

# Symbolic Representation and Retrieval of Moving Object Trajectories \*

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

Vincent Oria  
New Jersey Inst. of Technology  
Dept. of Computer Science  
Newark, New Jersey, USA  
{vincent.oria@njit.edu}

Technical Report CS-2003-30 Sept 2003



---

\*submitted to EDBT2004

## Abstract

Similarity-based retrieval of moving object trajectory is useful to many applications - GPS systems, sport and surveillance video analysis. However, due to sensor failures, errors in detection techniques, or different sampling rates, noises, local shifts and scales may appear in the trajectory records. Hence, it is difficult to design a robust and fast similarity measure for similarity-based retrieval in a large database. In this paper, *normalized edit distance* (NED) is proposed to measure the similarity between two trajectories. We evaluate the efficacy of NED and compare it with those of Euclidean distance, Dynamic Time Warping (DTW), and Longest Common Subsequences (LCSS), showing that NED is more robust and accurate for trajectories that contain noise and local time shifting. Furthermore, in order to improve the retrieval efficiency, we propose a novel representation of trajectories, called *movement pattern strings*, which convert the trajectories into a symbolic representation. Movement pattern strings encode both the movement direction and the movement distance information of the trajectories. The distances that are computed in a symbolic space are lower bounds of the distances of original trajectory data, which guarantees that no false dismissals will be introduced using movement pattern strings to retrieve trajectories. Finally, we define a *modified frequency distance* for frequency vectors that are obtained from movement pattern strings to reduce the dimensionality of movement pattern strings and computation cost of NED. The experimental results show that the cost of retrieving similar trajectories can be greatly reduced when the modified frequency distance is used as a filter.

## 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 have been developed based on the analysis of trajectories. For example, using a GPS 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, such as hockey, 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. All of these applications require the definition of an accurate and robust similarity measure to determine similarity among trajectories.

The trajectory of a moving object is defined as the successive positions of the moving object over a period of time. Therefore, trajectories can be considered as two ( $X - Y$  plane) or three ( $X - Y - Z$  plane) dimensional time series data. 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 [1, 13, 14, 18, 19, 23, 24, 30]. A question that can be easily raised is: “Can we apply these techniques for one dimensional time series data to trajectories?” The answer is unfortunately, negative; directly applying these techniques will not get satisfactory results. The reason is that trajectories of moving objects have their own characteristics, which will be briefly introduced in next section.

## 1.1 Characteristics of Trajectories

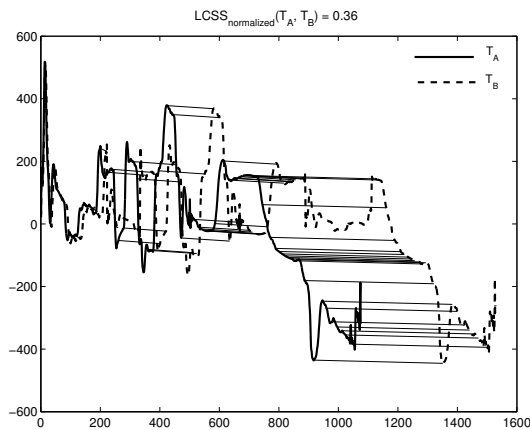
Compared to one dimensional time series data, trajectories of moving objects have the following differences:

- Trajectories are always two or three dimensional. Since each point of a trajectory is represented as a vector in two or three dimensions, dimensionality reduction techniques for one dimensional time series data, such as *Discrete Fourier Transform* (DFT) [1], *Discrete Wavelet Transform* (DWT) [19, 23], *Single Value Decomposition* (SVD) [13, 18] and *Piece-Wise Aggregate Approximation* (PAA) [14, 30], cannot be applied to trajectories. Naively treating each dimension of the moving object positions independently, the trajectories can be considered as two or three one-dimensional time series data. However applying dimensionality reduction techniques independently on each of the dimensions will lead to the loss of valuable information on the interdependency among the dimensions embedded in the positions of a trajectory.
- Trajectories may 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 the moving object from frame-to-frame in video data). Therefore, due to sensor failures or errors in detection techniques, many outliers may appear. The similarity measures for one dimensional time series data, such as Euclidean distance [1] and Dynamic Time Warping (DTW) [31] are very sensitive to noise and can not be applied to trajectories [26].
- Similar movement patterns may appear in different spatial regions of trajectories. Different sampling rates of tracking and recording devices combined with different speeds of the moving objects may introduce various local scaling and shifting factors into trajectories. Several techniques have been proposed to remove the shifting and scaling effects by introducing shifting and scaling functions [5, 6]. Unfortunately, these techniques work fine for global shifting and scaling but not for the local shifting and scaling in movement patterns that appear in the trajectories.

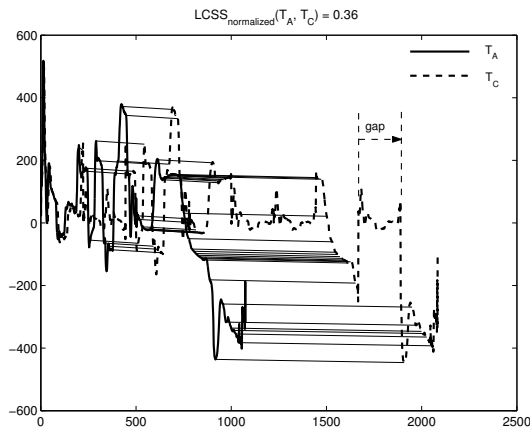
After reviewing the complex characteristics of trajectory data, a question comes to our mind is “can we find a suitable similarity measure which takes

these characteristics into consideration when we compare trajectories?” Furthermore, with the proposed similarity measure, “how can we improve the retrieval efficiency?” We will address these two questions in our paper.

## 1.2 Accurate and Robust Similarity Measures for Trajectories



(a)



(b)

Figure 1: A comparison of trajectories with the same normalized LCSS but different gap sizes

Recently, Longest Common Subsequence (LCSS) has been proposed to measure the similarity between trajectories [26]. Compared to DTW and Euclidean distance, LCSS allows the matching sequence to stretch and some elements to be unmatched, which makes it robust to noise [26]. However, LCSS has difficulties in differentiating the sequences that have the longest common subsequences of the same length but different sizes of gaps in between. Figure 1 shows an example of this case<sup>1</sup>, where the normalized LCSS

<sup>1</sup>The original trajectory data are two dimensional. For clarity, in the figures, we only

score [26] between trajectories  $T_A$  and  $T_B$  (Figure 1(a)) is the same as that between  $T_A$  and  $T_C$  (Figure 1(b)). However, by comparing the three trajectories (the horizontal grey lines are used to show the common subsequences between two trajectories), it is quite clear that  $T_A$  is more similar to  $T_B$  than to  $T_C$ .

In this paper, we define a distance measure called *Normalized Edit Distance* (NED) to measure the similarity between two trajectories. NED is based on Edit Distance (ED) [20], which is widely used in bio-informatics and speech recognition to measure the similarity between two strings. In contrast to LCSS, NED considers the gaps in between subsequences as well as the subsequences themselves. For example, for the trajectories shown in Figure 1, the value of NED between  $T_A$  and  $T_B$  is 0.7 and 0.78 for  $T_A$  and  $T_C$  (the detailed definition of NED is given in Section 2), which conform to the perceptual similarity that  $T_A$  is similar to  $T_B$  than to  $T_C$ .

However, the space and time cost of computing NED is very high, increasing the retrieval cost as a consequence. Since edit distance is originally defined for strings, it seems possible to convert the real-valued trajectory data into strings and utilize the well defined algorithms and embedded distance functions of strings to improve the retrieval efficiency. Thus, we propose a novel trajectory representation, called *movement pattern strings* (MPS). A MPS is derived from a trajectory by quantizing the (movement direction, distance ratio) space into a set of distinct equal-sized subregions and representing each subregion by a symbol. Most importantly, the NED that is computed from two MPSs establishes the lower bound of the NED of two original sequences of movement direction and distance pairs, which guarantees that no dismissals will be introduced using the symbolic representation. Furthermore, we define a *modified frequency distance* (MFD) between two *frequency vectors* (FV) of movement pattern strings to reduce the cost of CPU time on computing NED of two movement sequences. A normalized MFD (NMFD) between two FVs is also the lower bound of NED between two trajectories. Therefore, we can directly use FV as a filter to remove the false candidates during the retrieval.

### 1.3 Our Main Contributions

The main contributions of our paper are the following:

1. We define a distance measure, NED, based on ED, to measure the similarity between two trajectories. NED is more robust than DTW and Euclidean distance and more accurate than LCSS.
2. We develop a transformation scheme to convert a trajectory into a symbolic representation, called movement pattern strings, and prove that the NED that is computed over a symbolic space is the lower

---

show one dimension.

bound of the NED of real trajectory data. A movement pattern string is transformed from a sequence of (movement direction, distance ratio) pairs, which makes this representation invariant to spatial rotation, scaling and shifting.

3. We propose a modified frequency distance, MFD, as a distance measure of frequency vectors that are obtained from movement pattern strings. We also prove that the normalized MFD of frequency vectors is the lower bound of NED of real movement sequences, which reduces the computation cost of NED.

## 1.4 Organization of the Paper

The rest of the paper is arranged as follows: Section 2 introduces our distance measure NED, and also briefly presents some definitions. In Section 3, we present our symbolic representation of trajectories and prove the lower bound property. Section 4 introduces the modified frequency distances followed by experimental results comparing NED with three other similarity measures in terms of their efficacy and robustness in classification and clustering trajectories and the retrieval efficiency of our symbolic representation and frequency vectors in Section 5. Section 6 provides a in depth comparison with the related work. We conclude in Section 7 and indicate some further work.

## 2 Normalized Edit Distance

In this section, we first give the formal definitions of trajectories, sequences of (movement direction, distance ratio) pairs, we then introduce our similarity model, the Normalized Edit Distance between two trajectories.

### 2.1 Preliminaries

In the following, we assume that objects are points that move in a two-dimensional space ( $x - y$  plane) and that time is discrete.

**Definition 1** Given a moving object  $A$ , its trajectory *trajectory*,  $T_A$ , is defined as a sequence of pairs with each pair showing the position of object  $A$  in the  $x - y$  plane:

$$T_A = [(x_{a,1}, y_{a,1}), \dots, (x_{a,n}, y_{a,n})]$$

Here  $n$ , the number of positions in  $T_A$ , is defined as the *length* of  $T_A$ .

We refer to  $T_A$  as the *raw representation of the trajectory*, since this is the most likely data format that we can get from tracing sensor or extraction techniques. However, directly using this raw representation to compare

trajectories will not find the trajectories with similar movement but different spatial rotation, shifting, or scaling factors. For example, in Figure 2, three trajectories are shown:  $T_B$  can be derived from  $T_A$  by scaling its  $x$  and  $y$  positions by a factor of 2, while  $T_C$  is translated from  $T_B$  by shifting its  $x$  and  $y$  positions by 1. Three trajectories have similar movement patterns, however, comparing their raw representations, (which are:  $T_A = [(3.5, 4.5), (1.5, 2.5), (2.5, 3.5), (2, 3), (3, 4)]$ ,  $T_B = [(7, 9), (3, 5), (5, 7), (4, 6), (6, 8)]$ , and  $T_C = [(8, 10), (4, 6), (6, 8), (5, 7), (7, 9)]$ .) can not lead to a conclusion that they are similar unless scaling or shifting factors are introduced in the similarity measures. However, this will increase the cost for computing the similarity measure (e.g. in [26], by only introducing the shifting factors in LCSS, the computation cost is increased by  $O(n)$ , where  $n$  is the length of the trajectory).

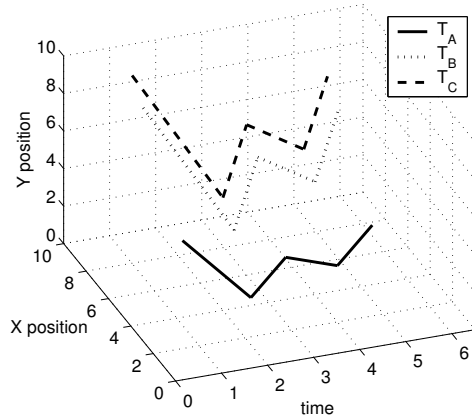


Figure 2: The trajectories with different scaling and shifting factors share similar movement pattern

Thus, instead of directly using raw representation of trajectories, we represent them by means of a sequence of (movement direction, distance ratio) pairs. This representation is not affected by rotation, shifting or scaling [22].

**Definition 2** Given a moving object  $A$  and its trajectory  $T_A$  of length  $n$  ( $n > 1$ ), the sequence of (movement direction, distance ratio) pair  $M_A$  is defined as a sequence of pairs:

$$M_A = [(\theta_{a,1}, \sigma_{a,1}), \dots, (\theta_{a,m}, \sigma_{a,m})]$$

where  $m = n - 1$  and  $n$  is the length of  $T_A$ . The movement direction  $\theta_{a,i}$  is

$$\theta_{a,i} = \begin{cases} \arctan\left(\frac{y_{a,(i+1)} - y_{a,i}}{x_{a,(i+1)} - x_{a,i}}\right) & x_{a,(i+1)} - x_{a,i} \geq 0, \\ \arctan\left(\frac{y_{a,(i+1)} - y_{a,i}}{x_{a,(i+1)} - x_{a,i}}\right) - \pi & y_{a,(i+1)} - y_{a,i} \leq 0 \text{ and} \\ & x_{a,(i+1)} - x_{a,i} < 0, \\ \arctan\left(\frac{y_{a,(i+1)} - y_{a,i}}{x_{a,(i+1)} - x_{a,i}}\right) + \pi & y_{a,(i+1)} - y_{a,i} > 0 \text{ and} \\ & x_{a,(i+1)} - x_{a,i} < 0. \end{cases}$$

The *movement distance ratio*  $\sigma_{a,i}$  is

$$\sigma_{a,i} = \begin{cases} \frac{\sqrt{(y_{a,(i+1)} - y_{a,i})^2 + (x_{a,(i+1)} - x_{a,i})^2}}{TD(T_A)} & TD(T_A) \neq 0 \\ 0 & 0 \end{cases}$$

where the *total movement distance* of  $T_A$  is

$$TD(T_A) = \sum_{1 \leq j \leq n-1} \sqrt{(y_{a,(j+1)} - y_{a,j})^2 + (x_{a,(j+1)} - x_{a,j})^2}$$

Base on this definition, we know that  $\theta_{a,i}$  ranges from  $-\pi$  to  $\pi$  and  $\sigma_{a,i}$  ranges from 0 to 1. We use  $M_A(n)$  to denote the sequence  $[(\theta_{a,1}, \sigma_{a,1}), \dots, (\theta_{a,n}, \sigma_{a,n})]$  where  $n$  is the length of the sequence. All three example trajectories in Figure 2 have the same sequence of (movement direction, distance ration) pairs, which conforms to the fact that they have similar movement patterns.

In the rest of this paper, we use the terms “movement sequence” or “sequence” interchangeably to refer to the sequence of (movement direction, distance ratio) pairs unless specified otherwise.

## 2.2 Normalized Edit Distance Definition

We introduce a similarity measure, called *normalized edit distance* (NED), which considers gaps within sequences. It does this by counting similar subsequences and assigning penalties to the gaps in between these subsequences. Since movement sequences are not strings but numerical value pair sequences, we first have to define the cases which element pairs of different movement sequences *match*. We introduce two thresholds  $\epsilon_{dir}$  and  $\epsilon_{dis}$  to that end:  $\epsilon_{dir}$  is used to determine whether two movement directions match and  $\epsilon_{dis}$  is for determining the similarity of two movement distance ratios.

**Definition 3** Two (movement direction, distance ratio) pairs  $(\theta_{a,i}, \sigma_{a,i})$  and  $(\theta_{b,j}, \sigma_{b,j})$  are said to *match* if and only if  $|\theta_{a,i} - \theta_{b,j}| \leq \epsilon_{dir}$  and  $|\sigma_{a,i} - \sigma_{b,j}| \leq \epsilon_{dis}$ . This is specified as predicate  $match((\theta_{a,i}, \sigma_{a,i}), (\theta_{b,j}, \sigma_{b,j}))$ .

Based on the definition of *match*, we define *edit distance* between two movement sequences.

**Definition 4** Given two moving objects  $A$  and  $B$  and their movement sequences  $M_A$  of length  $n$  and  $M_B$  of length  $m$ , respectively, the *edit distance* (ED) between  $M_A$  and  $M_B$  is the number of insert, delete, or replace operations that is needed to change  $M_A$  into  $M_B$ .  $ED(M_A(n), M_B(m))$  can be computed as follows:

$$\left\{ \begin{array}{ll} n & \text{if } m = 0; \\ m & \text{if } n = 0; \\ ED(M_A(n-1), M_B(m-1)) & \text{if } match((\theta_{a,i}, \sigma_{a,i}), (\theta_{b,j}, \sigma_{b,j})) \\ \min[ED(M_A(n-1), M_B(m-1)) + 1, & \\ ED(M_A(n-1), M_B(m)) + 1, & \\ ED(M_A(n), M_B(m-1)) + 1] & \\ \text{otherwise} & \end{array} \right.$$

In Definition 4, we assume that the cost of replace, insert, or delete operations is only 1, which corresponds to the original definition of edit distance [20]. The value of ED depends on the length of compared sequences, which is a dependency that we wish to remove. Therefore, we define *normalized edit distance*, which is the distance measure that we use in this paper.

**Definition 5** The *normalized edit distance* (NED) between two movement sequences  $M_A$  and  $M_B$  is defined as:

$$NED(M_A(n), M_B(m)) = \begin{cases} 0 & \text{if } m = n = 0; \\ \frac{ED(M_A(n), M_B(m))}{\max(m, n)} & \text{otherwise} \end{cases}$$

since the maximum number of operations to transfer  $M_A$  to  $M_B$  is the maximum length of the two sequences.

DTW also assigns penalties to gaps in between subsequences (requires all the elements in the matching sequences to match and sums the differences), however, in case of NED, the penalties are only related to the length of the gap (each penalty scores 1), while the penalties computed from DTW are related to both the values of the matched elements and the length of the gap (each penalty scores the difference between two matched elements). This causes DTW to be very sensitive to noise data. For example, given three one dimensional sequences (for simplicity, we use one dimensional data),  $s_1 = [0.1, 0.2, 0.3]$ ,  $s_2 = [0.1, 0.2, 2.3, 0.3]$  and  $s_3 = [0.1, 0.2, 0.4, 0.4, 0.4, 0.3]$ , the perceptual similarity is that  $s_1$  is more similar to  $s_2$  than it is to  $s_3$  (the value 2.3 is a possible noise point). The DTW distance between  $s_1$  and  $s_2$  is 4, and 0.03 for  $s_1$  and  $s_3$ , implying that  $s_1$  is more similar to  $s_3$  than to  $s_2$ , which contradicts the perceptual similarity. The distances measured by NED between these pairs are  $\frac{1}{7}$  for  $s_1, s_2$  and  $\frac{1}{3}$  for  $s_1, s_3$ , which is closer to the perceptual similarity. Incidentally, the LCSS distances between two example

pairs of sequences are same, which is better than DTW, but less accurate than NED in reflecting the perceptual similarity. In Section 5, we experimentally show that, in terms of efficacy, NED is much better than LCSS, Euclidean distance, and DTW.

Although NED is more accurate, its computation still costs  $O(n \times m)$  time and space [9] with dynamic programming, where  $n$  and  $m$  are the lengths of the two sequences  $M_A$  and  $M_B$ , respectively. The time cost becomes critical when the database of sequences is large. Moreover, NED is not a metric because it does not follow the triangle inequality. For example, given three movement sequences:  $M_A = [(0, 0.4), (\frac{\pi}{4}, 0.6)]$ ,  $M_B = [(\frac{\pi}{4}, 0.4), (\frac{\pi}{2}, 0.6)]$ , and  $M_C = [(\frac{\pi}{2}, 0.4), (\frac{3\pi}{4}, 0.6)]$  and  $\epsilon_{dir} = \frac{\pi}{4}$ ,  $\epsilon_{dis} = 0.125$ ,  $NED(M_A, M_C) > NED(M_A, M_B) + NED(M_B, M_C)$ . As a consequence, traditional access methods which assume that triangle inequality holds for the distance measure can not be used to index NED without introducing false dismissals. Therefore, in next two sections, we will discuss how to reduce the computation cost of NED to improve the retrieval efficiency and without losing the accuracy.

### 3 Symbolic Representation of Trajectories

Since edit distance was originally defined on strings for which many algorithms, data structures, and embedded edit distance functions have been developed, it is intuitive to think about the possibility of converting real valued movement sequences into symbolic representation and applying the string dimensionality reduction techniques to reduce the computation and retrieval cost. Therefore, we propose a symbolic representation of movement sequences. The basic idea is to quantize the (movement direction, distance ratio) space and represent each subregion by a distinct symbol.

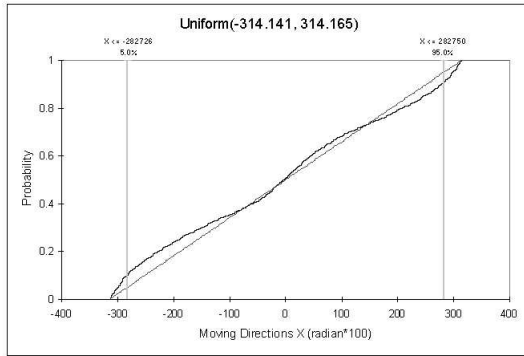


Figure 3: Probability distribution of movement directions of hokey players

As defined in Section 2, the values of (movement direction, distance ratio) of an object range from  $-\pi$  to  $\pi$  and 0 to 1, respectively. We investigated the movement direction distribution of three trajectory data sets, which are:

hockey players’ trajectories that are extracted from National Hockey League (NHL) videos, the “Cameramouse”, and Australian Sign Language data sets (the last two data sets are explained in detail in Section 5). We found that movement directions of all three data sets are uniformly distributed. Figure 3 shows the values of the probability density function of movement directions of NHL data, which shows that the movement direction of a hockey player at each sampled position follows an uniform distribution. Based on these observations, we divide the (movement direction, distance ratio) space into equal sized subregions.

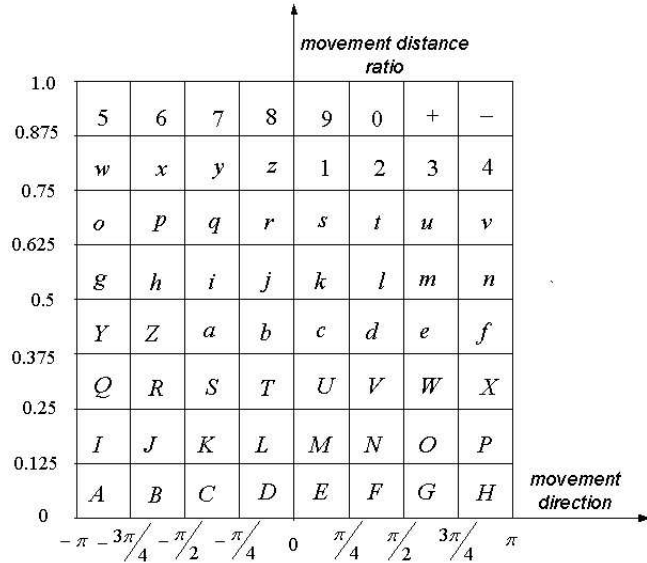


Figure 4: An example of (movement direction, distance ratio) quantization map

Given  $\epsilon_{dir}$  and  $\epsilon_{dis}$ <sup>2</sup>, we equally divide the two dimensional (movement direction, distance ratio) space into  $(2\pi/\epsilon_{dir}) \times (1.0/\epsilon_{dis})$  subregions and assign each subregion a distinct symbol. The whole set of symbols makes up the *movement pattern alphabet*, which we denote as  $\mathcal{A} = A_1, A_2, \dots, A_s$ , where the size of the movement pattern alphabet is  $s = (2\pi/\epsilon_{dir}) \times (1.0/\epsilon_{dis})$ . Once the two threshold values  $\epsilon_{dir}$  and  $\epsilon_{dis}$  are given for the trajectory data, the size of movement pattern alphabet is fixed.  $A_i$  is a distinct symbol that represents a subregion  $SB_i$  of size  $\epsilon_{dir} \times \epsilon_{dis}$  ( $1 \leq i \leq s$ ). Each subregion  $SB_i$  is represented by two (movement direction, distance ratio) pairs:  $(\theta_{bl,i}, \sigma_{bl,i})$  and  $(\theta_{ur,i}, \sigma_{ur,i})$ , which are the bottom left and upper right coordinates of  $SB_i$ .

A *quantization map* (QM) that contains all the subregions and associated

<sup>2</sup>The values of  $\epsilon_{dir}$  and  $\epsilon_{dis}$  are application dependent. They must be given beforehand to determine whether two elements match when we use NED or LCSS as a similarity measure to query the trajectory data set.

---

**Algorithm 1** The algorithm for mapping a (movement direction, distance ratio) pair to a symbol

---

**Require:** /\*input:  $(\theta, \sigma)$ , quantization map  $QM$  \*/

**Ensure:** /\*output: mapped symbol\*/

```

1: if  $\sigma == 1.0$  /* boundary case, the movement distance ratio is 1.0*/
   then
2:   for each subregion  $SB_i$  do
3:     if  $\theta_{bl,i} < \theta \leq \theta_{ur,i}$  and  $\sigma_{bl,i} \leq \sigma \leq \sigma_{ur,i}$  then
4:       return  $A_i$ 
5:     end if
6:   end for
7: else
8:   for each subregion  $SB_i$  do
9:     if  $\theta_{bl,i} < \theta \leq \theta_{ur,i}$  and  $\sigma_{bl,i} \leq \sigma < \sigma_{ur,i}$  then
10:      return  $A_i$ 
11:    end if
12:   end for
13: end if

```

---

symbols is stored as a lookup table, and is used to convert (movement direction, distance ratio) pairs into symbols and to obtain the neighbors for a given symbol. The second function is used to compute the distance in the converted symbolic space (Definitions 7 and 8). Figure 4<sup>3</sup> gives an example quantization map, which divides (movement direction, distance ratio) space into 64 subregions, each subregion corresponding to a movement symbol.

Once we quantize the (movement direction, distance ratio) space into subregions and derive the movement alphabet  $\mathcal{A}$ , we use Algorithm 1 to map a (movement direction, distance ratio) pair  $(\theta, \sigma)$  into a symbol. For example, according to the quantization map in Figure 4,  $(\frac{\pi}{3}, 0.16)$ ,  $(\frac{\pi}{4}, 0.1)$ , and  $(-\frac{\pi}{4}, 1.0)$  are mapped into symbols ‘N’, ‘E’, and ‘7’, respectively.

**Definition 6** Given a movement sequence  $M_A = [(\theta_{a,1}, \sigma_{a,1}), \dots, (\theta_{a,n}, \sigma_{a,n})]$  of length  $n$  and movement pattern alphabet  $\mathcal{A}$ , a *movement pattern string* (MPS) is defined as a sequence of symbols:  $S_{a,1}S_{a,2} \dots S_{a,n}$ , where each symbol  $S_{a,i}$  ( $1 \leq i \leq n$ ) is mapped from the movement direction and distance pair  $(\theta_{a,i}, \sigma_{a,i})$  according to  $\mathcal{A}$ .

MPS retains the order of movement sequences by arranging the corresponding symbols from left to right. We use  $MPS_A(n)$  to denote the string  $S_{a,1}S_{a,2} \dots S_{a,n}$ . Figure 5 gives an example of converting a movement sequence  $M_A = [(\frac{\pi}{3}, 0, 16), (-\frac{\pi}{4}, 0, 16), (\frac{\pi}{6}, 0.33), (-\frac{\pi}{3}, 0.33)]$  of  $T_A$  to the movement pattern string “NKUS” using movement pattern alphabet as given in Figure 4.

---

<sup>3</sup>In Figure 4,  $\epsilon_{dir} = \pi/4$  and  $\epsilon_{dis} = 0.125$ .

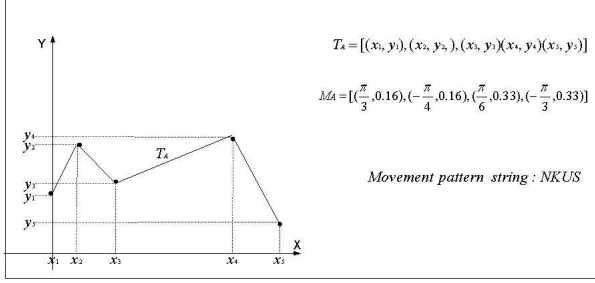


Figure 5: An example of converting a trajectory to a movement pattern string

After converting movement sequences to MPSs, we can directly compute the NED between two MPSs. Using the standard edit distance [20] does not provide correctness. since the NED that is computed in this way is not the lower bound of the NED of the original movement sequences. This is because the (movement direction, distance ratio) pairs that are located near the boundary of quantization subregions may be assigned different symbols and require a *replace* operation that is not needed in the original sequence comparison. For example, given two movement sequences  $M_A = [(0, 0.4), (\frac{\pi}{4}, 0.6)]$ ,  $M_B = [(\frac{\pi}{4}, 0.4), (\frac{\pi}{2}, 0.6)]$  and  $\epsilon_{dir} = \frac{\pi}{4}$ ,  $\epsilon_{dis} = 0.125$ , the corresponding MPSs are:  $MPS_A = "bk"$  and  $MPS_B = "cl"$ . NED between original movement sequences  $M_A$  and  $M_B$  is 0, whereas the NED between  $MPS_A$  and  $MPS_B$  that is computed based on the standard edit distance [20] is 1, which is not the lower bound of 0. As a consequence, trajectory retrieval by using this NED will introduce false dismissals, which is not allowed in applications that require high accuracy. Therefore, we define a NED for MPSs, which is the lower bound of the NED between original movement sequences. We need to define the cases that two symbols *approximately match*.

**Definition 7** Two symbols  $A_i$  and  $A_j$  are said to *approximately match* (denoted by predicate  $ap\_match(A_i, A_j)$ ), if and only if  $A_i == A_j$  or  $A_i$  is the neighbor of  $A_j$ .  $A_i$  is a neighbor of  $A_j$  if the subregions that they represent in the quantization map are directly connected. For example, according to the quantization map in Figure 4, the neighbors of T are K, L, M, S, U, a, b, c and the neighbors of A are H, B, P, I, J (note that movement directions are in polar space).

**Definition 8** The *ED* between two movement pattern strings  $MPS_A$  and  $MPS_B$  of length  $n$  and  $m$ , respectively, is defined as the number of insert, delete, or replace operations that is needed to change  $MPS_A$  into  $MPS_B$ :

(denoted as  $ED(MPS_A(n), MPS_B(m))$ )

$$\left\{ \begin{array}{ll} n & \text{if } m = 0; \\ m & \text{if } n = 0; \\ ED(MPS_A(n-1), MPS_B(m-1)) & \\ & \text{if } ap\_match(S_{a,n}, S_{b,m}) \\ \min[ED(MPS_A(n-1), MPS_B(m-1)) + 1, & \\ ED(MPS_A(n-1), MPS_B(m)) + 1, & \\ ED(MPS_A(n), MPS_B(m-1)) + 1] & \\ & \text{otherwise} \end{array} \right.$$

The computation of NED on MPS is the same as that given in Definition 5. Now we need to prove that this NED of MPS is the lower bound to NED of original movement sequences.

**Lemma 1** Given two movement sequences  $M_A$  and  $M_B$  and their corresponding movement pattern strings  $MPS_A$  and  $MPS_B$ , let  $(\theta_{a,i}, \sigma_{a,i})$  and  $(\theta_{b,j}, \sigma_{b,j})$  be two (movement direction, distance ratio) pairs of  $M_A$  and  $M_B$ , respectively, and  $S_{a,i}$  and  $S_{b,j}$  be their corresponding symbols from  $MPS_A$  and  $MPS_B$ . If  $match((\theta_{a,i}, \sigma_{a,i}), (\theta_{b,j}, \sigma_{b,j})) = true$ , then  $ap\_match(S_{a,i}, S_{b,j}) = true$  and if  $ap\_match(S_{a,i}, S_{b,j}) = false$ , then  $match((\theta_{a,i}, \sigma_{a,i}), (\theta_{b,j}, \sigma_{b,j})) = false$ .

**Proof:** During the process of mapping (movement direction, distance ratio) pairs into symbols, there are only three cases that could happen:

1.  $S_{a,i}$  and  $S_{b,j}$  are the same symbol;
2.  $S_{a,i}$  and  $S_{b,j}$  are neighbors;
3.  $S_{a,i}$  and  $S_{b,j}$  are not neighbors;

If  $match((\theta_{a,i}, \sigma_{a,i}), (\theta_{b,j}, \sigma_{b,j})) = true$ ,  $S_{a,i}$  and  $S_{b,j}$  must be the same symbol or neighbors. Otherwise there are at least one subregion gap between them in the quantization map. From the definition of movement pattern alphabet, the size of each subregion is  $\epsilon_{dir} * \epsilon_{dis}$ . Thus, we have:  $|\theta_{a,i} - \theta_{b,j}| > \epsilon_{dir}$  and  $|\sigma_{a,i} - \sigma_{b,j}| > \epsilon_{dis}$ , which contradict to  $match((\theta_{a,i}, \sigma_{a,i}), (\theta_{b,j}, \sigma_{b,j})) = true$ . Therefore, according to Definition 7,  $ap\_match(S_{a,i}, S_{b,j}) = true$ .

If  $ap\_match(S_{a,i}, S_{b,j}) = false$ , then  $S_{a,i}$  and  $S_{b,j}$  are not the same symbol or neighbors. As above, we have:  $|\theta_{a,i} - \theta_{b,j}| > \epsilon_{dir}$  and  $|\sigma_{a,i} - \sigma_{b,j}| > \epsilon_{dis}$ . According to Definition 7,  $match((\theta_{a,i}, \sigma_{a,i}), (\theta_{b,j}, \sigma_{b,j})) = false$ .  $\square$

**Theorem 1** Given two movement sequences  $M_A$  and  $M_B$ , and their corresponding movement pattern strings  $MPS_A$  and  $MPS_B$ ,  $NED(MPS_A, MPS_B) \leq NED(M_A, M_B)$ .

**Proof:** Since the conversion from movement sequence to MPS will not change the length, we only need to show:  $ED(MPS_A, MPS_B) \leq ED(M_A, M_B)$ .

Let  $\#ap\_match$  denote the number of pairs of symbols in  $MPS_A$  and  $MPS_B$  that approximately match (i.e.,  $\#ap\_match = \text{count}(ap\_match(S_{a,i}, S_{b,j}) = true), S_{a,i} \in MPS_A, S_{b,j} \in MPS_B)$ .

Let  $\#match$  denote the number of (movement direction, movement distance ratio) pairs in  $M_A$  and  $M_B$  that match (i.e.,  $\#match = \text{count}(match((\theta_{a,i}, \sigma_{a,i}), (\theta_{b,j}, \sigma_{b,j})) = true), (\theta_{a,i}, \sigma_{a,i}) \in M_A, (\theta_{b,j}, \sigma_{b,j}) \in M_B)$ .

Define  $\#ap\_match$  and  $\#match$  as converses of these (i.e., count where predicates are false). Then, according to Lemma 1,  $\#ap\_match \geq \#match$  and  $\#ap\_match \leq \#match$ . Therefore the cost (in terms of number of edit operations) of changing  $MPS_A$  to  $MPS_B$  is less than that of changing  $M_A$  to  $M_B$ .  $\square$

Converting movement sequences into MPS has several advantages:

1. MPSs require much less storage space compared to the original movement sequences. For example, if each symbol in a MPS is stored as a character, a movement pattern string only needs  $\frac{1\text{byte}}{8\text{bytes}} = 12.5\%$  of the storage space needed to store the original movement sequence (assuming a character type needs 1 byte and floating type (real value) needs 4 bytes). As a consequence, the retrieval cost of a movement pattern string should be less than its corresponding movement sequence.
2. According to Theorem 1, the NED of the two movement patterns strings is the lower bound of the NED of this original sequences, which guarantees that no false dismissals will be introduced when we use movement patterns strings as a filter in answering queries such as  $k$ -nearest neighbors. As mentioned in Section 1, two factors affect the efficiency of retrieval: I/O cost and CPU cost. Using MPS as filter is based on the assumption that the retrieval cost may be reduced due to the smaller size of MPS compared to movement sequences. Algorithm 2 describes how MPS is used to answer 1-nearest neighbor query (1-NN).
3. Compared to movement sequences, movement pattern strings are one dimensional strings, so the dimensionality reduction techniques for strings can be applied to them. This is addressed in detail in the next section.

## 4 Modified Frequency Distances

Even though we reduce the storage requirements by converting movement sequences into movement pattern strings, the cost of computing the NED between two MPSs is still  $O(n * m)$ , since the length of a movement sequence and that of its corresponding movement pattern string are the same. Therefore, directly using movement pattern strings as filter in retrieval trajectories

---

**Algorithm 2** The algorithm for answering 1-NN query using MPS

---

**Require:** /\*input: a query movement sequence  $M_Q$  and its corresponding movement pattern string  $MPS_Q$ , quantization map  $QM^*$ \*/

**Ensure:** /\*output: the movement sequence that is the nearest neighbor of  $M_Q$  \*/

```

1: smallest_distance = maxDistance
2: for each movement sequence  $M_i$  and its corresponding movement pattern
   string  $MPS_i$  in the database do
3:   compute  $NED(MPS_i, MPS_Q)$ 
4:   if  $NED(MPS_i, MPS_Q) < smallest\_distance$  then
5:     compute  $NED(M_i, M_Q)$ ;
6:     if  $NED(M_i, M_Q) < smallest\_distance$  then
7:        $smallest\_distance = NED(M_i, M_Q)$ ;
8:        $result = M_i$ 
9:     end if
10:  end if
11: end for
12: return result

```

---

will not reduce the computation cost. In [12], Kahveci and Singh propose a transformation of strings into a multidimensional integer space by mapping them to their *frequency vectors* (FV). They prove that the *frequency distance* (FD) between the FVs of two strings is the lower bound of the actual edit distance. The definitions of the frequency vector, the frequency distance and the their theorem are as following [12]:

**Definition 9** Given a string  $S$  from the alphabet  $\mathcal{A} = A_1, A_2, \dots, A_s$ , let  $n_i$  be the number of occurrences of character  $A_i$  in  $S$  for  $1 \leq i \leq s$ . The *frequency vector*  $f(s)$  of  $s$  is:  $f(s) = [n_1, n_2, \dots, n_s]$ .

**Definition 10** Let  $u$  and  $v$  be integer points in  $s$  dimensional space, The *frequency distance*  $FD(u, v)$  between  $u$  and  $v$  is defined as the minimum number of steps that is required to go from  $u$  to  $v$  (or equivalently from  $v$  to  $u$ ) by moving to a neighbor point at each step.  $u$  and  $v$  are neighbors if one of them can be obtained from the other using a single edit operation.<sup>4</sup>

**Theorem 2** Let  $S_A$  and  $S_B$  be two strings from the alphabet  $\mathcal{A} = A_1, A_2, \dots, A_s$ , then  $FD(f(S_A), f(S_B)) \leq ED'(S_A, S_B)$ .

Their proof uses standard edit distance [20] between strings (Let us denoted by  $ED'$  for simplicity of expositor). In contrast to ED of two MPS

---

<sup>4</sup>In [12], the frequency distance is defined on first one or two wavelet coefficients and the dimensionality of corresponding transformed integer space is  $s$  or  $2s$ . We select the frequency distance that is defined on first one wavelet coefficient (frequency vector) because of its low dimensionality.

that is based on the concept of approximately match (Definition 7),  $ED'$  is computed based on the equality of two symbols. Therefore their results can not be directly applied to MPS. In order to reduce the dimensionality of MPS, we define a *modified frequency distance* (MFD) as an extension of frequency distance.

**Definition 11** Let  $u$  and  $v$  be integer points in  $s$ - dimensional space. The *modified frequency distance*  $MFD(u, v)$  between  $u$  and  $v$  is defined as the minimum number of steps required to go from  $u$  to  $v$  (or equivalently from  $v$  to  $u$ ) by moving to a *next-to-neighbor* point at each step.  $u$  and  $v$  are next-to-neighbors if one of them can be obtained from the other using two single edit operations (one to become a neighbor and a second one for a match). Similar to NED’s definition (Definition 5), we define the *normalized MFD* (NMFD) by dividing MFD by the sum of lengths of two compared sequences.

Compared to FD, MFD takes boundary cases into consideration, which corresponds to the approximately match concept defined in Definition 7.

**Theorem 3** If  $MPS_A$  and  $MPS_B$  are two strings from the movement pattern alphabet  $\mathcal{A} = A_1, A_2, \dots, A_s$ , then  $MFD(f(MPS_A), f(MPS_B)) \leq ED(MPS_A, MPS_B)$ , where  $f(S_i)$  is the FV of string  $S_i$  as defined in Definition 10.

**Proof** A straightforward extension of proof of Theorem 2 in [12].

Based on Theorem 2 and Theorem 3, we have:

**Corollary 1** Given two movement sequences  $M_A$  and  $M_B$  of length  $n$  and  $m$  and their corresponding movement pattern strings  $MPS_A$  and  $MPS_B$ ,  $NMFD(f(MPS_A), f(MPS_B)) \leq NED(M_A, M_B)$ .

Corollary 1 is exciting, because it proves that the NMFD between two FVs of  $MPS_A$  and  $MPS_B$  is the lower bound of the NED between the corresponding movement sequences. Therefore, in order to answer queries such as  $k$ -nearest neighbor queries, instead of directly computing the NED of movement sequences, we can compute the NMFD to prune out false candidates from the database. Most importantly, the computation cost of NMFD is linear! Algorithm 3 computes NMFD between two FVs. The nested for loops in the algorithm may suggest that the computation time of NMFD is non-linear. However, as the number of neighbors of each integer point in the frequency space is limited (at most 8), the computation time of Algorithm 3 is still linear. Our experimental results also confirm this. Using FV as filter further reduces the retrieval cost since the dimensionality of FV is usually smaller compared to that of movement sequences. Algorithm 4 answers 1-NN query using FV.

---

**Algorithm 3** The algorithm for computing the NMFD

---

**Require:** /\*input:  $s$  dimensional integer points  $u$  and  $v$ , lengths of compared sequences  $len1$  and  $len2$ , quantization map  $QM^*$ \*/**Ensure:** /\*output: the value of NMFD \*/

```
1: posDist=0, negDist=0
2: for each  $i=1$  to  $s$  do
3:    $u_i = u_i - v_i$ ;
4: end for
5: for  $i=1$  to  $s$  do
6:   if  $u_i \neq 0$  then
7:     for each neighbor  $u_j$  of  $u_i$  do
8:       if  $(u_i * u_j < 0)$  then
9:         if  $abs(u_i) > abs(u_j)$  then
10:           $u_i = u_i + u_j, u_j = 0$ 
11:        else
12:           $u_j = u_j + u_i, u_i = 0$ 
13:        end if
14:      end if
15:    end for
16:  end if
17: end for
18: for  $i=1$  to  $s$  do
19:   if  $u_i > 0$  then
20:     $posDist+ = u_i$ 
21:  else
22:     $negDist+ = (-u_i)$ 
23:  end if
24: end for
25: if  $posDis > negDist$  then
26:  return  $posDist/(len1 + len2)$ 
27: else
28:  return  $negDist/(len2 + len2)$ 
29: end if
```

---

## 5 Experiments

In this section, we present the experimental results on testing the efficacy and robustness of NED and retrieval efficiency using MPS and frequency vectors. All experiments were run on a Sun-Blade-1000 workstation with 1G memory under Solaris 2.8.

---

**Algorithm 4** The algorithm for answering 1-NN query using FV

---

**Require:** /\*input: a query movement sequence  $M_Q$  and its corresponding feature vector  $u_Q$ , quantization map  $QM^*$ \*/

**Ensure:** /\*output: the movement sequence that is the nearest neighbor of  $M_Q$  \*/

```
1: smallest_distance = maxDistance
2: for each movement sequence  $M_i$  and its corresponding feature vector  $u_i$ 
   in the database do
3:   compute  $NMFD(u_i, u_Q)$ 
4:   if  $NMFD(u_i, u_Q) < smallest\_distance$  then
5:     compute  $NED(M_i, M_Q)$ ;
6:     if  $NED(M_i, M_Q) < smallest\_distance$  then
7:       smallest_distance =  $NED(M_i, M_Q)$ ;
8:       result =  $M_i$ 
9:     end if
10:  end if
11: end for
12: return result
```

---

## 5.1 Efficacy and Robustness of NED

We conduct experiments to test the efficacy of NED compared to DTW [22], LCSS [26] and Euclidean Distance [29]. According to a recent survey on time series data [16], the efficacy of a similarity measure can be evaluated by clustering and classification results on labelled data sets. Therefore, in our first experiment, we perform hierarchy clustering using four similarity measures on two labelled data sets.

In order to test the robustness of the four similarity measures, two labelled trajectory data sets are generated from the “Cameramouse” [8] and the Australian Sign Language (ASL)<sup>5</sup> data sets by adding interpolated Gaussian noise (about 10-15% of the length of trajectories) and time warping [27]. The “Cameramouse” data set contains 15 trajectories of 5 words (3 for each word) obtained by tracking the tip of finger when people “write” various words. The ASL data set from UCI data archive consists of samples of Australian Sign Language signs. 95 signs are collected for 5 different writers. We extract 5 recording for each of following 10 words: “Norway”, “cold”, “crazy”, “eat”, “forget”, “happy”, “innocent”, “later”, “lose”, and “spend”, which are used in [17, 26]. The raw trajectories in both data sets are converted into movement direction and distance sequences. For each data set, we take all possible pairs of words and use the “complete linkage” hierarchy clustering algorithm [11], which produces the best clustering results [26] to partition them into two clusters. We draw the dendrogram of each clustered result to see whether it correctly partitions the trajectories.

---

<sup>5</sup><http://kdd.ics.uci.edu>

Since  $\epsilon_{dir}$  and  $\epsilon_{dis}$  are data and application dependent, we run the experiments with different values of  $\epsilon_{dir}$  and  $\epsilon_{dis}$  for two data sets. We find that for ASL data, we get best results for LCSS and NED when  $\epsilon_{dir} = 0.167\pi$  and  $\epsilon_{dis} = 0.1 * \sigma_{max}$ , where  $\sigma_{max}$  is the maximum value of movement distance ratio in the data set, which can be obtained when we convert raw trajectories to movement sequences. For ‘‘Cameramouse’’ data set,  $\epsilon_{dir} = 0.1\pi$  and  $\epsilon_{dis} = 0.1 * \sigma_{max}$ . Since the Euclidean distance requires compared sequences with the same length, we use the same strategy that is used in [26], where the shorter of the two trajectories is slid along the longer one and the minimum distance is recorded. We report the best result of each measure in Table 1.

| Correct clustering results     | Euclidean distance | DTW | LCSS | NED | NED (mps) |
|--------------------------------|--------------------|-----|------|-----|-----------|
| Cameramouse (total 10 correct) | 2                  | 7   | 8    | 10  | 10        |
| ASL (total 45 correct)         | 4                  | 8   | 15   | 22  | 21        |

Table 1: Clustering results of four similarity measures

As we expect, the results show that NED outperforms the other three measures on both data sets. Since Euclidean distance is very sensitive to local time warping and noise, it has the poorest performance among four measures. DTW performs better than Euclidean distance, because it allows for the local time warping. However, it requires all the data in the sequence to match-even the noise data-leading to a number of wrong clusterings. LCSS is better than DTW, since it only counts the longest common subsequence, which removes the noise effect. However, because LCSS does not assign penalties to the gaps in between similar subsequences, trajectories from different words to be clustered together. NED overcomes the shortcomings of LCSS. The NED(mps) column in Table 1 shows the clustering results using MPS that are obtained from the corresponding movement sequences. Due to the lower bound property of NED on MPS, clustering on it achieves nearly the same number of correct results as that of clustering on original movement sequences.

From Table 1, we also find that, compared to LCSS, the improvement obtained by NED on ‘‘Cameramouse’’ data is less than that on ASL data (8-10 v.s. 15-22). This is due to the different lengths of trajectories. The longer the length of the trajectory, the less effect that gap penalties have in computing the similarity. The average length of ‘‘cameramouse’’ and ASL data are 1103 and 65, respectively. Regardless of the similarity measure we used, the number of correctly found clusters in ASL data set is less than half of the total clusterings (45). This is because the data length of ASL data is

relatively small and some trajectories from different words are quite similar to each other.

In our second experiment, we carry out simple classification using 1-Nearest Neighbor with four similarity measures and test the classification results using the “leave one out” verification mechanism [16]. The error rate is the ratio of number of wrong classifications to the total number of trajectories in the data set. We used the same program as in Experiment 1 to generate more sample data based on ASL and “Cameramouse”. The data set that is generated from “Cameramouse” contains 5 classes and 30 examples of each class, and the one from ASL data contains 10 classes where each class has 50 examples. The values of  $\epsilon_{dir}$  and  $\epsilon_{dis}$  are the same as in the first experiment. Table 2 reports the results. NED again performs best among the four similarity measures. The classification results on MPS using NED (NED(mps) column) are also better than the other three.

| Error Rate (%) | Euclidean distance | DTW | LCSS | NED | NED (mps) |
|----------------|--------------------|-----|------|-----|-----------|
| Cameramouse    | 57                 | 34  | 25   | 14  | 18        |
| ASL            | 67                 | 42  | 24   | 18  | 21        |

Table 2: Error rates of classification results of four similarity measures

To summarize, compared to Euclidean distance, DTW, and LCSS, NED is more effective and robustness for measuring the similarity between two trajectories that contain noise and local time shifts. The gap penalties of NED have less effect on the longer trajectories. MPS can effectively capture movement pattern of real movement sequences.

## 5.2 Efficiency of MPS and FV in Retrieval

As shown in Algorithms 2 and 4, both MPS and FV can be used as filters to remove false candidates before computing NED on real movement sequences. The aim of filtering is to remove as many false candidates as possible to reduce the retrieval and computation cost. Therefore, in our third experiment, we use *pruning power* as a measure of the filtering effect of MPS and FV. The pruning power is measured by  $P$ , which is defined as the fraction of the data set that must be examined before we can guarantee that the answer to 1-NN is found [15].

$$P = \frac{\text{number of movement sequence that required to compute NED}}{\text{total number of movement sequences in the data set}}$$

Besides ASL and “Cameramouse” data sets that are used in Experiment 2, we add three more data sets. The first two are synthetic trajectories that are generated by the program used in [29]. The first synthetic data set

(SYN1) contains 500 trajectories and the length of each trajectory is 128. The second synthetic data set (SYN2) has 500 trajectories and the length of each trajectory is 512. The program generates the trajectories by simulating the change in moving speeds and directions when people walk freely on a plane. The frequencies of the changes in these values are altered randomly in order to generate complex moving shapes. The trajectories are limited to an area of fixed size (500x500) and the movement directions follow a uniform distribution. The third data set contains 589 trajectories of hockey players, which were extracted from NHL videos. The length of each trajectory is 256.

For each data set, we randomly select 10% of the sequences as query data. A randomly selected movement sequence and its MPS and FV are used to conduct a 1-NN query using Algorithms 2 and 4. Results are averaged over 50 queries. For SYN1, SYN2, NHL and “Cameramouse” data sets, we use  $\epsilon_{dir} = 0.1\pi$  and  $\epsilon_{dis} = 0.1 * \sigma_{max}$ , and  $\epsilon_{dir} = 0.167\pi$  and  $\epsilon_{dis} = 0.1 * \sigma_{max}$  for ASL data set. Figure 6 reports the pruning power of MPS and FV on different data sets. The  $x$ -axis is used to denote different data sets that are arranged from left to right according to their average data length.

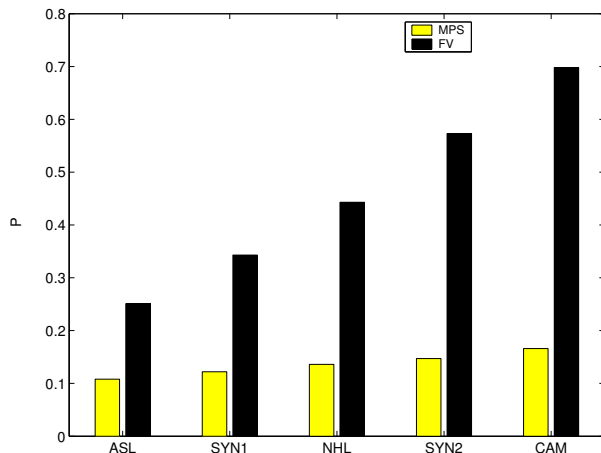


Figure 6: The pruning power of MPS vs. FV using data sets with different lengths

As shown in Figure 6, the pruning power of MPS is much higher than FV for all data sets, because MPS keeps the sequence order information of the original movement sequence while FV does not (only count the frequency). This also explains why the pruning power of FV drops much faster than MPS with increasing trajectory length. MPS has quite stable pruning power over trajectory length, because it maintains in the strings, the order of the corresponding (movement direction, distance ratio) pairs, and its ability to remove a lot of false candidates due to its consideration of neighbors of each symbol.

Based on pruning power, it seems that MPS is better than FV; however, the computation cost of NED on MPS is the same as that on movement

sequences, which is quadratic, while the computation of NMFD is linear! Therefore, we conduct a fourth experiment to check the retrieval efficiency of MPS and FV in terms of total time that is spent on answering 1-NN queries. Here the total time refers to the time that is spent on data retrieval and the time that is spent on computing the distance measures. Because the real data set we obtained is quite small, we use the same program as in Experiment 3 to generate 5 synthetic data sets with trajectory lengths ranging from 64 to 1024. Each data set contains 10,000 trajectories. As in experiment 3, we randomly select 10% of the sequences as query data and conduct 1-NN queries using MPS and FV (mps-scan and fv-scan), respectively. We also run the queries using *linear-scan*, which sequentially scans the movement sequences and computes NED. All the results are averaged over 50 queries. The same values are used for  $\epsilon_{dir}$  and  $\epsilon_{dis}$  as those for synthetic data in Experiment 3. Figure 7 shows the total time of different methods on different data sets.

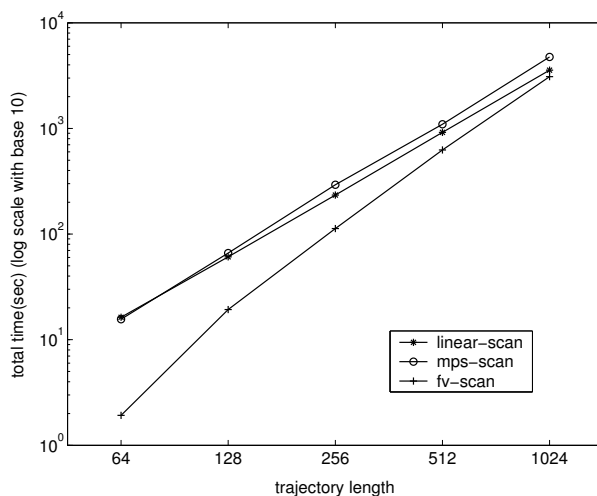


Figure 7: Comparison of total time of three methods using synthetic data sets with different lengths

From Figure 7, we can see that mps-scan is the worst among the three! This is quite reasonable, because NED is not a metric; thus, the mps-scan has to scan all the MPSs in the databases and compute the NED between them in addition to computing the NED for the candidate movement sequences, since the cost of computing NED on MPS and NED on movement sequences are the same (even though in terms of comparing each element data of the sequences, MPS only requires one and a movement sequence requires two, with respect to the total computation cost, this one comparison saving can be ignored), the total time that is spent by mps-scan is more than that of linear-scan. Even though fv-scan also has to scan all the FVs and compute NMFD between them, the computation cost of NMFD is only linear which leads fv-scan to save significant CPU time. This is reflected very well in Figure 7. Especially when the trajectory length is shorter, the higher pruning power

of FV results in bigger improvements over linear-scan. We observe the same phenomena as in Experiment 3: when the trajectory length becomes longer, the improvement brought by fv-scan degrades quickly.

In the last experiment, we test the scalability of fv-scan with different sizes of data sets. We run 1-NN queries using fv-scan and linear-scan on 5 synthetic data sets with sizes ranging from 1,000 to 100,000. The trajectory lengths in all data sets are the same, which is 256. Since mps-scan always performs worse than linear-scan, we do not include it in this experiment. We use the same set up for  $\epsilon_{dir}$  and  $\epsilon_{dis}$  as in Experiment 4. The results are shown in Figure 8.

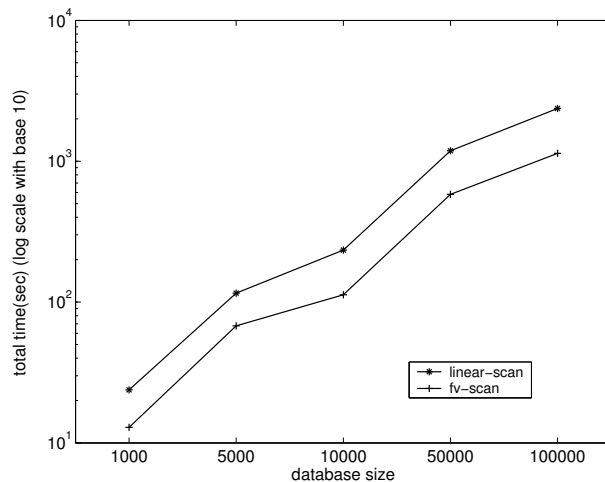


Figure 8: Comparison of total time of fv-scan and linear-scan using synthetic data sets with different sizes

In Figure 8, we can see that fv-scan scales smoothly with the increasing database size and it always takes nearly half of the total time of linear-scan (note that the y-axis scale is logarithmic).

Based on the experiments on retrieval efficiency of MPS and FV, we draw the following conclusions:

1. In terms of pruning power, MPS is more efficient than FV to remove false candidates, since MPS keeps the relative temporal order of elements in the movement sequence.
2. In terms of total retrieval efficiency, FV is much better than MPS due to the linearity of the computation cost of FV as opposed to quadratic cost for MPS.
3. Using FV as a filter can save nearly 2-10 (trajectory length ranges from 60-600) times of the retrieval cost compared to that of linear scan without the filter.

4. For trajectory data sets with longer length (e.g. the length is greater than 1,000), the pruning power of FV decreases. One possible way to address this issue is to segment the movement sequences of the data set into disjoint subsequences that can be handled by the FV filter. The query sequence is also segmented in the same way. We search the data set with each subsequence using FV as a filter and merge the results from each subsequence.
5. The pruning power of FV scales well with the increasing of database size.

## 6 Related Work

Very limited work has been done on multidimensional time-series data. Bozkaya et al. [3] present a modified version of longest common subsequences to compute the distance between two sequences. In order to answer the similarity-based queries efficiently, an index scheme was designed based on the lengths of the sequences and relative distances between sequences. However, they focused on retrieving sequences of feature vectors extracted from image sequences. Lee et al. [25] used the distance of minimum bounding rectangle to compute the distance between two multidimensional sequences. Even though they could achieve very high recall, the distance measure could not avoid false dismissals. Euclidean distances were used as the similarity metric in [25, 29], but, as argued earlier, this metric is not robust to noise or time shifting which often appear in trajectory data. Chen and Chang [4] used wavelet transform to decompose raw object trajectories (position sequences) into components at different scale. Global motion was described by coarsest scale components and finer components were used to partition the trajectory into sub-trajectories. Each sub-trajectory was characterized by a set of spatial and temporal attributes. However, the matching precision of their approach highly depends on the segmentation results.

Recently, Little and Gu [22] used the path and speed curves to represent the motion trajectories and measured the distance between two trajectories using DTW. However, DTW requires continuity along the wrapping path, which makes it sensitive to noise data and it is unable to find trajectories with similar shapes but with dissimilar gaps in between. Vlachos et al. [26] used LCSS to compare two trajectories. Compared with DTW, LCSS allows gaps to exist between similar shapes in the sequences. The similarity measure that we propose takes the longest common subsequences, gap penalties and compared sequence lengths into consideration.

Several approaches have been proposed to represent one dimensional time series data in symbolic form. Agrawal et al. [2] proposed *SDL*, which was a language for describing and retrieving the “shape” of one dimensional time series. The “shape” was defined based on the difference of every two con-

secutive values, which was quantized and represented by a distinct symbol. Huang and Yu [10] proposed IMPACT algorithm to transform time series data into symbol strings using change ratios between consecutive values. Lin et al. [21] proposed a symbolic representation of one dimensional time series data by first transforming it into piecewise aggregate approximation. They defined a new distance function between the symbolic representation which is the lower bound of the Euclidean distance between original time series data. A few approaches have been proposed in computer vision [7, 28] to represent trajectories by chain code. However, they only encoded the movement direction into the string, and discarded the information on movement distances.

Compared to the previous work on symbolic representation, our approach focuses on two dimensional trajectory data. MPSs are obtained from sequences of (movement direction, distance ratio), which encode both movement direction and distance into strings and are invariant to spatial scaling, rotation and translation. Most importantly, we prove that the NED computed on MPS is the lower bound of the NED that is computed on original movement sequences, which guarantees that no false dismissals will be introduced during the retrieval. Furthermore, we define NMFD between two frequency vectors and use frequency vectors as filters to save the cost of CPU time on computing NED.

## 7 Conclusions and Future Work

In this paper, we argue that an accurate and robust similarity measure is needed for searching similar trajectories in the database. Towards this end, we define a new similarity measure, called normalized edit distance (NED) to measure the similarity between trajectories of moving objects. Then, we propose a novel symbolic representation of object trajectories in order to reduce the computing cost of NED. Finally, we define the modified frequency distance (MFD) between two frequency vectors and prove that normalized MFD (NMFD) is the lower bound of NED between original sequences. This results means that, during the similarity-based retrieval, we can use the frequency vector as a filter to prune out the false candidates before we compute the NED between movement sequences. Our experimental results confirm that NED is a suitable and superior similarity measure for trajectory data and feature vector with NMFD can effectively reduce the false candidates in trajectory retrieval.

Future work includes the following problems:

1. Finding an embedding method, which keeps both the lower bound property and the temporal order of elements in the strings.
2. Defining a new metric distance function in the embedded space.

3. Developing an indexing structure for the new metric distance.
4. Investigating the mechanisms to handle sub-trajectory matching efficiently.

**Acknowledgements:** Thanks to Michalis Vlachos, Yutaka Yanagisawa, and Eamonn Keogh for providing their source codes or data sets. This research is funded by Intelligent Robotics and Information Systems (IRIS), a Network of Center of Excellence of the Government of Canada.

## References

- [1] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *Proc. 4th Int. Conf. of Foundations of Data Organization and Algorithms*, pages 69–84, 1993.
- [2] R. Agrawal, G. Psaila, E. L. Wimmers, and M. Zaït. Querying shapes of histories. In *Proc. 21th Int. Conf. on Very Large Data Bases*, pages 502–514, 1995.
- [3] T. Bozkaya, N. Yazdani, and Z. M. Ozsoyoglu. Matching and indexing sequences of different lengths. In *Proc. 6th Int. Conf. on Information and Knowledge Management*, pages 128–135, 1997.
- [4] W. Chen and S-F. Chang. Motion trajectory matching of video objects. In *Proc. 12th Int. Symp. on Storage and Retrieval for Image and Video Databases (SPIE)*, pages 544–553, 2000.
- [5] K. K-W Chu and M. H. Wong. Fast time-series searching with scaling and shifting. In *Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems*, pages 237–248, 1999.
- [6] G. Das, D. Gunopulos, and H. Mannila. Finding similar time series. In *Proc. 1st European Symp. on Principles of Data Mining and Knowledge Discovery*, pages 88–100, 1997.
- [7] N. Dimitrova and F. Golshani. Motion recovery for video content classification. *ACM Transactions on Information Systems*, 13(4):408–439, 1995.
- [8] J. Gips, M. Betke, and P. Fleming. The camera mouse: Preliminary investigation of automated visual tracking for computer access. In *In Proc. Conf. on Rehabilitation Engineering and Assistive Technology Society of North America*, pages 98–100, 2000.
- [9] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

- [10] Y. Huang and P. S. Yu. Adaptive query processing for time-series data. In *Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 282–286, 1999.
- [11] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.
- [12] T. Kahveci and A. Singh. Variable length queries for time series data. In *Proc. 17th Int. Conf. on Data Engineering*, pages 273–282, 2001.
- [13] K. V. Ravi Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 166–176, 1998.
- [14] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3):263–286, 2000.
- [15] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 151–162, 2001.
- [16] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. In *Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 102–111, 2002.
- [17] E. Keogh and M. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 285–289, 2000.
- [18] F. Korn, H. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 289–300, 1997.
- [19] K.P.Chan and A.W-C Fu. Efficient time series matching by wavelets. In *Proc. 15th Int. Conf. on Data Engineering*, pages 126–133, 1999.
- [20] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8):707–710, 1966.
- [21] J. Lin, E. Keogh, S. Lonardi, and P. Patel. Finding motifs in time series. In *Proc. 2nd Int. Workshop Temporal Data Mining*, pages 370–377, 2002.
- [22] J. L. Little and Z. Gu. Video retrieval by spatial and temporal structure of trajectories. In *Proc. 13th Int. Symp. on Storage and Retrieval for Image and Video Databases (SPIE)*, pages 544–553, 2001.

- [23] I. Popivanov and R. J. Miller. Similarity search over time series data using wavelets. In *Proc. 17th Int. Conf. on Data Engineering*, pages 212–221, 2001.
- [24] D. Rafiei and A. O. Mendelzon. Efficient retrieval of similar time sequences using DFT. In *Proc. 9th Int. Conf. of Foundations of Data Organization and Algorithms*, 1998.
- [25] S.-J. Chun S.-L. Lee, D.-H. Kim, J.-H. Lee, and C.-W. Chung. Similarity search for multidimensional data sequences. In *Proc. 16th Int. Conf. on Data Engineering*, pages 599–608, 2000.
- [26] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Proc. 18th Int. Conf. on Data Engineering*, pages 673 – 684, 2002.
- [27] M. Vlachos, J. Lin, E. Keogh, and D. Gunopulos. A wavelet-based anytime algorithm for k-means clustering of time series. In *Proc. Workshop on Clustering High Dimensionality Data and Its Applications*, 2003.
- [28] T. T. Y. Wai and A. L. P. Chen. Retrieving video data via motion tracks of content symbols. In *Proc. 6th Int. Conf. on Information and Knowledge Management*, pages 105–112, 1997.
- [29] Y. Yanagisawa, J. Akahani, and T. Satoh. Shape-based similarity query for trajectory of mobile objects. In *Proc. 4th Int. Conf. on Mobile Data Management*, pages 63–77, 2003.
- [30] B-K Yi and C. Faloutsos. Fast time sequence indexing for arbitrary Lp norms. In *Proc. 26th Int. Conf. on Very Large Data Bases*, pages 385–394, 2000.
- [31] B-K Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proc. 14th Int. Conf. on Data Engineering*, pages 23–27, 1998.