Robust and Fast Similarity Search for Moving Object Trajectories

Lei Chen, M. Tamer Özsu
University of Waterloo
School of Computer Science
{l6chen,tozsu}@uwaterloo.ca

Vincent Oria
New Jersey Inst. of Technology
Dept. of Computer Science
oria@homer.njit.edu

ABSTRACT

An important consideration in similarity-based retrieval of moving object trajectories is the definition of a distance function. The existing distance functions are usually sensitive to noise, shifts and scaling of data that commonly occur due to sensor failures, errors in detection techniques, disturbance signals, and different sampling rates. Cleaning data to eliminate these is not always possible. In this paper, we introduce a novel distance function, Edit Distance on Real sequence (EDR) which is robust against these data imperfections. Analysis and comparison of EDR with other popular distance functions, such as Euclidean distance, Dynamic Time Warping (DTW), Edit distance with Real Penalty (ERP), and Longest Common Subsequences (LCSS), indicate that EDR is more robust than Euclidean distance, DTW and ERP, and it is on average 50% more accurate than LCSS. We also develop three pruning techniques to improve the retrieval efficiency of EDR and show that these techniques can be combined effectively in a search, increasing the pruning power significantly. The experimental results confirm the superior efficiency of the combined methods.

1. INTRODUCTION

With the growth of mobile computing and the development of computer vision techniques, it has become possible to trace the trajectories of moving objects in real life and in videos. A number of interesting applications are being developed based on the analysis of trajectories. For example, using a remote sensing system, and by mining the trajectories of animals in a large farming area, it is possible to determine migration patterns of certain groups of animals. In sports videos, it is quite useful for coaches or sports researchers to know the movement patterns of top players. In a store surveillance video monitoring system, finding the customers' movement patterns may help in the arrangement of merchandise. The basis of all these applications are robust and accurate methods to determine similarity among trajectories.

The trajectory S of a moving object is defined as a sequence of pairs, $S = [(t_1, s_1) \ldots, (t_n, s_n)]$, which show the successive positions of the moving object over a period of time [5, 6]. Here, n, the number of sample timestamps in S, is defined as the *length* of S and s_i is a vector of arity d (d usually equals 2 or 3) that is sampled at timestamp t_i. Therefore, trajectories can be considered as two (x-y plane) or three (x-y-z plane) dimensional time series data. In terms of similarity-based retrieval, we are interested in the movement shape of the trajectories; sequences of sampled vectors are important in measuring the similarity between two trajectories and time components can be ignored. This separates similarity-based retrieval from queries in spatio-temporal databases where time components of trajectories are important to answer time slice or time interval queries [28]. Considerable research has been conducted on similarity-based retrieval on one-dimensional time series data, such as stock or commodity prices, sales volume, weather data and biomedical measurements (e.g. [1, 24, 20, 23, 40]). Unfortunately, the distance functions and indexing methods proposed for one-dimensional time series data can not be directly applied to moving object trajectories due to their unique characteristics.

- Trajectories are usually two or three dimensional data sequences and a trajectory data set often contain trajectories with different lengths. Most of the earlier proposals on similarity-based time series data retrieval are focused on one-dimensional time series data [1, 24, 23, 20, 40].
- Trajectories usually have many outliers. Unlike stock, weather, or commodity price data, trajectories of moving objects are captured by recording the positions of the objects from time to time (or tracing moving objects from frame-to-frame in videos). Thus, due to sensor failures, disturbance signals or errors in detection techniques, many outliers may appear. Longest Common Subsequences (LCSS) has been applied to address this problem [36]; however, it does not consider various *gap* between similar subsequences, which leads to inaccuracy. The gap refers to a sub-trajectory in between two identified similar components of two trajectories.
- Similar movement patterns may appear in different regions of trajectories. Different sampling rates of tracking and recording devices combined with different speeds of the moving objects may introduce *local shifts* into trajectories (i.e., the trajectories follow similar paths, but certain sub-paths are shifted in time). Even though the similarity measures, such as Dynamic Time Warping (DTW) [41, 8, 19], and Edit distance with Real Penalty (ERP) [6], can be used to measure the similarity between trajectories with local shifts, they are sensitive to noise.

Since existing similarity measures can not readily be used to retrieve trajectories, in this paper, we introduce a novel distance function that addresses the peculiarities of trajectories, and we discuss the retrieval efficiency issues relative to this distance function.

The major contributions of this paper are the following:

SIGMOD 2005 June 14-16, 2005, Baltimore, Maryland, USA.
Copyright 2005 ACM 1-59593-060-4/05/06 $5.00.

1. We introduce a novel distance function, *Edit Distance on Real sequence (EDR)*, to measure the similarity between two trajectories. EDR is based on edit distance on strings, and removes the noise effects by quantizing the distance between a pair of elements to two values, 0 and 1. Seeking the minimum number of edit operations required to change one trajectory to another offers EDR the ability to handle local time shifting. Furthermore, assigning penalties to the unmatched parts improves its accuracy. Through a set of objective tests on benchmark data, we show that EDR is more robust than Euclidean distance, DTW, and ERP, and and more accurate than LCSS when it is used to measure the similarity between trajectories that contain noise and local time shifting.

2. We develop three pruning techniques – mean value Q-grams, near triangle inequality, and trajectory histogram – to improve the retrieval efficiency of EDR. Unlike the pruning methods proposed for LCSS [36, 37] or DTW [19], these pruning methods do not require setting constraints on warping length (or matching region) between two trajectories, and therefore, offer users more flexibility.

3. We show how to combine the three pruning methods to significantly reduce the number of false candidates. Furthermore, we develop different variations of the three pruning methods and compare their performance in terms of pruning power and speedup ratio and we show the superior searching efficiency of the combined methods.

The rest of the paper is arranged as follows: we give a brief review of existing distance functions in Section 2, which motivates the necessity of a new distance function. Section 3 presents the new distance function EDR, as well as comparative efficacy test results on benchmark data sets. In Section 4, we introduce three pruning techniques and their variations that can be used to improve the retrieval efficiency. An optimization is also proposed by combining three pruning methods. Experimental studies on retrieval efficiency in terms of pruning power and speedup ratio for each pruning technique and the combination method are presented in Section 5. We conclude by comparing our approach with related work in Section 6.

2. BACKGROUND

In this paper, for simplicity and without loss of generality, we assume that objects are points that move in a two-dimensional space ($x - y$ plane) and that time is discrete. Thus, given a trajectory $S = [(t_1, s_1) \ldots, (t_n, s_n)]$, s_i is a pair, $(s_{i,x}, s_{i,y})$. We refer to (t_i, s_i) as an *element* of trajectory S. All the definitions, theorems, and techniques can be extended to more than two dimensions. Given S, we can normalize its x and y position values using the corresponding mean (μ_x), (μ_y) and standard deviation (σ_x), (σ_y), respectively [13]: $Norm(S) = [(t_1, (\frac{s_{1,x}-\mu_x}{\sigma_x}, \frac{s_{1,y}-\mu_y}{\sigma_y})), \ldots, (t_n, (\frac{s_{n,x}-\mu_x}{\sigma_x}, \frac{s_{n,y}-\mu_y}{\sigma_y}))]$. Normalization is recommended so that the distance between two trajectories is invariant to spatial scaling and shifting. Throughout this paper, we use S to denote $Norm(S)$. Figure 1 summarizes the main symbols used in this paper and Figure 2 lists the existing distance functions that we will review in this section.

Symbols	Meaning
S	a trajectory $[(t_1, s_1), \ldots, (t_n, s_n)]$
s_i	ith element vector of S
$dist(r_i, s_i)$	the distance between two elements (r_i, s_i) and (r_i, s_i)
$s_{i,x}$	the x coordinate of ith element vector of S
$Rest(S)$	the sub-trajectory of S without the first element: $[(t_2, s_2), \ldots, (t_n, s_n)]$
H_S	a histogram of trajectory

Figure 1: Meanings of symbols used

Given two trajectories R and S of length n, the Euclidean distance between them, $Eu(R, S)$, is defined as Formula 1 in Figure 2. Euclidean distance requires trajectories to be the same length. Dynamic time warping distance, $DTW(R, S)$, between two trajectories R and S of length m and n, respectively, is defined as Formula 2 in Figure 2. DTW does not require two trajectories to be the same length, and it can handle the local time shifting by duplicating the previous element. Edit distance with Real Penalty, $ERP(R, S)$, defined by Formula 3 in Figure 2, introduces a constant value g as the gap of edit distance and uses real distance between elements as the penalty to handle local time shifting. Euclidean distance and ERP are metric and they obey triangle inequality, therefore, they can be indexed by known distance access methods, while DTW is not. However, Euclidean distance, DTW, and ERP are all sensitive to noise. To illustrate this, let us consider the following example of four one-dimensional trajectories: $Q = [(t_1, 1), (t_2, 2), (t_3, 3), (t_4, 4)], R = [(t_1, 10), (t_2, 9), (t_3, 8), (t_4, 7)]$, $S = [(t_1, 1), (t_2, 100), (t_3, 2), (t_4, 3), (t_5, 4)], P = [(t_1, 1), (t_2, 100)$, $(t_3, 101), (t_4, 2), (t_5, 4)]$. Assume that Q is the query trajectory, and the second element of S as well as the second and third elements of P are noise (their values are significantly different from the values near them). The correct ranking in terms of similarity to Q is: S, P, R, since, except noise, the rest of the elements of S and P match the elements of Q perfectly. Euclidean distance ranks the three trajectories as R, S, P. DTW and ERP produce the same rank as Euclidean distance, however, even from general movement trends (subsequent values increase or decrease) of the two trajectories, it is obvious that S is more similar to Q than R. The example shows that noise can cause similar trajectories to be treated as dissimilar when noise-sensitive distance functions are used.

The LCSS score of two trajectories R and S of length m and n is computed according to Formula 4 in Figure 2. LCSS requires a threshold ϵ to be established. This threshold is used to determine whether or not two elements match and allows LCSS to handle noise by quantizing the distance between two elements to two values, 0 and 1, to remove the larger distance effects caused by noise. However, it does not consider variations in gap sizes between two similar subsequences of the trajectories. We use the same example given above to illustrate this point. Assume $\epsilon = 1$, LCSS ranks three trajectories in terms of their similarities to Q as $S = P, R$ ("=" means that S and P have the same distance to Q). However, we know that the gap between common subsequences of P is longer than that of S, and S is more similar to Q than P.

Figure 2 also compares four distance functions based on four criteria: ability to handle sequences with local time shifting, ability to handle sequences that contain noise, whether the distance function is a metric, and computation cost. Using these distance functions to measure the similarity between two trajectories has the following problems:

- Euclidean distance, DTW and ERP are all sensitive to noise, which occurs in trajectory data.
- Euclidean distance can not handle trajectories with local time shifting and different lengths.
- LCSS can handle trajectories with noise, but it is a very "coarse" measure, as it does not differentiate trajectories with similar common subsequences but different sizes of gaps in between.

3. EDR: A NEW DISTANCE FUNCTION

In this section, we propose a new distance function, called *Edit Distance on Real sequence* (EDR), to tackle the problems encoun-

Definition	Local Time Shifting	Noise	Metric	Computation Cost
$Eu(R,S) = \sqrt{\sum_{i=1}^{n} dist(r_i, s_i)} \quad dist(r_i,s_i) = (r_{i,x} - s_{i,x})^2 + (r_{i,y} - s_{i,y})^2$ (1)			✓	$O(n)$
$DTW(R,S) = \begin{cases} 0 & \text{if } m = n = 0 \\ \infty & \text{if } m = 0 \text{ or } n = 0 \\ dist(r_1,s_1) + min\{DTW(Rest(R), Rest(S)), & \text{otherwise} \\ DTW(Rest(R),S), DTW(R, Rest(S))\} & \end{cases}$ (2)	✓			$O(n^2)$
$ERP(R,S) = \begin{cases} \sum_1^n dist(s_i, g) & \text{if } m = 0 \\ \sum_1^m dist(r_i, g) & \text{if } n = 0 \\ min\{ERP(Rest(R), Rest(S)) + dist(t_1, s_1), & \\ ERP(Rest(R), S) + dist(r_1, g), & \text{otherwise} \\ ERP(R, Rest(S)) + dist(s_1, g)\} & \end{cases}$ (3)	✓		✓	$O(n^2)$
$LCSS(R,S) = \begin{cases} 0 & \text{if } m = 0 \text{ or } n = 0 \\ LCSS(Rest(R), Rest(S)) + 1 & \text{if } \lvert r_{1,x} - s_{1,x}\rvert \leq \epsilon \\ & \& \lvert r_{1,y} - s_{1,y}\rvert \leq \epsilon \\ max\{LCSS(Rest(R), S), LCSS(R, Rest(S))\} & \text{otherwise} \end{cases}$ (4)	✓	✓		$O(n^2)$

Figure 2: Distance Functions

tered by the existing distance functions, as we reviewed in the previous section. EDR is more robust and accurate than the existing ones in measuring the similarity between two trajectories.

3.1 Edit Distance on Real Sequences

EDR is based on Edit Distance (ED) [26], which is widely used in bio-informatics and speech recognition to measure the similarity between two strings. Given two strings A and B, $ED(A, B)$ is the number of insert, delete, or replace operations that are needed to convert A into B. Since trajectories are not strings but numerical value pair sequences, for EDR, it is crucial to properly define *matching* between element pairs of different trajectories.

Definition 1. A pair of trajectory element vectors r_i and s_j from two trajectories R and S, respectively, are said to *match* ($match(r_i, s_j) = true$) if and only if $|r_{i,x} - s_{j,x}| \leq \epsilon$ and $|r_{i,y} - s_{j,y}| \leq \epsilon$, where ϵ is the matching threshold.

Definition 2. Given two trajectories R and S of lengths n and m, respectively, the *Edit Distance on Real sequence* (EDR) between R and S is the number of insert, delete, or replace operations that are needed to change R into S. $EDR(R, S)$ is defined as follows:

$$EDR(R,S) = \begin{cases} n & \text{if } m = 0 \\ m & \text{if } n = 0 \\ min\{EDR(Rest(R), Rest(S)) + subcost, & \\ \quad EDR(Rest(R), S) + 1, EDR(R, Rest(S)) + 1\} & \\ & \text{otherwise} \end{cases}$$

where $subcost = 0$ if $match(r_1, s_1) = true$ and $subcost = 1$ otherwise.

In Definition 2, we assume that the cost of a replace, insert, or delete operation is only 1, which corresponds to the original definition of edit distance [26]. Compared to Euclidean distance, DTW, ERP, and LCSS, EDR has following virtues:

- In EDR, the matching threshold reduces effects of noise by quantizing the distance between a pair of elements to two values, 0 and 1 (LCSS also performs the same quantization). Therefore, the effect of outliers on the measured distance is much less in EDR than that in Euclidean distance, DTW, and ERP.
- Like ERP, seeking the minimum number of edit operations required to change one trajectory to another offers EDR the ability to handle local time shifting.
- Contrary to LCSS, EDR assigns penalties to the gaps between two matched sub-trajectories according to the lengths of gaps, which makes it more accurate than LCSS.

Revisiting the previous example, the similarity ranking relative to Q with EDR ($\epsilon = 1$) is S, P, R, which is the expected result.

3.2 Evaluation of EDR

In order to compare the efficacy of different distance functions, we apply the following objective evaluation. In the first test, we test the efficacy of EDR using the approach in [36]. Specifically, we perform hierarchy clustering using four distance functions on two labelled data sets. The two labelled trajectory data sets are the "Cameramouse" (CM) [11] and the Australian Sign Language (ASL) data sets which were also used in [22, 36]. The "Cameramouse" data set contains 15 trajectories of 5 words (3 for each word) obtained by tracking the finger tips of people as they "write" various words. The ASL data set from UCI KDD data archive[1] consists of samples of Australian Sign Language signs, and it is a 10 class data set with 5 trajectories per class[2]. For each data set, we take all possible pairs of classes and use the "complete linkage" hierarchy clustering algorithm [16], which was reported to produce the best clustering results [36], to partition them into two clusters. We draw the dendrogram of each clustered result to see whether it correctly partitions the trajectories. We run the experiments with different values of ϵ and find that setting the matching threshold ϵ to be a quarter of the maximum standard deviation of trajectories leads to the best clustering results, which is also confirmed by [33]. The same ϵ value is used for LCSS and EDR. In order to make a fair comparison with DTW [29], we also test DTW with different warping lengths and report the best results. Since Euclidean distance requires sequences with the same length, we apply the strategy used in [36], where the shorter of the two trajectories slides along the longer one and the minimum distance is recorded. The best result of each distance function is reported in Table 1.

Correct results	Eu	DTW	ERP	LCSS	EDR
CM (total 10 correct)	2	10	10	10	10
ASL (total 45 correct)	4	20	21	21	21

Table 1: Clustering results of five distance functions

As shown in Table 1, EDR performs as well as DTW, ERP and LCSS. The poor clustering results of Euclidean distance confirm that it is very sensitive to local time shifting. In this test, DTW and ERP perform similar to LCSS and EDR, because the two trajectory data sets contain local time shifting, but very little or no noise, which confirms results in [36]. Thus, this test shows that EDR is as effective as DTW, ERP, and LCSS in measuring similarities of trajectories when the trajectories contain little or no noise.

The second test uses classification of labelled data to evaluate the efficacy of a distance function, as proposed by Keogh et al. [21]. Specifically, each trajectory is assigned a class label. Then the "leave one out" prediction mechanism is applied to each trajectory

[1] University of California, Irvine: http://kdd.ics.uci.edu.

[2] The test data are available at http://db.uwaterloo.ca/~l6chen/data

in turn. That is, the class label of the chosen trajectory is predicted to be the class label of its nearest neighbor, defined based on the given distance function. If the prediction is correct, then it is a hit; otherwise, it is a miss. The classification error rate is defined as the ratio of the number of misses to the total number of trajectories. We use the same two data sets of the first test.

In order to test the ability of distance functions to handle local time shifting and noise, we add to three data sets interpolated Gaussian noise (about 10-20% of the length of trajectories) and local time shifting using the program in [37]. To get average values over a number of data sets, we use each raw data set as a seed and generate 50 distinct data sets that include noise and time shifting. The results are shown in Table 2. For two data sets, EDR performs the best, showing that it is superior to the other distance functions in handing noise and local time shifting.

Avg. Error Rate	Eu	DTW	ERP	LCSS	EDR
CM	0.25	0.14	0.14	0.10	0.03
ASL	0.28	0.18	0.17	0.14	0.09

Table 2: Classification results of five distance functions

To conclude, the results of above two tests prove that *EDR performs as well as DTW, ERP, and LCSS when trajectories contain little or no noise, and it is more robust than DTW and ERP, and on average* 50% *more accurate than LCSS in noisy conditions*. In terms of efficiency, the computation cost of DTW, ERP, LCSS, and EDR is the same, which are quadratic using dynamic programming.

4. EFFICIENT TRAJECTORY RETRIEVAL USING EDR

The matching threshold ϵ in EDR is introduced to remove the effect of noise. However, the introduction of the threshold causes EDR to violate triangle inequality, making it non-metric and, thus, non-indexable by traditional distance-based indexing methods. However, this does not mean that EDR is not a "good" distance function. As pointed out by Jacobs et. al [15], it is not the poor selection of features or careless design that cause a distance function not to follow triangle inequality. Inherently, distance functions that are robust to noisy data will usually violate triangle inequality. Many robust distance functions have been proposed in the domain of image retrieval, such as Hausdorff distance [14] and Dynamic Partial Function (DPF) [12], that do not follow triangle inequality. Furthermore, much work in psychology also suggests that human similarity judgements do not follow triangle inequality either [32]. Therefore, given a "good", robust but non-metric distance function, the issue is how to improve the retrieval efficiency for similarity search. The computation cost of EDR by dynamic programming is $O(m * n)$, where m and n are the lengths of the two trajectories (the cost of DTW, ERP, and LCSS are quadric as shown in Figure 2). This removes the possibility of using sequential scan when the database size is large. We, therefore, propose three pruning methods that can reduce the number of computations between the query trajectory and trajectories in the database. We have a strong requirement that the methods cause no false dismissals while reducing false candidates. The target queries are k-NN queries, which return k data elements from a database that have the nearest distances to the query data. The pruning techniques that we propose in this paper can also be applied to LCSS, the details are omitted due to space limitation. Extension to other non-metric distance functions are possible, but not trivial and is left as future work.

4.1 Pruning by Mean Value Q-gram

Given a string S, a Q-gram of S is defined as a substring of size q. Q-grams have been well studied as a solution to the approximate string matching problem [17, 31, 10], which is defined as follows: given a long text of length n and a pattern of length m ($m \leq n$), retrieve all the segments of the text whose *edit distance* to the pattern is at most k. If a pattern and a string are similar to each other, the number of substrings that are common to each other is high. This is the intuition of using Q-grams as a filter. The following theorem can be used to remove the segments that do not satisfy the requirement (at most k editing operations) before computing the real edit distance between the pattern and the segment.

Theorem 1 [17]. Let P and S be strings of length m and n. P and S within edit distance k have at least $p = max(m, n) - q + 1 - kq$ common Q-grams.

The value p comes from two parts: first part, $max(m, n) - q + 1$, is the maximum number of Q-grams of size q in P or S, and the second part, kq, is the maximum number of Q-grams that can be affected between P and S by k edit operations.

Theorem 1 can be used in the context of EDR to remove false candidates, but changes are required since we are not looking for exact match in counting common Q-grams between trajectories. Thus, what it means to "match" has to be redefined as follows.

Definition 3. Given two Q-grams $r = [(r_{1,x}, r_{1,y}), \ldots, (r_{q,x}, r_{q,y})]$ and $s = [(s_{1,x}, s_{1,y}), \ldots, (s_{q,x}, s_{q,y})]$ of trajectories R and S, respectively, r *matches* s if and only if each element of r *matches* its corresponding element in s.

However, the space requirement to store Q-grams is very high, since each Q-gram of a trajectory has to be stored. Furthermore, Theorem 1 only applies to one-dimensional strings, and naive implementation of Q-grams on multi-dimensional trajectories will not only increase the space cost but may also suffer the dimensionality curse problem [38]. Finally, Theorem 1 applies only to range queries (searching strings with at most k edit operation to the query string). In most cases, users may not know the range a priori. In these situations, k-NN search is more meaningful. In the rest of this section, we present solutions to these issues.

Compared to strings, elements of trajectories are real values; thus, we can use the properties of real values to reduce the storage requirement of Q-grams. Based on Definitions 1 and 3, we have the following theorem:

Theorem 2. Given a matching threshold ϵ, if two Q-grams $r = [(r_{1,x}, r_{1,y}), \ldots, (r_{q,x}, r_{q,y})]$ and $s = [(s_{1,x}, s_{1,y}), \ldots, (s_{q,x}, s_{q,y})]$ *match*, their mean value pairs $r_{mean} = (\frac{\sum r_{i,x}}{q}, \frac{\sum r_{i,y}}{q})$ and $s_{mean} = (\frac{\sum s_{i,x}}{q}, \frac{\sum s_{i,y}}{q})$ also *match*.

Proof. Straightforward induction from Definition 1. □

Based on Theorem 2, we need no more space to store Q-grams than that is required to store a trajectory, regardless of the size of the Q-gram. Most importantly, there is the possibility to index the Q-grams of trajectories with less dimensions. For example, given a trajectory $S = [(t_1, (1, 2)), (t_2, (3, 4)), (t_3, (5, 6)), (t_4, (7, 8)), (t_5, (9, 10))]$, Q-Grams of size 3 for S are: [(1,2), (3, 4), (5, 6)], [(3,4), (5,6), (7, 8)], [(5,6), (7,8), (9,10)]. We need to create a six-dimensional R-tree to index these Q-grams and with the increasing of Q-gram sizes, the dimensionality of the R-tree will grow (e.g. for three dimensional trajectories, Q-grams of size 5 need a 15 dimensional R-tree) and may cause R-tree to perform worse than sequential scan [38]. However, the mean value Q-gram pairs of S are: (3,4), (5,6), (7,8). Only a two dimensional R-tree is needed to index these pairs, even for Q-grams with larger size.

Q-grams were originally proposed as a filtering technique in answering range queries. We propose two algorithms to extend this technique to answer k-NN queries. The first algorithm utilizes indexes on Q-grams to speed up the process of finding common Q-grams between two trajectories. Procedure Qgramk-NN-index

```
Procedure Qgram$k$-NN-index($Q, k, Tree, result$) {
  /* $Tree \equiv$ a R*-tree storing mean value pairs for all Q-grams in database */
  (1) for each Q-gram $q$ of trajectory Q {
  (2)       $q_{mean} = mean(q)$
  (3)       conduct a standard R*-tree search on $Tree$ using $q_{mean}$
  (4)       increase the Q-gram counter for the trajectory that
                 have a match mean value pair to $q_{mean}$
        }
  (5) sort the Q-gram counters of trajectories in descending order.
  (6) pick the first $k$ trajectories pointed by Q-gram counters and
            initialize $result$ with the $k$ true (sorted) $EDR$ distances
  (7) let $v_i, \ldots, v_n$ be the data values of Q-gram counters
            starts from $i = k + 1$ and $l_Q$ be the length of query trajectory
  (8) for each $v_i$ {
  (9)       $bestSoFar = result[k].dist$ /* the $k$-NN distance so far */
  (10)      if (($v_i$) $\geq$ ($max(l_Q, l_S) - (bestSoFar + 1) * size(Q - gram)$)
                 /* need to check */
  (11)          for each trajectory $S$ pointed by $v_i$ {
  (12)              $realDist = EDR(Q, S)$ /* compute true distance */
  (13)              if ($realDist < bestSoFar$) { /* update $result$ */
  (14)                  insert $S$ and $realDist$ into $result$,
                          sorted in ascending order of EDR distance
  (15)                  $bestSoFar = result[k].dist$
                 } /* end-if, line 13 */
            } /* end-for, line 11 */
  (16)      else break /* else, line 10, skip the rest */
        } /* end-for, line 8 */
  (17) return result
}
```

Figure 3: Algorithm for applying Q-grams to answer k-NN query with indexes

(Figure 3) lists steps of the first algorithm. The algorithm first conducts a standard search for each mean value pair q_{mean} of the Q-grams in Q and updates the corresponding Q-gram counter for each trajectory in the database. The Q-gram counters are then sorted in descending order and the first k trajectories pointed by the first k elements of the Q-gram counters are used to initialize the $result$ array. Finally values in the rest of the Q-gram counters are visited in descending order. If the value satisfies the inequality stated in Theorem 1, the true distance $EDR(Q, S)$ is computed and updates to the result list are made if necessary. Otherwise, the remaining data values can be skipped entirely.

Theorem 3. Using procedure Qgramk-NN-index to answer a k-NN query does not introduce false dismissals.

Proof. We prove by contradiction. Assume that Qgramk-NN-index introduces false dismissals; then the following two statements are valid: (1) v_i is the Q-gram counter value of trajectory S (length n) and $v_i < max(m, n) + 1 - (bestSoFar + 1) * size(Qgram)$, and (2) $EDR(Q, S) < bestSoFar$. It is sufficient to show one of them to be wrong. According to Theorem 1, we get $v_i \geq max(m, n) + 1 - (EDR(Q, S) + 1) * size(Qgram)$. Based on statement (2), we know that $v_i \geq max(m, n) + 1 - (bestSoFar + 1) * size(Qgram)$, which contradicts (1). □

Furthermore, in line 7 of the Procedure Qgramk-NN-index, the Q-gram counters are visited in descending order,which also guarantees that skipping the rest of the elements in line 16 will not introduce false dismissals. This is because, if $v_i \geq v_{i+1}$, and $v_i < max(m, n) + 1 - (bestSoFar + 1) * size(Qgram)$, then $v_{i+1} < max(m, n) + 1 - (bestSoFar + 1) * size(Qgram)$.

Procedure Qgramk-NN-index utilizes an index on mean values of Q-grams to find common Q-grams between the query trajectory and each data trajectory in the database. The computation cost of this pruning step (not including sorting) is $O(l * \log(N * l_{max}))$, where N is the size of the database, l is the length of the query trajectory, and l_{max} is the maximum length of trajectories in the database. However, when the database size N increases, the index on Q-grams grows and the search operation on the index becomes expensive, which may increase the total execution time of each query as a consequence.

The second algorithm applies merge join on sorted Q-grams of trajectories to find the common Q-grams between them without any indexes. The full algorithm is given in [7]. The computation cost of this pruning step is only $O(l + l_{max})$.

The above two algorithms apply mean value Q-gram filters directly on trajectories to reduce the number of EDR calculations. Another possibility is to take the projection of the trajectory on each dimension, which produces a single dimensional data sequence, and apply Q-gram filters to them. Of course, care needs to be taken to ensure that no false dismissals will occur.

Theorem 4. Let R and S be trajectories of length m and n. If $EDR(R, S) \leq k$, the number of common Q-grams between two single dimensional data sequences R^x, S^x (or R^y, S^y) obtained by projecting R and S over dimension x (or y) is at least $max(m, n) - q + 1 - kq$.

Proof. We only need to prove that the claim holds for one of the dimensions and we prove for x dimension. From Definition 3, if two Q-grams $r = [(r_{1,x}, r_{1,y}), \ldots, (r_{q,x}, r_{q,y})]$ and $s = [(s_{1,x}, s_{1,y}), \ldots, (s_{q,x}, s_{q,y})]$ of trajectories R and S match, each element of r_i matches its corresponding element in s_i. As a consequence, each $r_{i,x}$ matches $s_{i,x}$. Therefore, $p_x \geq p$, where p_x and p are the number of common Q-grams between x dimensional data sequence and trajectories, respectively. From Theorem 1, $p \geq max(m, n) - q + 1 - kq$, thus $p_x \geq max(m, n) - q + 1 - kq$. □

Theorem 4 can be used to remove false candidates (as we did using Theorem 1) without introducing false dismissals. Mean value Q-gram filters can be applied on projected single dimensional data sequences and the false candidates can be removed.

Based on Theorems 2 and 4, we only need to store the mean value of each Q-gram of one-dimensional data sequence. This has the advantage that we can use a simple B$^+$-tree to index mean values of Q-grams or apply merge join algorithm on single dimensional data sequences of trajectories. Compared to indexing Q-grams of trajectories using R-tree or merge join on two dimensional sequences, we save both space and disk access time. However, since only information from one-dimension is used, the pruning power is reduced.

4.2 Pruning by Near Triangle Inequality

As we mentioned before, EDR does not follow the triangle inequality. However, the following property holds.

Theorem 5 (*Near Triangle Inequality*). Given three trajectories Q, S, and R, we have $EDR(Q, S) + EDR(S, R) + |S| \geq EDR(Q, R)$, where $|S|$ is the length of S.

Proof. The first part of left hand side of the inequality, $EDR(Q, S) + EDR(S, R)$, can be viewed as the number of edit operations needed to convert trajectory Q to S and then to convert S to R. $EDR(Q, S) + EDR(S, R)$ may be less than $EDR(Q, R)$, since an element s_i of S may match elements q_i and r_i of Q and R, respectively, but q_i and r_i may not match. In this case, $EDR(Q, R)$ has one more edit operation than that of $EDR(Q, S) + EDR(S, R)$. The extreme case is that all the elements of S match the elements of Q and R, however those matched elements of Q and R do not match each other, thus $EDR(Q, R)$ is larger than $EDR(Q, S) + EDR(S, R)$ by at most $|S|$, which is the second part of the left hand side of the inequality. □

By rewriting the near triangle inequality (Theorem 5), we get $EDR(Q, S) \geq EDR(Q, R) - EDR(S, R) - |S|$. If $EDR(Q, R)$ and $EDR(S, R)$ are known, $EDR(Q, R) - EDR(S, R) - |S|$ can be treated as a lower bound distance of $EDR(Q, S)$. Therefore, near triangle inequality can be used to prune out false candidates. Procedure NearTrianglePruning (Figure 4) gives the algorithm for the application of near triangle inequality to answer a k-NN query.

```
Procedure NearTrianglePruning($S, procArray, pmatrix, result, Q, k$) {
  /* $S \equiv$ the current trajectory; $procArray \equiv$ the array of
  trajectory with computed true distance to $Q$; $pmatrix \equiv$
  precomputed pairwise distance matrix; $result \equiv$ the $k$-NN
  trajectory */
  (1) $maxPruneDist = 0$
  (2) for each trajectory $R$ in $procArray$ {
  (3)     if ( $(procArray[R].dist - pmatrix[R,S] - |S|) > maxPruneDist$)
  (4)         $maxPruneDist = procArray[R].dist - pmatrix[R,S] - |S|$
        } /* end-for, line 2 */
  (5) $bestSoFar = result[k].dist$ /* the $k$-NN distance so far */
  (6) if ($maxPruneDist \leq bestSoFar$) { /* cannot be pruned */
  (7)     $realDist = EDR(Q,S)$ /* compute true distance */
  (8)     insert $S$ and $realDist$ into $procArray$
  (9)     if ($realDist < bestSoFar$) /* update $result$ */
  (10)        insert $S$ and $realDist$ into $result$,
                sorted in ascending order of EDR distance
        } /* end-if, line 6 */
}
```

Figure 4: Algorithm for near triangle inequality pruning

The matrix $pmatrix$ holds the precomputed pairwise EDR distances of the trajectory database. The array $procArray$ stores the true EDR distances computed so far. That is, if $\{R_1, \ldots, R_u\}$ is the set of trajectories for which $EDR(Q, R_i)$ has been computed, the distance $EDR(Q, R_i)$ is recorded in $procArray$. Thus, for trajectory S currently being evaluated, the near triangle inequality ensures that $EDR(Q,S) \geq EDR(Q,R_i) - EDR(R_i,S) - |S|$, for all $1 \leq i \leq u$. Thus, it is necessary that $EDR(Q,S) \geq (max_{1 \leq i \leq u}\{EDR(Q,R_i) - EDR(R_i,S) - |S|\})$ (lines 2 to 4). If distance $maxPruneDist$ is larger than the current k-NN distance stored in $result$, then S can be skipped. Otherwise, the true distance $EDR(Q,S)$ is computed, and $procArray$ is updated to include S. Finally, the $result$ array is updated to reflect the current k-NN neighbours and distances in sorted order. The computation cost of pruning step in NearTrianglePruning is constant, which is the size of $procArray$.

Application of the NearTrianglePruning procedure encounters two issues: (i) the size of the pairwise distance matrix $pmatrix$; and (ii) the size of $procArray$, i.e., the maximum number of trajectories whose true EDR distances are kept for near triangle inequality pruning. We use the dynamic strategies proposed in [6] to resolve these issues and make the procedure practical for large databases and for situations where buffer space is limited.

Let $maxTriangle$ denote the maximum number of trajectories whose true EDR distances are kept for near triangle inequality pruning. Hereafter we call these trajectories the *reference trajectories*. The value of $maxTriangle$ should be determined at query time by the query engine based on the buffer size. The larger $maxTriangle$ is, the more pruning power can be achieved. Dynamic strategies pick these reference trajectories as NearTrianglePruning runs, therefore, the entire $pmatrix$ is not needed. As the reference trajectories are picked and kept, the appropriate column of the distance matrix is read into the buffer space. The buffer space requirement is $maxTriangle$ columns, each of size N (N is the trajectory database size). Thus, the total buffer space required is $N * maxTriangle$. Given a database that contains 1,000,000 trajectories and $maxTriangle = 400$, the buffer space requirement is only around 400M, which is acceptable based on the current hardware configuration of PC. The selection of reference trajectories is query dependent. In our implementation, we simply pick the first $maxTriangle$ trajectories that fill up $procArray$.

We should note that the near triangle inequality is a "weak" version of triangle inequality, as it filters only when trajectories have different lengths (both query and data trajectories). If all the trajectories have the same length, applying near triangle inequality will not remove any false candidates.

We also investigate a general approach, called *Constant Shift Embedding* (CSE) [30], to convert a distance function that does not follow triangle inequality to another one that follows. The idea is as follows.

Given a distance function $dist$ that is defined on data space $\mathcal{D}$ and does not follow triangle inequality, there exist three data elements $x, y, z \in \mathcal{D}$, such that $dist(x,y) + dist(y,z) < dist(x,z)$. $dist$ can be converted to another distance function, $dist'$, by adding a positive value c to each distance value calculated by $dist$. For example, $dist'(x,y) = dist(x,y) + c$. If c is large enough, we may have $dist'(x,y) + dist'(y,z) \geq dist'(x,z)$ (which equals $dist(x,y) + dist(y,z) + c \geq dist(x,z)$), thus, triangle inequity can be hold on $dist'$.

However, we do not apply CSE approach to improve trajectory retrieval efficiency due to the following reasons:

1. All the pairwise distances in the data set have to be investigated to find c. In [30], the c is set to the minimum eigenvalue of pairwise distance matrix. We tested this minimum eigenvalue with some trajectory data sets, such as ASL, Kungfu, and Slip (the details of these data sets are explained in the experiment section), and we found that very few distance computations can be saved. An analysis of the converted pairwise distance matrix showed that this minimum eigenvalue is quite large and makes the pruning by triangle inequality meaningless, since the lower bound of $dist(x,y)$, $(dist(x, z) - dist(y,z) - c)$ is too small to prune anything. Reducing the minimum eigenvalue may increase pruning ability, but it may cause some distances not to follow triangle inequality and introduce false dismissals.
2. Usually, for similarity search, query data are not inside the database. The constant value c derived by only investigating the data in the database may not be large enough to make the distances between query data to any data in the database follow triangle inequality. Using the CSE approach to compute the distances between query data and all the data in the database does not make senses, since it is precisely these distance computations that we want to save.

4.3 Pruning by Histograms

Embedding methods have been used to improve the efficiency of k-NN queries on strings under edit distance. The basic idea is to embed strings into a vector space and define a distance function in the embedded vector space. To avoid false dismissals, the distance in the embedded space is required to be the lower bound of the edit distance on strings. A number of embedding methods have been proposed for strings [2, 3, 9, 18]; however, only two of these [18, 2] avoid introducing false dismissals. Both of these take a similar approach in that they transform strings into a multidimensional integer space by mapping strings to their *frequency vectors* (FV). A frequency vector of a string over an an alphabet records the frequency of occurrence of each character of the alphabet in that string. It is proven that the *frequency distance* (FD) between the FVs of two strings is the lower bound of the actual edit distance. FD of two points u and v in s-dimensional space, $FD(u,v)$, is defined as the minimum number of steps (insertion, deletion, replacement operations) that is required to go from u to v (or equivalently from v to u).

In fact, frequency vectors are one-dimensional histograms over strings, where each bin is a character in the alphabet. Therefore, we propose an embedding technique which transforms trajectories into trajectory histograms and uses histograms to remove false candidates. We first develop two dimensional histograms of trajectories in the following way. Given the maximum (max_x) and minimum

```
Procedure CompHisDist($H_R$, $H_S$) {
  /* $H_R$ and $H_S$ ≡ histograms of trajectories
     $result$ ≡ histogram distance */
  (1) $posDist = 0, negDist = 0$
  (2) for each histogram bin of $H_R$ {
  (3)     $H_{R,i} = H_{R,i} - H_{S,i}$
        } /* end-for, line 2 */
  (4) for each histogram bin of $H_R$ {
  (5)     for each approximately match bin $H_{R,j}$ of $H_{R,i}$ {
  (6)         if the value of $H_{R,j}$ or $H_{R,i}$ have opposite signs {
  (7)             reduce the values of $H_{R,j}$ or $H_{R,i}$
                      /* elements in the approximately match bins
                         should treated as from the same bin */
              } /*end-if line 6 */
          }/*end-for line */
        } /* end-for, line 4 */
  (8) for each histogram bin of $H_R${
  (9)     if $H_{S,i} > 0$ $posDist+ = H_{R,i}$
  (10)    else $negDist+ = 0 - H_{R,i}$
        } /* end-for, line 8 */
  (11) return $result = max(posDist, negDist)$
}
```

Figure 5: Algorithm for Computing Histogram Distances

(min_x) x dimensional values of trajectories, we divide the range $[min_x, max_x]$ into τ_x disjoint equal size subranges and the size of each subrange is ϵ. We do the same on the y dimension to get τ_y disjoint equal size subranges. Any distinct combination of these two subranges is called a *histogram bin*. Given a trajectory S, we can compute its histogram H_S by counting the number of elements h_i ($1 \leq i \leq \tau_x * \tau_y$) that are located in each histogram bin i: $H = [h_1, \ldots, h_{\tau_x * \tau_y}]$.

Based on this embedding, we define a histogram distance DH on histograms of trajectories.

Definition 4. Let H_R and H_S be histograms of two trajectories R and S, respectively. The *histogram distance*, $HD(H_R, H_S)$, is defined as the minimum number of steps required to go from H_R to H_S (or equivalently from H_S to H_R) by moving to a *neighbor* point at each step. H_R and H_S are neighbors if R can be obtained from S (or vice versa) using a single edit operation of EDR.

Since EDR is defined based on a matching threshold ϵ, neighbor points of histograms are different from those of FVs. For example, two FVs $v_1 =< 1, 0 >$ and $v_2 =< 0, 1 >$ are neighbors according to the definition of string edit distance, and frequency distance between them is 1. Given two one-dimensional trajectories $R = [(t_1, 0.9)]$ and $S = [(t_1, 1.2)]$ and $\epsilon = 1$, the histograms of R and S are exactly the same vectors as v_1 and v_2. However, they are not neighbors according to Definition 4. The transformation from R to S does not need any edit operations because 0.9 and 1.2 are treated as matched elements under EDR. Consequently, the corresponding histogram distance is also 0. Therefore, to overcome the problem that elements located near the boundary of two different histogram bins may match each other under EDR, we treat the elements from two different histogram bins as if they were from the same bin if these two histogram bins *approximately match*.

Definition 5. Given two histograms H_R and H_S, histogram bin $h_{R,i}$ of H_R *approximately matches* histogram bin $h_{S,j}$ of H_S, if $h_{R,i}$ and $h_{S,j}$ are the same bin or they are adjacent bins.

For example, given two histograms of $H_R = [h_{R,1}, h_{R,2}, h_{R,3}]$ and $H_S = [h_{S,1}, h_{S,2}, h_{S,3}]$ of two one-dimensional trajectory data, $h_{R,1}$ approximately matches $h_{S,1}$ as well as $h_{S,2}$, and $h_{R,2}$ approximately matches $h_{S,1}$, $h_{S,2}$, and $h_{S,3}$. Figure 5 shows the algorithm for computing HD between two histograms H_R and H_S. In procedure CompHisDist, the first for loop is used to compute the difference between two histograms, the second loop (line 4-7) is used to find the elements in the histogram bins that approximately match each other, and the third loop is used to count the minimum number of steps that is required to transfer H_R to H_S.

Theorem 6. Let R and S be two trajectories, ϵ be a matching threshold and H_R and H_S be the histograms of R and S, respectively. We have $HD(H_R, H_S) \leq EDR(R, S)$.

Proof. Any single edit operation on R to convert it to S corresponds to a change in its histogram H_R: deletion of one element from R corresponds to subtracting one from the value of some histogram bin; insertion of one element to R corresponds to adding one to the value of some histogram bin; replacement of an element in R corresponds to adding one in some bin and subtracting one in the other bin. Furthermore, each movement step that is used to transform H_R to H_S moves H_R to its neighbor point, and the change of H_R made by each movement step is same as that caused by a single edit operation. Thus, the number of steps used in the transformation of the histograms is the lower bound of the number of edit operations in EDR. □

With Theorem 6, to answer k-NN queries, we can compute HDs to prune out false candidates from the trajectory database. Most importantly, the computation cost of HD is linear. The nested for loops in CompHisDis may suggest that the computation time of HD is non-linear. However, as the number of bins that approximately match each other in the histogram space is limited to a small constant, the computation time of CompHisDist is still linear. The algorithm that uses HD as lower bound distance to prune false candidates can be achieved by modifying procedure NearTriAnglePruning (Figure 4) as follows:

- Delete lines (2) to (4) and line (8), as it is no longer necessary to keep the array $procArray$.
- Change line (1) to: $maxPruneDist = HD(H_Q, H_S)$.

This modified algorithm searches trajectory histograms one after another and it does not utilize previously computed histogram distances. We call it Histogram SEquential scan (HSE). We propose another algorithm, Histogram SoRted scan (HSR), to answer k-NN queries. HSR first computes all the histogram distances between query and data trajectories. Then it sorts the histogram distances in ascending order. Finally, the trajectories are accessed according to the histogram distance order and EDR is computed if necessary. It is obvious that the pruning power of HSR is better than that of HSE since the trajectories are accessed in an ascending order of lower bound distances (histogram distances) of EDR. However, to achieve this improvement, HSR requires an additional sorting step. Their relative efficiency is compared in the experiment section.

When we construct histograms, we use ϵ as the histogram bin size. If the matching threshold ϵ is small, we may get trajectory histograms with a lot of bins. The storage and computation cost will increase as a consequence. To address this issue, we propose two solutions to reduce the number of bins:

1. Create histograms with a larger histogram bin size, which is δ ($\delta \geq 2$) times the matching threshold ϵ.
2. Create individual histograms for each one-dimensional data sequence of trajectories using ϵ as the histogram bin size.

Assume that the number of bins for trajectory histograms with bin size ϵ is $\tau_x * \tau_y$, where τ_x, τ_y are the number of histogram bins in each dimension. The two methods above reduce the number of bins by a factor of $\delta * \delta$ and $\frac{\tau_x * \tau_y}{\tau_x + \tau_y}$, respectively. Most importantly, they do not introduce false dismissals based on the following.

Theorem 7. Given two trajectories R, S, and a matching threshold ϵ, we have $EDR_{\delta * \epsilon}(R, S) \leq EDR_{\epsilon}(R, S)$ where $\delta \geq 2$, where $EDR_{\delta * \epsilon}$ stands for EDR computed with $\delta * \epsilon$ as a matching threshold.

Proof. It is clear that if an element of S matches an element of R within ϵ, they must match each other within $\delta * \epsilon$. Thus, the number

```
Procedure EDRCombineK-NN($Q, procArray, pmatrix, result, Q, k$) {
  (1) pick the first $k$ trajectories and
      initialize $result$ with the $k$ true (sorted) $EDR$ distances
  (2) $bestSoFar = result[k].dist$ and $i = k + 1$
  (3) for each trajectory $S_i$ in the database {
  (4)   $maxPruneDist = HD(H_Q, H_{S_i})$ /* trajectory histogram distance*/
  (5)   if ($maxPruneDist \leq bestSoFar$) {
          /* cannot be pruned by trajectory histogram*/
  (6)     $v_i$ =merge-join(mean-value-pairs($Q$), mean-value-pairs($S_i$))
            /* applying merge join to compute common mean value pairs */
  (7)     if (($v_i$) $\geq (max(l_Q, l_{S_i}) - (bestSoFar + 1) * size(Q - gram))$) {
            /* cannot be pruned by mean value Q-grams*/
              invoke procedure NearTrianglePruning() in Figure 4
            } /* end-if, line 7 */
          } /* end-if, line 5 */
      } /* end-for, line 3 */
}
```

Figure 6: Combination Algorithm for Applying Histogram Pruning followed by Mean Value Q-gram and Triangle Inequality Pruning

of matching elements will not be reduced if the matching threshold is increased from ϵ to $\delta * \epsilon$. As a consequence, the number of edit operations needed to convert R to S within $\delta * \epsilon$ is not higher than that of converting within ϵ. □.

Theorem 8. Given two trajectories R and S, and a matching threshold ϵ, we have $EDR^{x,y}_{\epsilon}(R, S) \leq EDR_{\epsilon}(R, S)$ where $EDR^{x,y}_{\epsilon}(R, S)$ is EDR on projected one-dimensional data sequence (x or y) of trajectories.

Proof. Similar to the proof of Theorem 7. If an element of S matches an element of R within ϵ, the individual values of each dimension between two elements must match each other within threshold ϵ. Thus, the number of matching elements is not reduced for each single dimensional data sequence compared to that of whole trajectories. □.

Corollary 1. Let R and S be two trajectories, $H_{(R,\delta*\epsilon)}$ and $H_{(S,\delta*\epsilon)}$ be the histograms created with bin size $\delta * \epsilon$, and $H^x_{(R,\epsilon)}$ and $H^x_{(S,\epsilon)}$ be the histograms on x dimensional data sequence. We have $HD(H_{(R,\delta*\epsilon)}, H_{(S,\delta*\epsilon)}) \leq EDR_{\epsilon}(R, S)$ and $HD(H^x_{(R,\epsilon)}, H^x_{(S,\epsilon)}) \leq EDR_{\epsilon}(R, S)$.

Using Corollary 1, we can use either histograms with larger bin size or histograms on one-dimensional data sequence of trajectories to prune false candidates. In the following experiments, we also compare efficiency of these two methods with those of using trajectory histograms with ϵ as bin size.

4.4 Combination of three pruning methods

Because the three pruning techniques introduced earlier are orthogonal, it is possible to combine three methods – use one pruning method to save the computation of the true distance $EDR(Q, S)$ after another. An example of combination skeleton is shown in Figure 6. In the example procedure EDRCombineK-NN, histogram pruning is applied first, then, mean value Q-gram filters are applied. Finally, the procedure NearTrianglePruning is invoked to remove more false candidates based on computed real EDR distances. In our experiments, we also tried other combinations, such as applying mean value Q-gram filtering before trajectory histograms and near triangle inequality pruning or applying near triangle inequality pruning before the other two. The results are discussed in the experimental section.

5. EXPERIMENTS

We present experimental results regarding the efficiency of each pruning technique as well as the combination of methods. Our experiments measure both speedup ratio and pruning power. Speedup ratio is defined as the ratio between the average total time (including both CPU and I/O measured in seconds) required for a sequential scan and the average total time needed with a pruning technique in answer a k-NN query. Given a k-NN query Q, the pruning power is defined to be the fraction of the trajectories S in the data set for which the true distance $EDR(Q, S)$ is not computed (without introducing false dismissals). Moreover, we vary k from 1 to 20 and the results of $k = 20$ are reported. Since the matching threshold is application dependent [36], we run several probing k-NN queries on each data set with different matching thresholds and choose the one that ranks the results close to human observations. All experiments were run on a Sun-Blade-1000 workstation with 1GB memory under Solaris 2.8.

5.1 Efficiency of Pruning by Q-gram Mean Values

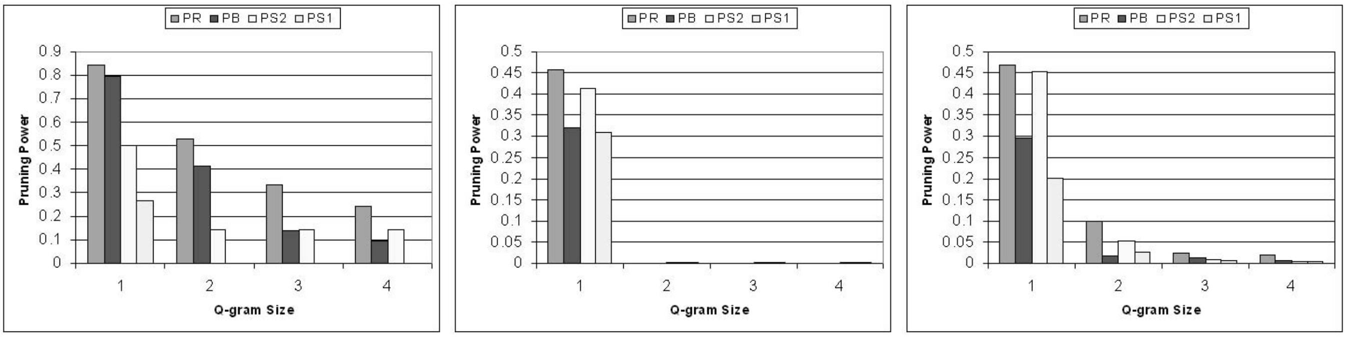
In this experiment, we use ASL data set from UCI KDD archive[3], Kungfu and Slip data sets from [5]. The ASL data set contains 710 trajectories with lengths varying from 60 to 140. The Kungfu data set contains 495 trajectories that record positions of body joints of a person playing kung fu and the length of each trajectory is 640. The Slip data set also has 495 trajectories which record positions of body joints of a person slipping down and trying to stand up and the length of each trajectory is 400. This experiment is designed to compare the pruning efficiency of (1) Q-grams with different sizes, (2) indexed Q-grams versus merge join, and (3) two dimensional Q-grams versus one-dimensional Q-grams.

Figure 7 shows the pruning power comparison of 4 different implementations of pruning by mean values of Q-grams with various sizes (from 1 to 4): pruning with a R-tree on two dimensional Q-grams (PR), pruning with B$^+$-tree on one-dimensional Q-grams (PB), and pruning with merge join on sorted two dimensional Q-grams (PS2) and one-dimensional Q-grams (PS1).

The results of the three data sets show that, in terms of pruning power, PR is better than PB and PS2 is better than PS1, which confirms our previous claim that two dimensional Q-grams perform better than one-dimensional Q-grams. The results also show that with the increasing in size of Q-gram, the pruning power drops, especially for Slip data where the pruning power drops to 0 when the Q-gram size is greater than 1. Thus, Q-grams of size 1 are the most effective ones in removing false candidates. This is because that the larger the size of Q-gram is, the more the number of matching mean value Q-grams, which leads to less pruning power. The results also show that PR is always better than PS2 which indicates pruning with indexed Q-grams is better than pruning with merge join. However, we can not conclude that PR on Q-grams of size 1 is the best pruning method, since the higher pruning power may be more expensive in terms of space and computation cost. Thus, we also compare the speedup ratio of four methods. The results are shown in Figure 8.

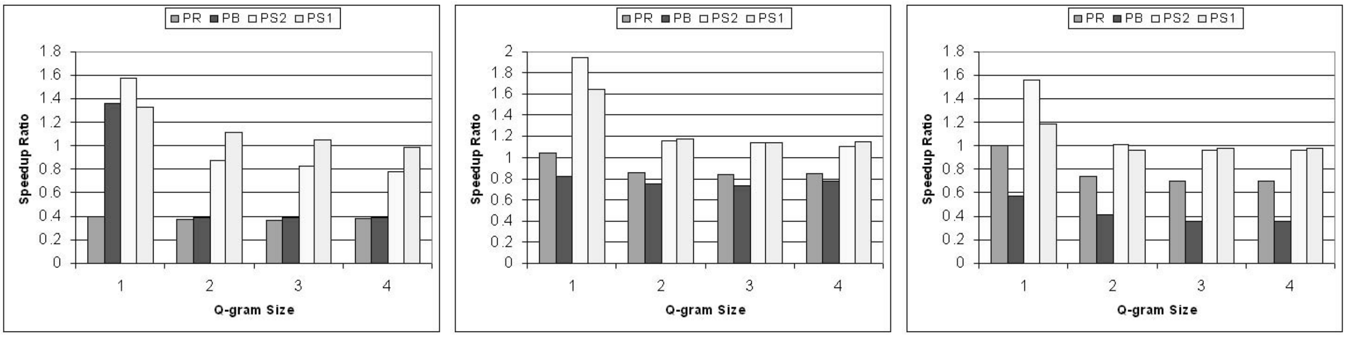
The results in Figure 8 seem to contradict the pruning power results in Figure 7, since the speedup ratios of PR and PB are less than those of PS2 and PS1. This is due to the additional index search time and the time for counting the number of matching Q-grams. Even though PR and PB can remove more false candidates, this does not compensate the cost of index traversal. This also explains why PR or PB perform worse than a sequential scan (speedup ratio is less than 1) in some cases shown in Figure 8. The speedup ratio results also show that PR and PB perform worse on data sets with shorter length, such as ASL data (Figure 8(a)), than they do on trajectories with longer length, such as Kungfu data (Figure 8(c)). The reason is that the time required to compute EDR of short trajectories is less than that of longer trajectories; as a consequence, the total time saved from computing EDR of short trajectories is

[3] This data set combines all the trajectories of ten word classes into one data set.



(a) ASL data (b) Slip data (c) Kungfu data

Figure 7: Pruning power comparisons of mean value Q-Grams on three data sets



(a) ASL data (b) Slip data (c) Kungfu data

Figure 8: Speedup ratio comparisons of mean value Q-Grams on three data sets

less than that of longer trajectories. We find that PS2 needs less time than PS1 with Q-grams of size 1, which reflects the fact that the total time spent on finding common Q-grams of two trajectories can be compensated by the time saved from removing more false candidates. However, as shown in Figure 8, when PS2 prunes little, such as Q-grams of size greater than 1, it performs worse than PS1 because the saved time can not cover the cost of finding common Q-grams on trajectories. From two test results, we conclude that PS2 on Q-gram of size 1 is the best method to remove false candidates with mean value Q-grams.

5.2 Efficiency of Pruning by Near Triangle Inequality

If trajectories in a database have the same size, the near triangle inequality can not remove false candidates. Kungfu and Slip data sets contain trajectories of the same length and are not used in this experiment. Instead, we generated two random walk data sets with different lengths (from 30 to 256), the lengths of one random walk data set follow uninform distribution (RandU) and the other one has normal distribution (RandN). There are 1,000 trajectories in each random walk data set. The pruning power and speedup ratio results for these and ASL data sets are shown in Table 3.

	ASL	RandN	RandU
Pruning Power	0.09	0.07	0.26
Speedup Ratio	1.10	1.07	1.31

Table 3: Test results of near triangle inequality

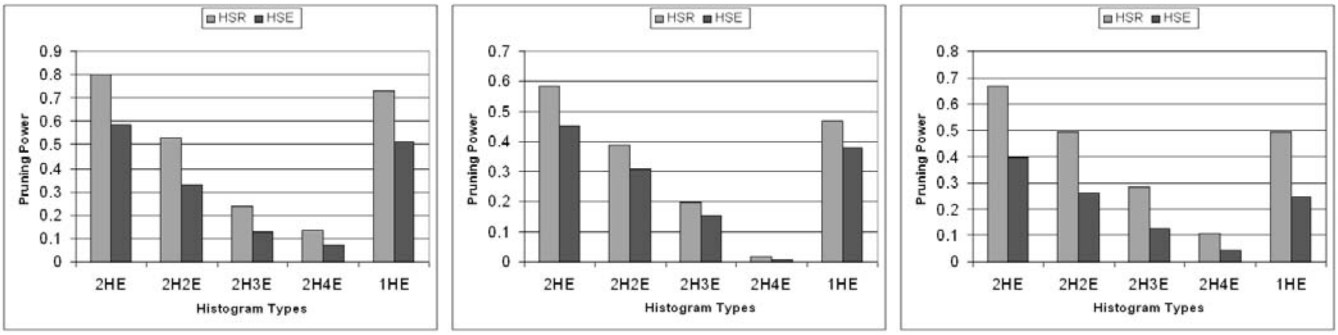
The results show that both the pruning power and the speedup ratio of near triangle inequality is pretty low compared to the results of mean value Q-grams. This is because the factor $|S|$ that we introduced in near triangle inequality is too big, which reduces its pruning power. Finding a smaller suitable value is left as future work. We also find that near triangle inequality works better on the data set whose lengths follow a uniform distribution (trajectory lengths of ASL data follow a near normal distribution), confirming our claim that it is more effective for trajectories of variable lengths.

5.3 Efficiency of Pruning by Histograms

In this experiment, we test the efficiency of different types of histograms. We test two scan methods, histogram sorted scan (HSR) and histogram sequential scan (HSE), with trajectory histograms of four different bin sizes (ϵ, 2ϵ, 3ϵ, 4ϵ) and one-dimensional data sequence histograms of bin size ϵ. The same three data sets used in Section 5.1 are tested here. The test results of pruning power and speedup ratio are shown in Figures 9 and 10, respectively. In both figures, $1HE$ stands for one-dimensional data sequence histograms with bin sizes ϵ. $2HE$, $2H2E$, $2H3E$, and $2H4E$ mean trajectory histograms with bin size ϵ, 2ϵ, 3ϵ, and 4ϵ, respectively.

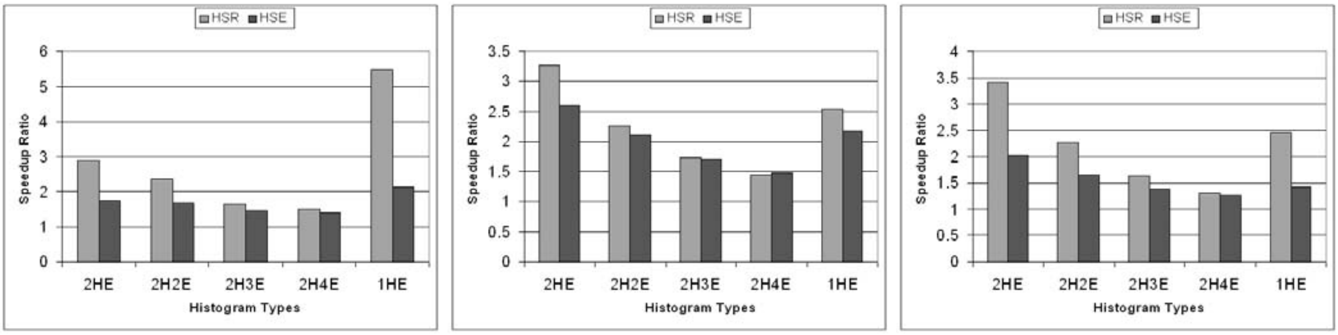
The pruning power results show that trajectory histograms with bin size ϵ has the highest pruning power on three data sets. A closer examination of Figure 9 shows that pruning power of one-dimensional data sequence histograms is better than that of trajectory histograms with larger bin sizes. Thus, in terms of efficiency of two methods used to reduce the number of histogram bins, creating one-dimensional sequence histograms with the same bin size ϵ is better than enlarging the bin size ϵ of trajectory histograms.

Even though HSR requires an additional sorting step, the results show that HSR beats HSE both in pruning power and speedup ratio tests, which indicates that it is worth sorting to increase the search efficiency. Since the computation cost of histogram distance is linear, nearly all the speedup ratio results match the pruning power test results, that is, the method that has higher pruning power needs less time to answer k-NN queries. However, the speedup ratio of one-dimensional data sequence histograms is very close to or even more than that of trajectory histograms with the same bin size (Figure 10(a)). This is because the pruning powers of two types of histograms are very similar and the time saved from computing distances using one-dimensional data sequence histograms is



(a) ASL data (b) Slip data (c) Kungfu data

Figure 9: Pruning power comparisons of histograms on three data sets



(a) ASL data (b) Slip data (c) Kungfu data

Figure 10: Speedup ratio comparisons of histograms on three data sets

more than the time that is spent on computing the extra number of EDR. Comparing the pruning power and speedup ratio results of mean value Q-grams and histograms (Figures 7 vs. 9 and Figures 8 vs. 10), we also find that histograms generally perform better than mean value Q-grams on removing false candidates.

5.4 Efficiency of Combined Methods

We test the combination of methods proposed in Section 4.4 on NHL data [5], a mixed data set [34] and a randomwalk trajectory data set [6, 19]. The NHL data consists of 5000 two dimensional trajectories of National Hockey League players and their trajectory lengths vary from 30 to 256. The mixed data set contains 32768 trajectories whose lengths vary from 60 to 2000. The randomwalk data set contains 100,000 two dimensional trajectories and their lengths vary from 30 to 1024. Based on the results of above experiments, for trajectory histograms, we select HRE rather than HSE, even though it requires an additional sorting step, since HRE outperforms HSE both in pruning power and speedup ratio. The PS2 method on mean value Q-grams is selected as the Q-grams filter. We do not select PR because it can only archive higher pruning power at a very expensive search cost. In the experiment, we test all six possible ordering combinations of methods. As we expect, the six combinations achieve the same pruning power on three data sets, which confirms the claim that the three pruning methods are independent of each other. With respect to speedup ratio, because of the differences in pruning power and computation cost of each pruning method, the application order affects the speedup ratio as shown in Figure 11 (The results of all the three data sets are listed in [7]). $2HPN$ means applying trajectory histograms with bin size ϵ pruning first, then Q-grams filtering, and at last, near triangle inequality pruning; the other symbols represent the rest of application orders. As shown in Figure 11, the example combination method listed in Figure 6 (applying histogram pruning first, then mean value Q-gram filtering, finally, near triangle inequality pruning) achieves the best performance, which is also the case in the other two data sets. The reason is that applying a pruning method with more pruning power and less expensive computation cost first will cause fewer false candidates left for subsequent pruning, resulting in a decrease in the time cost of subsequent pruning.

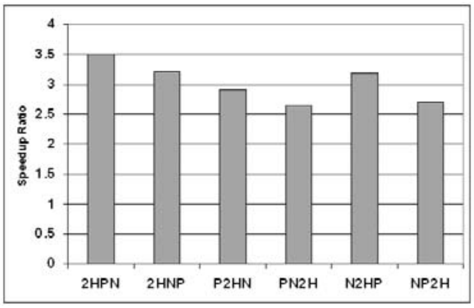
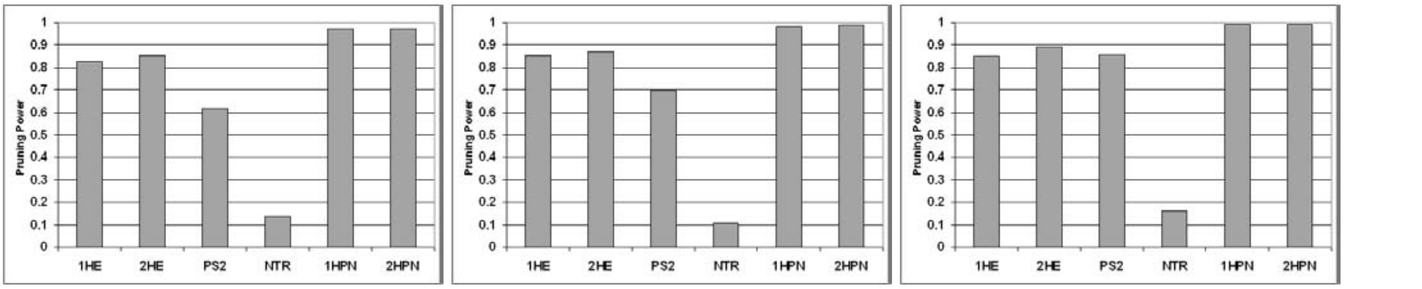


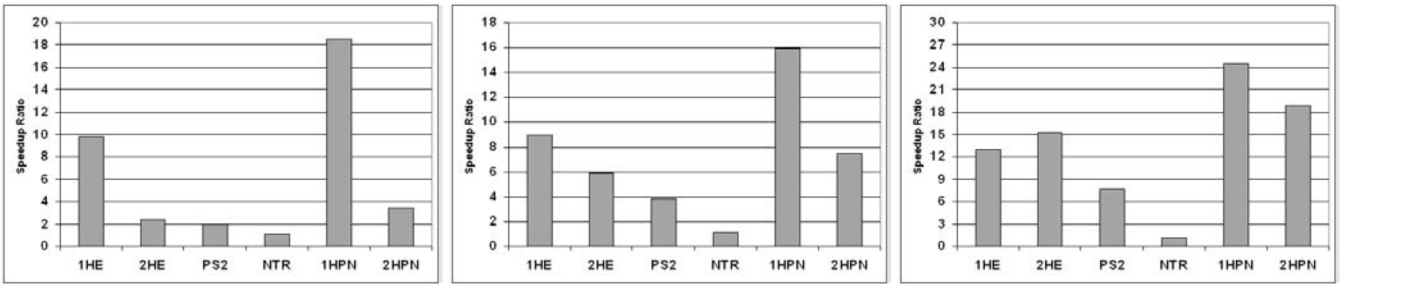
Figure 11: Speedup ratio comparison of different applying orders of three pruning methods on NHL data

Finally, we test the performance of different types of histogram pruning in the combination method. Two types of histograms are used in the combined method in Figure 6: trajectory histograms and one-dimensional data sequence histograms with the same bin size ϵ. Based on previous experiment results, we compare the combined method of applying HSR on histograms first, then, PS2 on mean value Q-grams, and finally, pruning by near triangle inequality (with 400 reference trajectories). The results are shown in Figures 12 and 13. In both figures, NTR stands for pruning by near triangle inequality, $1HPN$ stands for the combination with one-dimensional data sequence histograms with bin size ϵ, merge join on two dimensional Q-grams, and near triangle inequality.



(a) NHL data (b) Mixed data (c) Randomwalk data

Figure 12: Pruning power comparisons of combined methods on three large data sets



(a) NHL data (b) Mixed data (c) Randomwalk data

Figure 13: Speedup ratio comparisons of combined methods on three large data sets

The results show that combined methods using one-dimensional data sequence histograms achieve the best performance. Both the pruning power and the speedup ratio are increased. The speedup ratio is nearly twice of using histogram pruning only, five times that of mean value Q-grams only, and twenty times that of near triangle inequality. The combined method with trajectory histograms also beats the method using trajectory histograms only. However, because of the large number of bins of trajectory histograms, its performance improvement is diminished by the time spent on computing the histogram distances, especially for large databases.

6. CONCLUSIONS AND RELATED WORK

We argue that an accurate and robust similarity measure is needed for searching similar trajectories in the database, since existing ones do not handle real data with noise well. In this paper, we propose a new distance function, Edit distance on Real sequence (EDR) to measure the similarity between trajectories. EDR is more robust and more accurate than existing distance functions. We show that EDR has similar efficacy as DTW, ERP and LCSS over trajectories without noise, but more robust performance over trajectories with noise.

In order to improve the retrieval efficiency of EDR, we propose three pruning techniques and prove that they do not introduce false dismissals. We also propose different implementation methods of three pruning techniques and test their efficiency by extensive experimental studies. Most importantly, we show the three pruning methods can be combined to deliver superior retrieval efficiency.

Limited work has been done on multidimensional time-series data. Bozkaya et al. [4] present a modified version of LCSS to compute the distance between two sequences. In order to answer the similarity-based queries efficiently, an index scheme is designed based on the lengths of the sequences and relative distances between sequences. However, they focus on retrieving sequences of feature vectors extracted from image sequences and indexing is based on exact equality match on real values, which is not suitable for similarity search. Our work focuses on trajectory retrieval and EDR is defined based on range value match. Lee et al. [25] use the distance between minimum bounding rectangle to compute the distance between two multidimensional sequences. Even though they can achieve very high recall, the distance function can not avoid false dismissals. Our work guarantees that there are no false dismissals. Cai and Ng [5] propose an effective lower bound technique for indexing trajectories. However, Euclidean distances are used as the similarity measure [25, 39, 5], and, as argued earlier, this measure is not robust to noise or time shifting which often appear in trajectory data. Little and Gu [27] use path and speed curves to represent the motion trajectories and measure the distance between two trajectories using DTW. Vlachos et al. [35] also use DTW on rotation invariant representation of trajectories, sequences of angle and arc-length pairs. However, DTW requires continuity along the warping path, which makes it sensitive to noise and it is unable to find trajectories that have similar shapes but with dissimilar gaps in between. Chen and Ng [6] introduce a metric distance function, ERP, to measure the similarity between time series data. Like DTW, ERP can handle the time series data with local time shifts. However, because it takes the differences of real values as distance, ERP is also sensitive to noise. As shown in the experimental results over benchmark data, EDR distance function we proposed in this paper is robust to noise.

Vlachos et al. [36] use LCSS to compare two trajectories with the consideration of spatial space shifting. Compared with DTW and ERP, LCSS is robust to noise. However, LCSS allows gaps with various sizes to exist between similar shapes in the sequences, which cause it inaccurate. In our work, the spatial shifting of trajectories are handled by normalization. Compared to LCSS, EDR is not only robust to noise, it also assigns penalties according to the sizes of the gaps in between similar shapes, which makes it more accurate. A cluster-based indexing tree is proposed in [36] to improve the retrieval efficiency using LCSS. The performance of this indexing method depends on the clustering results. However, due to LCSS not following triangle inequity, it is hard to find good clusters and representing points in the data set [15]. Our three pruning techniques do not have this limitation.

Acknowledgements

Thanks to Dr. Raymond Ng, Dr. Edward Chang, and Dr. Ihab Ilyas for their valuable comments. This research is funded by Intelligent Robotics and Information Systems (IRIS), a Network of Center of Excellence of the Government of Canada.

7. REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *Proc. Conf. of Foundations of Data Organization and Algorithms*, 1993.

[2] S. A. Aghili, D. Agrawal, and A. El Abbadi. BFT: Bit Filtration Technique for Approximate String Join in Biological Databases. In *Proc. SPIRE*, 2003.

[3] A. Andoni, M. Deza, A. Gupta, P. Indyk, and S.Raskhodnikova. Lower bounds for embedding edit distance into normed spaces. In *Proc. Symp. on Discrete Algorithms*, 2003.

[4] T. Bozkaya, N. Yazdani, and Z. M. Ozsoyoglu. Matching and indexing sequences of different lengths. In *Proc. CIKM*, 1997.

[5] Y. Cai and R. Ng. Indexing spatio-temporal trajectoires with Chebyshev polynomials. In *Proc. SIGMOD*, 2004.

[6] L. Chen and R. Ng. On the marriage of edit distance and Lp norms. In *Proc. VLDB*, 2004.

[7] L. Chen Similarity-based Search Over Time Series and Trajectory Data. *Ph.D. thesis*, University of Waterloo, 2005, http://db.uwaterloo.ca/~l6chen/.

[8] S.-C. Chen and R. L. Kashyap. A spatio temporal semantic model for multimedia presentations and multimedia database systems. *TKDE*, 13(4), 2001.

[9] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proc. Symp. on Discrete Algorithms*, 2002.

[10] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (Almost) for free. In *Proc. VLDB* 2001.

[11] J. Gips, M. Betke, and P. Fleming. The camera mouse: Preliminary invertigation of automated visaul tracking for computer access. In *Proc. Conf. on Rehab. Eng. and Assis. Tech. Soc. of North America*, 2000.

[12] K-S. Goh, B. T. Li, and Ed. Chang. Dyndex: a dynamic and non-metric space indexer. In *Proc. SIGMM*, 2002.

[13] D.Q. Goldin and P.C. Kanellakis. On similarity queries for time series data: Constraint specification and implementation. In *Proc. Conf. on Principles and Pratice of Constraint Programming*, 1995.

[14] D. P. Huttenlocher, G. A. Klanderman, and W. A. Rucklidge. Comparing images using the hausdorff distance. *PAMI*, 15(9), 1993.

[15] D. W. Jacobs, D. Weinshall, and Y. Gdalyahu. Classification with nonmetric distances: Image retrieval and class representation. *PAMI*, 22(6), 2000.

[16] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[17] P. Jokinen and E. Ukkonen. Two algorithms for approximate string matching in static texts. In *Proc. Conf. of Mathematical Foudations of Computer Science*, 1991.

[18] T. Kahveci and A. Singh. Variable length queries for time series data. In *Proc. ICDE*, 2001.

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

[20] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. SIGMOD*, 2001.

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

[22] E. Keogh and M. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proc. SIGKDD*, 2000.

[23] F. Korn, H. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *Proc. SIGMOD*, 1997.

[24] K.P.Chan and A.W-C Fu. Efficient time series matching by wavelets. In *Proc. ICDE*, 1999.

[25] S.-L. Lee, S.-J. Chun, D.-H. Kim, J.-H. Lee, and C.-W. Chung. Similarity search for multidimensional data sequences. In *Proc. ICDE*, 2000.

[26] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8), 1966.

[27] J. L. Little and Z. Gu. Video retrieval by spatial and temporal sturcture of trajectories. In *Proc. Symp. on Storage and Retrieval for Image and Video Databases*, 2001.

[28] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches in query processing for moving object trajectories. In *Proc. VLDB*, 2000.

[29] C. A. Ratanamahatana and E. Keogh. Making time-series classification more accurate using learned constraints. In *Proc. SIAM Conf. on Data Mining*, 2004.

[30] V. Roth, J. Laub, J. Buhmann, and K.-R. Muller. Going metric: Denoising pairwise data. In *Proc. Conf. on Neural Info. Processing Systems*, 2002.

[31] E. Sutinen and J. Tarhio. Filtration with q-samples in approximate string matching. In *Proc. Symp. on Combinatorial Pattern Matching*, 1996.

[32] A. Tversky. Features of similarity. *Psychological Review*, 84, 1977.

[33] M. Vlachos. Personal communication. 2004.

[34] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In *Proc. SIGKDD*, 2003.

[35] M. Vlachos, D. Gunopulos, and G. Das. Rotation Invariant Distance Measures for Trajectories. In *Proc SIGKDD*, 2004.

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

[37] M. Vlachos, J. Lin, E. Keogh, and D. Gunopulos. A wavelet-based anytime algorithm for k-means clustering of time series. In *Proc. Wksp. on Clustering High Dim. Data and its Appl.*, 2003.

[38] R. Weber, H.-J Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. VLDB*, 1998.

[39] Y. Yanagisawa, J. Akahani, and T. Satoh. Shape-based similarity query for trajectory of mobile objects. In *Proc. Conf. on Mobile Data Management*, 2003.

[40] B-K Yi and C. Faloutsos. Fast time sequence indexing for arbitrary Lp norms. In *Proc. VLDB*, 2000.

[41] B-K Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proc. ICDE*, 1998.