Discovering Similar Multidimensional Trajectories

Michail Vlachos
UC Riverside
mvlachos@cs.ucr.edu

George Kollios
Boston University
gkollios@cs.bu.edu

Dimitrios Gunopulos
UC Riverside
dg@cs.ucr.edu

Abstract

We investigate techniques for analysis and retrieval of object trajectories in a two or three dimensional space. 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 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.

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, 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 spatiotemporal data and there is increasing interest to perform data analysis tasks over this data [4]. 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 spatiotemporal 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 analysis tasks.

1.1 Robust distance metrics for trajectories

In general these trajectories will be 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* or in other words incorrect data measurements (unlike for example, stock data which contain no errors whatsoever).

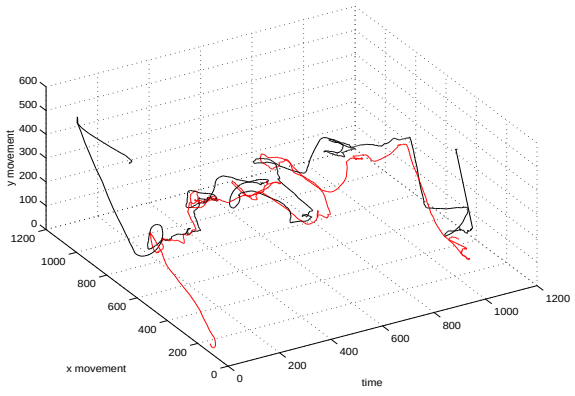

Figure 1. Examples of 2D trajectories. Two instances of video-tracked time-series data representing the word 'athens'. Start & ending contain many outliers.

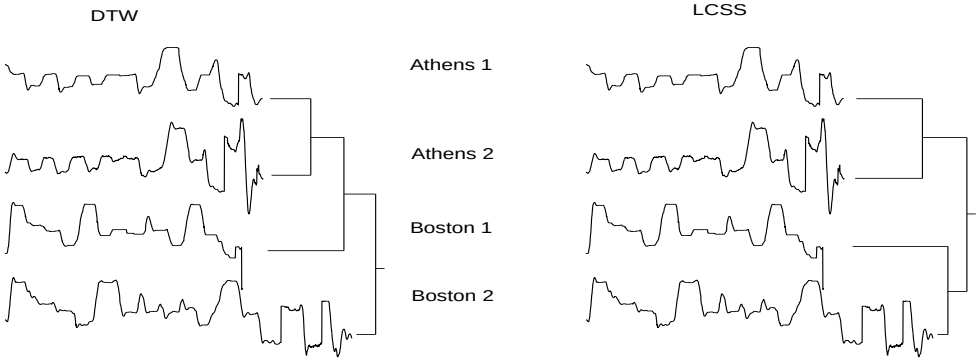

Figure 2. Hierarchical clustering of 2D series (displayed as 1D for clariry). *Left*: The presence of many outliers in the beginning and the end of the sequences leads to incorrect clustering. DTW is not robust under noisy conditions. *Right*: The $LCSS$ focusing on the common parts achieves the correct clustering.

Our objective is the automatic classification of trajectories using Nearest Neighbor Classification. It has been shown that the one nearest neighbor rule has asymptotic error rate that is at most twice the Bayes error rate[12]. So, the problem is: 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 $\mathcal{Q}$. We need to define the following:

1. A realistic distance function,
2. An efficient indexing scheme.

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 dissimilar parts of the objects. However, they are useful, because they represent an accurate model of the human perception, 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 difference 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 some costly L_p Norm).

In figure 2 we can see the clustering produced by the 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 DTW 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. 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.

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. So, 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 section 2 we formalize the new similarity functions by extending the $LCSS$ model. Section 3 demonstrates efficient algorithms to compute these functions and section 4 elaborates on the indexing structure. Section 5 provides the experimental validation of the accuracy and efficiency of the proposed approach and section 6 presents the related work. Finally, section 7 concludes the paper.

2 Similarity Measures

In this section we define similarity models that match the user perception of similar trajectories. First we give some useful 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.

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:*

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

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 3).

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:*

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

We use this 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.

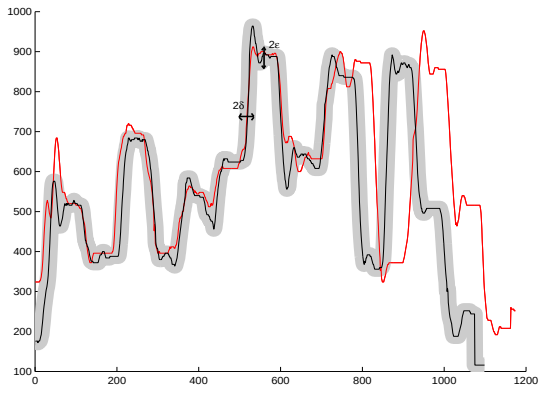

Figure 3. The notion of the $LCSS$ matching within a region of δ & ϵ for a trajectory. The points of the 2 trajectories within the gray region can be matched by the extended $LCSS$ function.

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:*

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

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 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 4 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.

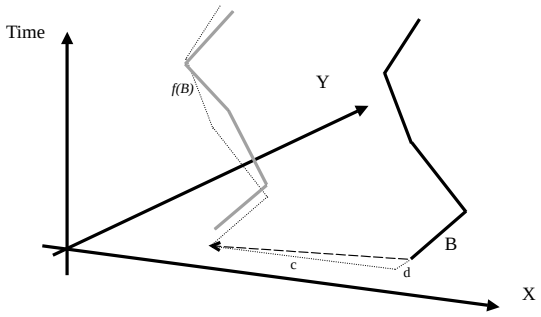

Figure 4. Translation of trajectory B.

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 Efficient Algorithms to Compute the Similarity

3.1 Computing the similarity function $S1$

To compute the similarity functions $S1$ we have to run a $LCSS$ 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 [5], [11].

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. However, for some applications δ may need to be large. For that case, we can speed-up the above computation using random sampling. Given two trajectories A and B, we compute two subsets RA and RB by sampling each trajectory. Then we use the dynamic programming algorithm to compute the $LCSS$ on RA and RB. We can show that, with high probability, the result of the algorithm over the samples, is a good approximation of the actual value. We describe this technique in detail in [35].

3.2 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. Let us 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 to 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: $f_c(x_1, \ldots, x_m) = (x_1 + c, \ldots, x_m + c)$.

Take the x projections of A and B, A_x and B_x respectively. We can show the following lemma:

Lemma 2
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, $LCSS_{\delta,\epsilon}(A_y, f_{d_1}(B_y)) \geq a$.

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 5).

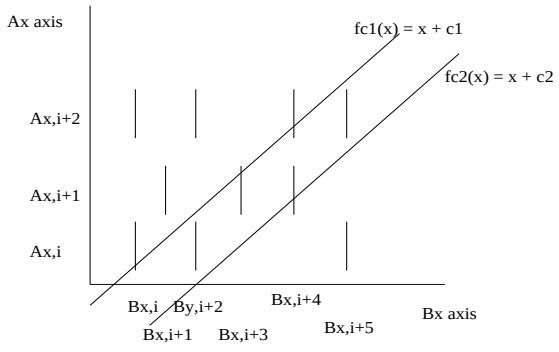

Figure 5. An example of two translations .

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 $LCSS$ 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 $O(\delta n)$.

A translation $f_{c'}$ in one dimension is a function of the form $f_{c'}(b_{x,i}) = b_{x,i} + c'$. Therefore, in the plane we described above, $f_{c'}(b_{x,i})$ is 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 $f_{c1}(x) = x + c1$ and $f_{c2}(x) = x + c2$ in figure 5 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 3 *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.*

3.3 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 in our technique is that we can bound the difference between the sets of line segments that different lines of slope 1 intersect, based on how far apart the lines are.

Consider again the one dimensional projections A_x, B_x. Lets us consider the $O(\delta(n+m))$ translations that result to 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 neighbor translations in this order intersect similar sets of line segments.

Lemma 4 *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 $c'_1 \leq \ldots \leq c'_N$. Then the symmetric difference $L_{f_i} \Delta L_{f_j} = |i - j|$.*

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 into the x and y axes. There exists a translation f_i on x only such that L_{f_i} is a superset of the matches in the optimal $LCSS$ of A and B. In addition, by 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. 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:

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.*

4 Indexing Trajectories for Similarity Retrieval

In this section we show how to use the hierarchical tree of a clustering algorithm in order to efficiently answer nearest neighbor queries in a dataset of trajectories.

The distance function $D2$ is not a metric because it does not obey the triangle inequality. Indeed, it is easy to construct examples where we have trajectories A, B and C, where $D2(\delta, \epsilon, A, C) > D2(\delta, \epsilon, A, B) + D2(\delta, \epsilon, B, C)$. This makes the use of traditional indexing techniques difficult.

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 5 *Given trajectories A, B, C,*

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

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. Since there are at least $|B| - (|B| - LCSS_{\delta,\epsilon,\mathcal{F}}(A, B)) - (|B| - LCSS_{\delta,\epsilon,\mathcal{F}}(B, C))$ elements of B that match with elements of A and with elements of C, it follows that $LCSS_{\delta,2\epsilon,\mathcal{F}}(A, C) > |B| - (|B| - LCSS_{\delta,\epsilon,\mathcal{F}}(A, B)) - (|B| - LCSS_{\delta,\epsilon,\mathcal{F}}(B, C)) = LCSS_{\delta,\epsilon,\mathcal{F}}(A, B) + LCSS_{\delta,\epsilon,\mathcal{F}}(B, C) - |B|$ □

4.1 Indexing Structure

We first partition all the trajectories into sets according to 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 each cluster. The medoid is the trajectory that has the minimum distance (or maximum LCSS) from every other trajectory in the cluster: $max_{v_i \in C} min_{v_j \in C} LCSS_{\delta,\epsilon,\mathcal{F}}(v_i, v_j, e)$. So 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 a lower bound 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.

4.2 Searching the Index tree for Nearest Trajectories

We assume that we search an index tree that contains trajectories with minimum length $minl$ and maximum length $maxl$. 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. 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 $mindist$.

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 $LCSS$.

5 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 3 and 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.

5.1 Time and Accuracy Experiments

Here we present the results of some experiments using the approximation algorithm to compute the similarity function $S2$. Our dataset here comes from marine mammals' satellite tracking data.[1] It consists of sequences of 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 1 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). As we can see, for $K = 25$ and 49 we get very good results. We got similar results for synthetic datasets. Also, in table 1 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 $K = 49$.

As can be observed from the experimental results, 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 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.

5.2 Classification using the Approximation Algorithm

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

[1] http://whale.wheelock.edu/whalenet-stuff/stop_cover.html

		Similarity					Running Time (sec)				
δ	ϵ	Exact	Approximate for K tries				Exact	Approximate for K tries			
			4	9	25	49		4	9	25	49
2	0.25	0.316	0.1846	0.22	0.253	0.273	17.705	0.0012	0.0014	0.0017	0.0022
2	0.5	0.571	0.410	0.406	0.510	0.521	17.707	0.0012	0.0014	0.00169	0.0022
4	0.25	0.387	0.196	0.258	0.306	0.323	32.327	0.0016	0.0018	0.0022	0.00281
4	0.5	0.612	0.488	0.467	0.563	0.567	32.323	0.0015	0.0018	0.0023	0.00280
6	0.25	0.408	0.250	0.313	0.357	0.367	90.229	0.0017	0.00191	0.00231	0.0031
6	0.5	0.653	0.440	0.4912	0.584	0.591	90.232	0.0017	0.00193	0.0023	0.0031

Table 1. Similarity values and running times between two sequences from our *SEALS* dataset.

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. Our method does not need any normalization, since it computes the necessary translations.

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.

5.2.1 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 $20 - 30\%$ of the trajectories length did not yield significant improvement. Furthermore, after some point the similarity stabilizes to a certain value. The determination of ϵ is application dependent. In our experiments we used a value equal to the smallest standard deviation between the two trajectories that were examined at any time, 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.

5.2.2 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 conjuction with a *"spelling program"* the user can *"write"* various words [19]. 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 method we performed hierarchical clustering after computing the $N^2/2$ pairwise distances for all three distance functions. We evaluate the total time required by each method, as well as the quality of the clustering, based on our knowledge of which word each trajectory actually represents. We take all possible pairs of words (in this case $5 * 4/2 = 10$ pairs) and use the clustering algorithm to partition them into two classes. While at the lower levels of the dendrogram the clustering is subjective, the top level should provide an accurate division into two classes. We clustered using single, complete and average linkage. Since the best results for every distance function are produced using the *complete linkage*, we report only the results for this approach (table 2). The same experiment is conducted with the rest of the datasets. Experiments have been conducted for different 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.

5.2.3 Experiment 2 - Australian Sign Language Dataset (ASL) [2].

The dataset 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 [25] (but there only one dimension was used). Examples of this dataset can be seen in figure 6.

[2] `http://kdd.ics.uci.edu`

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

Table 2. Results using the video tracking data for various sizes of sample s and δ.

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

Table 3. Results for ASL data and ASL with added noise for the Euclidean and DTW distance functions.

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 translations were not going to achieve any further improvement (see figure 7). 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.

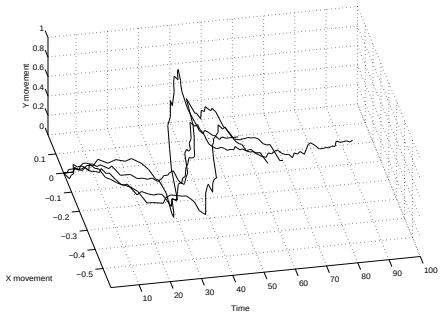

Figure 6. Four recordings of the word *'norway'* in the Australian Sign Language. The graph depicts the x & y position of the writer's hand.

5.2.4 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. The noise was added using the function: $\langle \vec{x}_{noise}, \vec{y}_{noise} \rangle = \langle \vec{x}, \vec{y} \rangle + randn * rangeValues$, 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 affect the performance of the three 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 3, figure 7, 8). The Euclidean again performed poorly, recognizing only 5 clusters, the DTW recognized 7 and the LCSS up to 14 clusters (almost as many recognized by the Euclidean *without* any noise!).

5.3 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.

We have tested the index performance for different number of clusters in a dataset consisting of a total of 2000 tra-

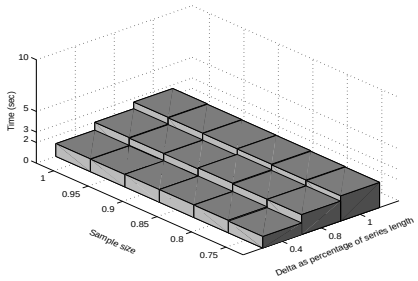

Figure 7. *ASL data*: Time required to compute the pairwise distances of the 45 combinations(same for ASL and ASL with noise)

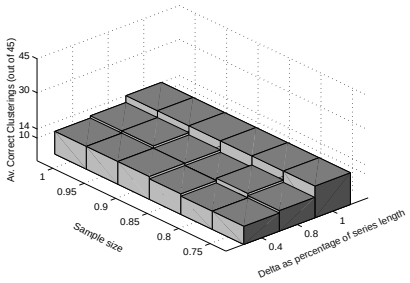

Figure 8. *Noisy ASL data*: The correct clusterings of the LCSS method using complete linkage.

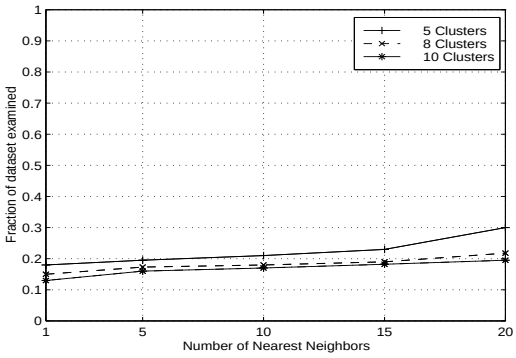

Figure 9. Performance for increasing number of Nearest Neighbors.

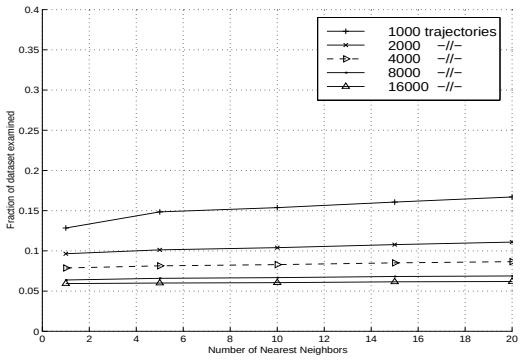

Figure 10. The pruning power increases along with the database size.

jectories. 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 9 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 [9, 6, 21]. 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 [6, 36].

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 10. As the size of the database increases, we can avoid examining a larger fraction of the database.

6 Related Work

The simplest approach to define the similarity between two sequences is to map each sequence into a vector and then use a p-norm distance to define the similarity measure. The p-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 $p = 1$ the Manhattan distance. Various approaches have used, extended and indexed this distance metric [2, 37, 18, 14, 10, 32, 10, 20, 24, 23].

Another approach is based on the time warping technique that first has been used to match signals in speech recognition [33]. Berndt and Clifford [5] proposed to use this technique to measure the similarity of time-series data in data mining. Recent works have also used this similarity measure [25, 28].

A similar technique is to find the longest common subsequence ($LCSS$) of two sequences and then define the distance using the length of this subsequence [3, 7, 11]. The $LCSS$ shows how well the two sequences can match one another if we are allowed to stretch them but we cannot rearrange the sequence of values. Since the values are real numbers, we typically allow approximate matching, rather than exact matching. In [7, 11] fast probabilistic algorithms to compute the $LCSS$ of two time series are presented.

Other techniques to define time series similarity are based on extracting certain features (Landmarks [29] or signatures [13]) from each time-series and then use these features to define the similarity. An interesting approach to represent a time series using the direction of the sequence at regular time intervals is presented in [31]. Ge and Smyth [17] present an interesting alternative approach for sequence similarity that is based on probabilistic matching. A domain independent framework for defining queries in terms of similarity of objects is presented in [22].

Note that all the above work deals mainly with one dimensional time-series. The most related paper to our work is the Bozkaya et al. [8]. They discuss how to define similarity measures for sequences of multidimensional points using a restricted version of the edit distance which is equivalent to the $LCCS$. Also, they present two efficient methods to index the sequences for similarity retrieval. However, they focus on sequences of feature vectors extracted from images and not trajectories and they do not discuss transformations or approximate methods to compute the similarity. In another recent work, Lee et al. [27] propose methods to index sequences of multidimensional points. They extend the ideas presented by Faloutsos et al. in [15] and the similarity model is based on the Euclidean distance.

A recent work that proposes a method to cluster trajectory data is due to Gaffney and Smyth [16]. 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. Also, it implicitly assumes that the data (trajectories) follow some basic models which are not easy to find and describe in real datasets.

Lately, there has been some work on indexing moving objects to answer spatial proximity queries (range and nearest neighbor queries) [26, 1, 34]. Also in [30], Pfoser et al. present index methods to answer topological and navigational queries in a database that stores trajectories of moving objects. However these works do not consider a global similarity model between trajectories but they concentrate on finding objects that are close to query locations during a time instant, or time period that is also specified by the query.

7 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 with 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[14]. The challenge of course is to find an embedding that approximately preserves the original pairwise distances and gives good approximate results to similarity queries.

References

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. *In Proc. of the 19th ACM Symp. on Principles of Database Systems (PODS)*, pages 175–186, 2000.

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

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

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

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

[6] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proc of ICDT, Jerusalem*, pages 217–235, 1999.

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

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

[9] K. Chakrabarti and S. Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. *In Proc. of VLDB*, pages 89–100, 2000.

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

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

[12] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, Inc., 1973.

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

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

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

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

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

[18] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. *In Proc. of 25th VLDB*, pages 518–529, 1999.

[19] J. Gips, M. Betke, and P. Fleming. 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*, pages 98–100, Orlando, FL, July 2000.

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

[21] J. Goldstein and R. Ramakrishnan. Contrast plots and p-sphere trees: Space vs. time in nearest neighbour searches. In *Proc. of the VLDB, Cairo*, pages 429–440, 2000.

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

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

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

[25] E. Keogh and M. Pazzani. Scaling up Dynamic Time Warping for Datamining Applications. *In Proc. 6th Int. Conf. on Knowledge Discovery and Data Mining, Boston, MA*, 2000.

[26] G. Kollios, D. Gunopulos, and V. Tsotras. On Indexing Mobile Objects. *In Proc. of the 18th ACM Symp. on Principles of Database Systems (PODS)*, pages 261–272, June 1999.

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

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

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

[30] D. Pfoser, C. Jensen, and Y. Theodoridis. Novel Approaches in Query Processing for Moving Objects. *In Proceedings of VLDB, Cairo Egypt*, Sept. 2000.

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

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

[33] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-26(1):43–49, Feb. 1978.

[34] S. Saltenis, C. Jensen, S. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. *In Proceedings of the ACM SIGMOD*, pages 331–342, May 2000.

[35] M. Vlachos. Indexing similar trajectories. Technical report, 2001.

[36] R. Weber, H.-J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity Search Methods in High-Dimensional Spaces. In *Proc. of the VLDB, NYC*, pages 194–205, 1998.

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