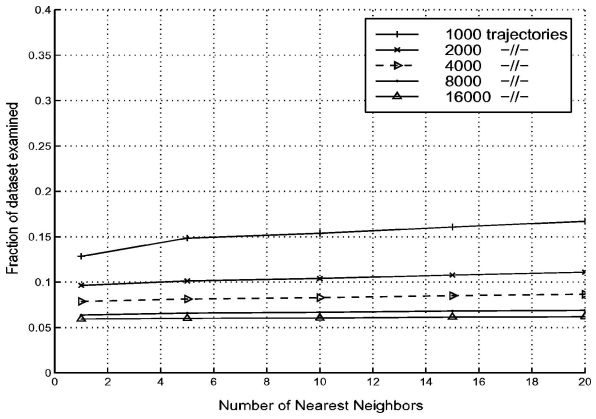

Figure 25. The pruning power increases along with the database size.

the Manhattan distance. The advantage of this simple metric is that it allows efficient indexing by a dimensionality reduction technique (Agrawal, Faloutsos, & Swami, 1993; Yi & Faloutsos, 2000; Faloutsos, Ranganathan, & Manolopoulos, 1994). On the other hand the model cannot deal well with outliers and is very sensitive to small distortions in the time axis (Vlachos, Kollios, & Gunopulos, 2002). There are a number of interesting extensions to the above model to support various transformations such as scaling (Chu & Wong, 1999; Rafiei & Mendelzon, 2000), shifting (Chu & Wong, 1999; Goldin & Kanellakis, 1995), normalization (Goldin & Kanellakis, 1995) and moving average (Rafiei & Mendelzon, 2000). Other recent works on indexing time series data for similarity queries assuming the Euclidean model include (Keogh et al., 2001; Kahveci & Singh, 2001). A domain independent framework for defining queries in terms of similarity of objects is presented in Jagadish, Mendelzon, and Milo (1995).

Other techniques to define time series similarity are based on extracting certain features (Landmarks (Perng et al., 2000) or signatures (Faloutsos et al., 1997)) from each time-series and then using these features to define the similarity. Another approach to represent a time series using the direction of the sequence at regular time intervals is presented in Qu, Wang, and Wang (1998). Ge and Smyth (2000) present an alternative sequence similarity model which is based on probabilistic matching, using Hidden Markov Models. However, their approach cannot be scaled to large databases. A recent work that proposes a method to cluster trajectory data is due to Gaffney and Smyth (1999). They use a variation of the EM (expectation maximization) algorithm to cluster small sets of trajectories. However, their method is a model based approach that usually has scalability problems.

Most of the above work has concentrated on 1D time-series. Work on multidimensional sequences can be found in Lee et al. (2000) and Kahveci, Singh, and Gurel (2002), but only support the use of Euclidean distance, which 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, utilize 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, warping distances have been used for gene expression data mining (Aach & Church, 2001; Bar-Joseph et al., 2002), for tracking time series extracted from video (Gavrila & Davis, 1995), classifying handwritten text (Rath & Manmatha, 2002; Munich & Perona, 1999) and even fingerprint indexing (Kovács-Vajna, 2000). Finally, perhaps the most familiar example of easy classifiable patterns, which nevertheless cause problems for Lp metrics, are electrocardiograms. The cardiological community has used DTW for several decades (Strik & Boves, 1988).

Approaches to mitigate the large computational cost of the DTW have appeared in Yi, Jagadish, and Faloutsos (1998); Kim, Park, and Chu (2001); Keogh (2002), Zhu and Shasha (2003) where lower bounding functions are used in order to speed up the execution of DTW. Keogh (2002) and Zhu and Shasha (2003) utilize envelope based approaches for approximating the DTW distance, while (Vlachos et al., 2003) generalizes these approaches for multiple distance measures (without however providing translation invariant matching).

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 (Levenshtein, 1966). 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 (Agrawal et al., 1995; Bozkaya, Yazdani, & Ozsoyoglu, 1997; Bollobás et al., 1997; Das, Gunopulos, & Mannila, 1997).

The majority of the above work on warping distances deals with 1-dimensional time-series. (Vlachos et al., 2003) has presented ways to speed up the execution of the multidimensional LCSS case. However, while most previous work typically utilizes some heuristic normalizations (e.g. subtract mean and divide by standard deviation the time-series) to align distant sequences and identify similar patterns, this is the only work that can provide *guarantees* on the quality of the match for trajectories, even under translation transformations. In that sense, this work is more appropriate for applications where more weight is given on the accurate registration of time-series (eg for bio-medical or engineering applications).

9. Conclusion

In this paper we presented efficient techniques to accurately compute the similarity between trajectories of moving objects. Our distance measure is based on the LCSS model and performs very well for noisy signals. Since the exact computation is inefficient, we presented approximate algorithms with provable performance bounds. Moreover, we presented an efficient index structure, which is based on hierarchical clustering, for similarity (nearest neighbor) queries. The distance that we use is not a metric and therefore the triangle inequality does not hold. However, we prove that a similar inequality holds (although a weaker one) that allows to prune parts of the datasets without any false dismissals.

Our experimentals indicate that the approximation algorithm can be used to get an accurate and fast estimation of the distance between two trajectories even under noisy conditions. Also, results from the index evaluation show that we can achieve good speed-ups for searching similar trajectories comparing to the brute force linear scan.

We plan to investigate biased sampling to improve the running time of the approximation algorithms, especially when full rigid transformations (eg. shifting, scaling and rotation) are necessary. Another approach to index trajectories for similarity retrieval is to use embeddings and map the set of trajectories to points in a low dimensional Euclidean space (Faloutsos & Lin, 1995). The challenge of course is to find an embedding that approximately preserves the original pairwise distances and gives good approximate results to similarity queries.

Acknowledgments

Michail Vlachos and Dimitrios Gunopulos were partially supported by NSF grants ITR-0220148 and IIS-0330481. George Kollios was partially supported by NSF grants IIS-0133825 and IIS-0308213.

The authors would like to thank the editor of the special issue, Eamonn Keogh, and the anonymous reviewers for their valuable comments and suggestions, which significantly improved the appearance and clarity of the article.

Notes

1. More descriptions about the datasets can be found in the experimental section.
2. `http://whale.wheelock.edu/whalenet-stuff/stop_11_cover.html`
3. `http://kdd.ics.uci.edu/databases/auslan2/auslan.data.html`

References

Aach, J., & Church, G. (2001). Aligning gene expression time series with time warping algorithms. *Bioinformatics, 17*, 495–508.

Aggarwal, C. C., Hinneburg, A., & Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional spaces. In *ICDT* (pp. 420–434).

Agrawal, R., Faloutsos, C., & Swami, A. (1993). Efficient similarity search in sequence databases. In *Proc. of the 4th International Conference of Foundations of Data Organization (FODO)* (pp. 69–84).

Agrawal, R., Lin, K., Sawhney, H. S., & Shim, K. (1995). Fast similarity search in the presence of noise, scaling and translation in time-series databases. In *Proc of International Conference on Very Large Databases (VLDB)* (pp. 490–501).

Bar-Joseph, Z., Gerber, G., Gifford, D., Jaakkola, T., & Simon, I. (2002). A new approach to analyzing gene expression time series data. In *Proc. of 6th RECOMB* (pp. 39–48).

Barbará, D. (1999). Mobile computing and databases—A survey. *IEEE TKDE* (pp. 108–117).

Berndt, D., & Clifford, J. (1994). Using dynamic time warping to find patterns in time series. In *AAAI Workshop on Knowledge Discovery in Databases* (pp. 229–248).

Betke, M., Gips, J., & Fleming, P. (2002). The camera mouse: Visual tracking of body features to provide computer access for people with severe disabilities. *IEEE Transactions on Neural Systems and Rehabilitation Engineering 10:1*, 1–10.

Beyer, K., Goldstein, J., Ramakrishnan, R., & Shaft, U. (1999). When is nearest neighbor meaningful? In *Proc of ICDT* (pp. 217–235).

Bollobás, B., Das, G., Gunopulos, D., & Mannila, H. (1997). Time-Series similarity problems and well-separated geometric sets. In *Proc of the 13th SCG, Nice*, France.

Bozkaya, T., Yazdani, N., & Ozsoyoglu, M. (1997). Matching and indexing sequences of different lengths. In *Proc.of the CIKM,* Las Vegas.

Chakrabarti, K., & Mehrotra, S. (2000). Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proc. of VLDB* (pp. 89–100).

Chu, K., & Wong, M. (1999). Fast time-series searching with scaling and shifting. *ACM Principles of Database Systems* (pp. 237–248).

Das, G., Gunopulos, D., & Mannila, H. (1997). Finding similar time series. In *Proc. of the First PKDD Symp.* (pp. 88–100).

Duda, R., & Hart, P. (1973). *Pattern Classification and Scene Analysis*. John Wiley and Sons, Inc.

Faloutsos, C., Jagadish, H., Mendelzon, A., & Milo, T. (1997). Signature technique for similarity-based queries. In *Proc. of the Compression and Complexity of Sequences*.

Faloutsos, C., & Lin, K.-I. (1995). FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proc. ACM SIGMOD* (pp. 163–174).

Faloutsos, C., Ranganathan, M., & Manolopoulos, I. (1994). Fast subsequence matching in time series databases. In *Proc. of ACM SIGMOD* (pp. 419–429).

Gaffney, S., & Smyth, P. (1999). Trajectory clustering with mixtures of regression models. In *Proc. of the 5th ACM SIGKDD*, San Diego, CA (pp. 63–72).

Gavrila, D., & Davis, L. (1995). Towards 3-d model-based tracking and recognition of human movement: A multi-view approach. In *Int. Workshop on Face and Gesture Recognition* (pp. 272–277).

Ge, X., & Smyth, P. (2000). Deformable Markov model templates for time-series pattern matching. In *Proc. of ACM SIGKDD* (pp. 81–90).

Gips, J., Betke, M., & Fleming, P. (2000). The camera mouse: Preliminary investigation of automated visual tracking for computer access. In *Proceedings of the Rehabilitation Engineering and Assistive Technology Society of North America 2000 Annual Conference* (pp. 98–100) Orlando, FL.

Goldin, D., & Kanellakis, P. (1995). On similarity queries for time-series data. In *Proc. Int. Conference on Principles and Practice of Constraint Programming* (pp. 137–153).

Goldstein, J., & Ramakrishnan, R. (2000). Contrast plots and P-Sphere trees: Space vs. time in nearest neighbor searches. In *Proc. of VLDB* (pp. 429–440).

Guha, S., Rastogi, R., & Shim, K. (1998). CURE: An efficient clustering algorithm for large databases. In *Proc. of ACM SIGMOD* (pp. 73–84).

Jagadish, H. V., Mendelzon, A. O., & Milo, T. (1995). Similarity-Based aueries. In *Proc. of the 14th ACM PODS* (pp. 36–45).

Kahveci, T., Singh, A., & Gurel, A. (2002). Similarity searching for Multi-attribute sequences. In *Proc. of SSDBM*.

Kahveci, T., & Singh, A. K. (2001). Variable length queries for time series data. In *Proc. of IEEE ICDE* (pp. 273–282).

Keogh, E. (2002). Exact indexing of dynamic time warping. In *Proc. of VLDB*.

Keogh, E., Chakrabarti, K., Mehrotra, S., & Pazzani, M. (2001). Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. of ACM SIGMOD* (pp. 151–162).

Keogh, E., & Pazzani, M. (2000). Scaling up dynamic time warping for datamining applications. In *Proc. 6th SIGKDD*.

Keogh, E., & Ratanamahatana, A. (2004). Everything you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data (SIGKDD)*.

Kim, S., Park, S., & Chu, W. (2001). An Index-based approach for similarity search supporting time warping in large sequence databases. In *Proc. of 17th ICDE*.

Kivinen, J., & Mannila, H. (1994). The power of sampling in knowledge discovery. *Proceedings of PODS*.

Kovács-Vajna, Z. (2000). A fingerprint verification system based on triangular matching and dynamic time warping. In *IEEE Transactions on PAMI, 22:11*, 1266–1276.

Lee, S.-L., Chun, S.-J., Kim, D.-H., Lee, J.-H., & Chung, C.-W. (2000). Similarity search for multidimensional data sequences. In *Proceedings of ICDE* (pp. 599–608).

Levenshtein, V. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics–Doklady, 10:10*, 707–710.

Medin, D., Goldstone, R., & Gentner, D. (1993). Respects for similarity. In *Psychological Review, 100:2*, 254–278.

Munich, M., & Perona, P. (1999). Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *7th International Conference on Computer Vision* (pp. 108–115).

Perng, S., Wang, H., Zhang, S., & Parker, D. S. (2000). Landmarks: A New model for similarity-based pattern querying in time series databases. In *Proc. of International Conference on Data Engineering (ICDE)* (pp. 33–42).

Qu, Y., Wang, C., & Wang, X. (1998). Supporting fast search in time series for movement patterns in multiple scales. In *Proc of the ACM CIKM* (pp. 251–258).

Rafiei, D., & Mendelzon, A. (2000). Querying time series data based on similarity. *IEEE Transactions on Knowledge and Data Engineering, 12:5*, 675–693.

Ratanamahatana, C. A., & Keogh, E. (2004). Making time-series classification more accurate using learned constraints. In *Proc. of SIAM International Conference on Data Mining (SDM)*.

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

Strik, H., & Boves, L. (1988). Averaging physiological signals with the use of a DTW algorithm. In *SPEECH'88, 7th FASE Symposium, Book 3* (pp. 883–890).

Vlachos, M., Hadjieleftheriou, M., Gunopulos, D., & Keogh, E. (2003). Indexing multi-dimensional time-series with support for multiple distance measures. In *Proc. of 9th SIGKDD* (pp. 216–225).

Vlachos, M., Kollios, G., & Gunopulos, D. (2002). Discovering similar multidimensional trajectories. In *Proc. of ICDE* (pp. 673–684).

Weber, R., Schek, H.-J., & Blott, S. (1998). A quantitative analysis and performance study for similarity search methods in high-dimensional spaces. In *Proc. of VLDB* (pp. 194–205).

Yi, B.-K., & Faloutsos, C. (2000). Fast time sequence indexing for arbitrary Lp norms. *In Proc. of VLDB*.

Yi, B.-K., Jagadish, H. V., & Faloutsos, C. (1998). Efficient retrieval of similar time sequences under time warping. In *Proc. of ICDE* (pp. 201–208).

Zaki, M. J., Parthasarathy, S., Li, W., & Ogihara, M. (1997). Evaluation of sampling for data mining of association rules. *Proceedings of RIDE*.

Zhu, Y., & Shasha, D. (2003). Warping indexes with envelope transforms for query by humming. In *Proc. of ACM SIGMOD*. pp. 181–192.

Received March 31, 2004
Revised October 12, 2004
Accepted November 15, 2004
Final manuscript November 15, 2004