Shapes Based Trajectory Queries for Moving Objects *

Bin Lin and Jianwen Su
Dept. of Computer Science, University of California, Santa Barbara
Santa Barbara, CA, USA
linbin@cs.ucsb.edu, su@cs.ucsb.edu

ABSTRACT

An interesting issue in moving objects databases is to find similar trajectories of moving objects. Previous work on this topic focuses on movement patterns (trajectories with time dimension) of moving objects, rather than spatial shapes (trajectories without time dimension) of their trajectories. In this paper we propose a simple and effective way to compare spatial shapes of moving object trajectories. We introduce a new distance function based on "one way distance" (OWD). Algorithms for evaluating OWD in both continuous (piece wise linear) and discrete (grid representation) cases are developed. An index structure for OWD in grid representation, which guarantees no false dismissals, is also given to improve the efficiency of similarity search. Empirical studies show that OWD out-performs existent methods not only in precision, but also in efficiency. And the results of OWD in continuous case can be approximated by discrete case efficiently.

Categories and Subject Descriptors

H.3.3 [**Database Management**]: Information Search and Retrieval.

General Terms

Algorithms, Performance

Keywords

Moving object trajectories, similarity search, spatial shape, one way distance

1. INTRODUCTION

Advances in wireless communications and ubiquitous computing technologies provide significant stimuli for location dependent applications. The available hardware for both collecting and storing data makes it more tempting than ever to obtain and manage various types of data for applications. Among these data of general interest are location information of "moving objects". Indeed, global positioning systems (GPS) are now widely adopted in variety of applications. Navigation and positioning are becoming more important and even critical in many cases. One interesting type of applications requires finding "similar" trajectories of moving objects. For instance, in many sports such as football and tennis, it is very useful for sports researchers to figure out the movement patterns of top players by finding similar trajectories of objects' (players, balls) motions. By analyzing similar trajectories of animals, it is possible to determine migration patterns of them. In a city traffic monitoring system, it is helpful to locate popular routes by comparing similarity between vehicles' trajectories. This paper aims at similarity search for trajectories.

Similarity search is not a new topic and has been investigated in various context, e.g., motion tracking in videos [10, 8], time series analysis [6, 2, 15, 14, 7, 3, 12], and recently, trajectories, [16, 14, 9, 18, 17, 4, 11]. Partly due to the difficulty in formalizing "similarity" and partly due to the diversity of applications, known results aren't satisfactory in many application contexts.

Consider an imaginary traffic control application where commuters are tracked and their daily trajectories are stored. The trajectories of a commuter can be aggregated over a specific time interval (e.g. a month) to find its typical shape. An interesting problem is to analyze the aggregated trajectories to investigate the feasibility of a new bus route. The goal is to find how much benefit commuters can get. In this case, the similarity of the planned bus route and a commuter trajectory concerns more about the shared segments or location proximity, rather than completely matching the actual timed location sequences. In other words, the speed and direction information of trajectories are not critical, but the "spatial shape" is. For example, in Figure 1, trajectory B is more similar to A than C, while C is more similar to D than B. Analogously if the aggregated commuter trajectories are stored and compared, the similarity will also focus on the spatial closeness. In fact, the time ordering of an aggregated location sequence does not have much semantic meanings in this context.

In general, travel or traffic routes concern mostly with the starting and ending locations. In many applications, similarity of routes focus primarily on the closeness of the physical positions of the routes, while detailed timing and speed information may not be as important. This motivates the need for a new similarity notion based on closeness of their spatial shapes. Compared with movement pattern similarity, spatial shape similarity has the following characteristics:

- Time is not sensitive, and thus trajectories are represented without time information.
- Trajectories can be of different length.
- Similar trajectories must be spatially close to each other.
- When performing similarity comparison, the mapping between trajectories may not be continuous and monotonic.

Known results in dealing with similarity are not well suited for

*Supported in part by NSF grant IIS-0101134.

GIS'05, November 4, 2005, Bremen, Germany.

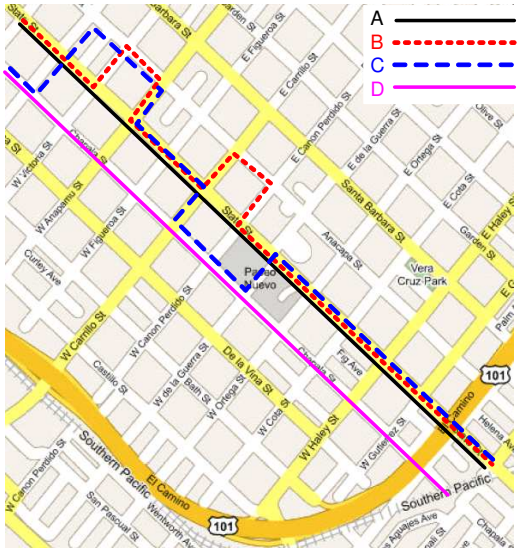

Figure 1: Spatial Similarity of Trajectories [1]

dealing with aggregated trajectory information. Similarity notion for motion tracking in videos are typically invariant with respect to rotations and translations. In time series analysis, trajectories are one dimensional and the time dimension is treated very differently; it is not trivial to employ the techniques in the moving objects setting. In the prior work on similarity of moving object trajectories, the similarity notion takes the time ordering into consideration.

Since existing algorithms focus on movement patterns of moving objects, they cannot be directly applied to the spatial shape similarity search problem. In this paper, we propose a simple but effective similarity distance function and develop algorithms based on this distance function for spatial shapes similarity search of moving object trajectories. The major contributions of this paper are:

- We introduce a new similarity distance definition based on "one way distance" (OWD, Section 3) of trajectories (continuous and discrete).
- We develop an efficient algorithm for computing similarity distance between trajectories in the discrete case (grid representation), which achieves a $O(mn)$ complexity, where n is the length of a trajectory and m is the number of local min points.
- An index structure, which guarantees no false dismissals, is also introduced to enhance the performance of OWD algorithm.

The remainder of the paper is organized as follows. Section 2 discusses related works. Section 3 defines trajectories, OWD distance in different representation, and the similarity search problem. Detailed algorithm description and analysis of OWD are given in Section 4. An index structure for OWD is described in Section 5. Section 6 evaluates OWD by comparing it other algorithms. Section 7 concludes this paper.

2. RELATED WORK

Similarity search has been well studied in the context of time series [6, 2, 15, 14, 7, 3, 12]. Original similarity search algorithm is based on Euclidean distance, which is sensitive to noise and cannot be performed on sequences with different length and sampling rate. Referenceds [6] and [2] are dimensionality reduction methods based on Discrete Fourier Transformation and Discrete Wavelet Transformation respectively to improve search performance with approximation. Reference [7] presents an indexing method called Adaptive Piecewise Constant Approximation, which is also a dimensionality reduction technique for sequence matching based on the Euclidean distance. This technique uses constant-value segments to approximate sequences and achieves better approximation quality [7]. [12] approximates trajectories with Chebyshev Polynomials, then compute the distance based on the polynomials. However, similar to the Euclidean distance, these measures also require that trajectories are of the same length.

[15] introduces Dynamic Time Warping (DTW) distance to make it possible compare sequences with different lengths, with and without time information. DTW is a method that allows local stretch of sequences to minimize the distance between sequences. [14] gives an improved version of DTW, which filters out unpromising sequences by using an index structure with segmentation and lower bounded distance measure. Its indexing techniques are extended and utilized in our index structure (see Section 5). A distance measure which is both metric and allowing time-shifting is introduced in [3]. But all their algorithms require the mapping between trajectories to be continuous and monotonic.

Recently interests of similarity search also arise in trajectory comparison [10, 16, 14, 9, 18, 17, 8, 4, 11]. DTW has some variants for trajectories similarity search [10, 16]. [10, 16] both present a representation of trajectories invariant to rotation, shifting, and scaling. [9] utilizes the distance between minimum bounding rectangle to compute the distance between two multidimensional sequences. But the distance function can not avoid false dismissals. [18] shows an algorithm extended from time series similarity search method which is based on Euclidean distance. But this algorithm can only compare trajectories with the same lengths or with the same time interval. [8] proposes a method which allows global stretch to match sequences similar but in different scale. [4] introduces EDR (Edit Distance on Real sequence) distance function based on edit distance. But the mapping between trajectories in these two algorithms is still continuous and monotonic. [17] compares sequences by extracting their longest common subsequences. This method removes the mapping continuity requirement and is more flexible in similarity comparison. But it still require the mapping between trajectories to be monotonic. In [11] an aggregation method based on rasters is developed for both spatial and spatial-temporal trajectories. But the method can only support aggregation queries, not similarity search queries.

Most of these algorithms focus on movement patterns of moving object trajectories. As we discussed in Section 1, most techniques cannot be directly applied to spatial shape similarity search. One exception is the DTW technique, which is originally introduced in sequences comparison [15]. But its complexity is quadratic and its mapping requires continuity and monotonicity. Comprehensive comparison of DTW and OWD will be presented in Section 5.

3. TRAJECTORY SIMILARITY

In this section we define the two notions of a trajectory based on "linear" and "grid" representations, and the notion of similarity between a pair of trajectories (in their respective representations) using "one way distance" (OWD) from one trajectory to the other. A formulation of the similarity search problem for trajectories is given in Subsection 3.3.

[1] The map is obtained from `http://maps.google.com`.

3.1 Linear representation

In order to define the similarity measure between trajectories, we need to formulate the notion of a trajectory. As mentioned in Section 1, our focus is on the spatial shapes of trajectories. Therefore, there is no time information in a trajectory.

Since completely continuous location information of a trajectory is not always available and the computation cost of such continuous data is likely high, in many applications the (piecewise) linear representation is employed to solve the problems. A linear representation is to approximate a trajectory using a sequence of line segments.

A *line segment* is represented by a pair of points (p, q), the *length* of a line segment (p, q) is defined as the Euclidean distance between the points p, q.

DEFINITION 1. A *(piece wise) linear* (or *PWL*) *trajectory* is a sequence of points $(p_1, p_2, ..., p_n)$, where each adjacent pair of points (p_i, p_{i+1}) $(1 \leqslant i \leqslant n-1)$ is a line segment in the trajectory. The *length* of a trajectory T, denoted as $|T|$, is the sum of lengths of the line segments in it.

For convenience, we assume that trajectories have lengths > 0.

The *distance from* a point p *to* a trajectory T is defined as:

$$D_{\text{point}}(p, T) = \min_{q \in T} D_{\text{Euclid}}(p, q)$$

where $D_{\text{Euclid}}(p, q)$ denotes the Euclidean distance between points p and q.

We now define the one way distance from a trajectory to another trajectory based on $D_{\text{point}}(p, T)$.

DEFINITION 2. The *one way distance* (or *OWD*) from a trajectory T_1 to another trajectory T_2 is defined as the integral of the distance from points of T_1 to trajectory T_2 divided by the length of T_1:

$$D_{\text{owd}}(T_1, T_2) = \frac{1}{|T_1|} \left(\int_{p \in T_1} D_{\text{point}}(p, T_2)\, dp \right)$$

The *distance between two trajectories* T_1 and T_2 is the average of their one-way distances:

$$D(T_1, T_2) = \frac{1}{2} \left(D_{\text{owd}}(T_1, T_2) + D_{\text{owd}}(T_2, T_1) \right)$$

Clearly $D_{\text{owd}}(T_1, T_2)$ is not symmetric but $D(T_1, T_2)$ is. Note that $D_{\text{owd}}(T_1, T_2)$ is the integral of shortest distances from points in T_1 to T_2. If T_1 is a sub-trajectory of T_2, $D_{\text{owd}}(T_1, T_2)$ is 0; if T_1 is very close to a sub-trajectory of T_2, $D_{\text{owd}}(T_1, T_2)$ is close to 0. In both of these cases, the opposite distance $D_{\text{owd}}(T_2, T_1)$ can be very large. That is, $D_{\text{owd}}(T, Q)$ can be understood as "how much trajectory T is similar to the query trajectory Q?" (Smaller values means more similarity). It is easy to see that $D_{\text{owd}}(T_1, T_2)$ can be used in sub-trajectory search. Furthermore, the algorithms developed in this paper can be used in both similarity search and sub-trajectory search.

3.2 Grid representation

As we will show in our experiments, the computation cost of distances between piecewise linear represented trajectories is still high. Therefore, we need to have discrete definition to achieve better performance with acceptable precision. In this paper we consider grid representation to describe discrete trajectories. In the remainder of this paper, our discussions focus on grid representation.

In a grid representation, the entire workspace is divided into equal-size *grid cells*, and each grid cell is labelled according to its position in the x and y dimensions. For instance, the left-bottom grid cell is labelled as $(1, 1)$ and the right-top grid cell is labelled as (m, n), where m and n are total numbers of columns and rows, respectively. Obviously all labels are integer numbers. Given a grid cell $g = (i, j)$, i is called the x-label of g (i.e. $g.x$) and j is the y-label (i.e. $g.y$).

We define the distance between two grid cells g_1, g_2 as follows:

$$D_{\text{grid}}(g_1, g_2) = \sqrt{(g_1.x - g_2.x)^2 + (g_1.y - g_2.y)^2}.$$

Based on $D_{\text{grid}}(g_1, g_2)$, a trajectory in grid representation can now be defined.

DEFINITION 3. A *grid trajectory* is a sequence of grid cells, $T^g = (g_1, g_2, ..., g_n)$ such that for each $1 \leqslant i < n$, g_i and g_{i+1} are adjacent, i.e., $D_{\text{grid}}(g_i, g_{i+1}) = 1$. The number n in the trajectory T^g shown above is called the *length* of T^g, and denoted as $|T^g| = n$.

The distance from a grid cell g to a grid trajectory T^g is defined as the shortest distance from g to T^g.

$$D^g_{\text{point}}(g, T^g) = \min_{g' \in T^g} D_{\text{grid}}(g, g').$$

Similar to Definition 2, the one-way grid distance and the distance between grid trajectories can be defined as follows.

DEFINITION 4. The *one way distance in grid representation* from one trajectory T^g_1 to another trajectory T^g_2 is defined as the sum of the distance from grid cells of T^g_1 to that of T^g_2 divided by the length of T_1.

$$D^g_{\text{owd}}(T^g_1, T^g_2) = \frac{1}{|T^g_1|} \left(\sum\nolimits_{p \in T^g_1} D^g_{\text{point}}(p, T^g_2) \right).$$

The *distance between two grid trajectories* T^g_1 and T^g_2 is defined as the average of $D^g_{\text{owd}}(T^g_1, T^g_2)$ and $D^g_{\text{owd}}(T^g_2, T^g_1)$.

$$D^g(T^g_1, T^g_2) = \frac{1}{2} \left(D^g_{\text{owd}}(T^g_1, T^g_2) + D^g_{\text{owd}}(T^g_2, T^g_1) \right)$$

One important feature of grid representation is its insensitivity to noises. And coarser grid leads to less sensitivity to noises. It is also quite clear that coarser grid representation means less computation cost and accuracy. Therefore, the trade-off between computation cost, accuracy, and sensitivity to noises can be achieved by varying grid cell size in grid representation.

3.3 Similarity Search

Based on the definitions provided in Subsections 3.1 and 3.2, we can now measure the similarity with distance between trajectories. Therefore, similarity search problem can be described as follows.

SIMILARITY SEARCH PROBLEM: *Given a set of trajectories $S = \{T_1, T_2, \ldots, T_n\}$, a query trajectory Q, and a positive integer k, find the k trajectories in S which have the smallest distances to Q.*

4. DISTANCE COMPUTATION

The similarity search problem stated in Section 3 can be addressed in two steps. The first step is to find a similarity comparison algorithm to compute the similarity (distance) between two trajectories. The second step is to find an indexing structure to speed up the search computation through a large number of trajectories.

The similarity comparison algorithms based on one-way distance (OWD) for both PWL and grid representations are given in Subsections 4.1 and 4.2, respectively. Theoretical analysis of both algorithms is also given. In the next section (Section 5) we will focus on the indexing problem for trajectories.

4.1 Linear (PWL) representation

The basic idea of this algorithm is to compute the OWD distance piece by piece. Since a trajectory in PWL representation is composed of a sequence of segments. Given a linear trajectory $T = \{p_1, p_2, ..., p_n\}$, let $LS(T) = \{\textit{linesegment}(p_i, p_{i+1}) \mid 1 \leqslant i \leqslant n-1\}$ represent all line segments in T. Then the one way distance from a trajectory T_1 to another trajectory T_2

$$D_{\text{owd}}(T_1, T_2) = \frac{1}{|T_1|}\left(\sum\nolimits_{s \in LS(T_1)} (D_{\text{owd}}(s, T_2) \cdot |s|)\right)$$

where $|s|$ is the length of a line segment s.

In order to compute $D_{\text{owd}}(s, T_2)$, we view s as a function of t where t is a new variable. Specifically, a line segment $s = (p_1, p_2)$ (p_1 and p_2 are the endpoints of s) can be represented as a function $s(t) = p_1 + (p_2 - p_1)t$, where t is in the range $[0, 1]$. So the OWD distance $D_{\text{owd}}(s, T_2)$ from s to T_2 can be represented by a function $f(t)$ and $D_{\text{owd}}(s, T_2) = \int_0^1 f(t)\, dt$. The function $f(t)$ is called the *global min function* from s to T_2. To compute $f(t)$, it is necessary to examine each segment $s' \in T_2$, compute the OWD distance function from s to s'. Then, $f(t)$ can be obtained as the lowest envelope of all these functions using plane-sweeping algorithms [5] and the algorithms computing the intersections of curves [13].

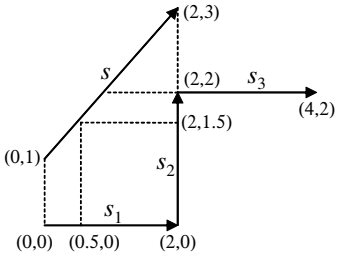

Figure 2: An Example of Global Min Function Computation

Figure 2 shows an example of the computation of $f(t)$. In this example the trajectory T_2 consists of three line segments s_1, s_2, s_3 (in this order). The OWD distance functions f_1, f_2, f_3 from a line segment s to s_1, s_2, s_3 (respectively) can be computed as follows.

$$f_1(t) = 1 + 2t \qquad (0 \leqslant t \leqslant 1)$$
$$f_2(t) = \begin{cases} 2 - 2t & (0 \leqslant t \leqslant 0.5) \\ \sqrt{8t^2 - 12t + 5} & (0.5 < t \leqslant 1) \end{cases}$$
$$f_3(t) = \sqrt{8t^2 - 12t + 5} \qquad (0 \leqslant t \leqslant 1)$$

Using these functions, $f(t)$ can be computed as

$$f(t) = \begin{cases} 1 + 2t & (0 \leqslant t \leqslant 0.25) \\ 2 - 2t & (0.25 < t \leqslant 0.5) \\ \sqrt{8t^2 - 12t + 5} & (0.5 < t \leqslant 1) \end{cases}$$

Figure 3 shows the computation of $f(t)$. It is easy to see that $f(t)$ is the lowest envelope of functions $f_1(t), f_2(t)$, and $f_3(t)$ in the range of $[0, 1]$.

The detailed algorithm is shown in Algorithm 1. The outer "for" loop goes through all segments in T_1, and Lines 3 to 8 compute $D_{\text{owd}}(s, T_2)$. Line 5 computes the OWD distance from a line segment s to another line segment s'. The global min function $f(t)$ from s to T_2 is computed in Line 7 using the algorithms in [13, 5]. Line 10 gets $D_{\text{owd}}(T_1, T_2)$.

The plane sweep algorithm in Line 7 requires $O(n \log n)$ time complexity [5]. It follows that Algorithm 1 has $O(n^2 \log n)$ complexity, where n is the numbers of segments in a trajectory.

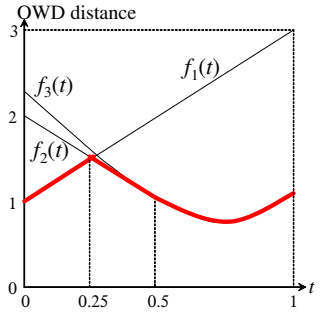

Figure 3: The $f(t)$ Computation of the Example in Figure 2

Algorithm 1 OwdPwl(T_1, T_2)

```
1:  C ← 0
2:  for each segment s = (p1, p2) in T1 do
3:      Initialize an empty function set F
4:      for each segment s' = (q1, q2) in T2 do
5:          Find the parameterized (with t) OWD function
            from s to s', put it into F
6:      end for
7:      Compare all functions in F and
        compute the global min function f(t) from s to T2
8:      C ← C + ∫_0^1 f(t) dt × |s|
9:  end for
10: Return C/|T1|
```

4.2 Grid representation

As discussed in Subsection 4.1, the computation cost of trajectory similarity comparison in the PWL representation is at least quadratic in terms of the number of segments. As we are going to show in this section, This complexity can be improved. A key idea is to find "local min points" of grid cells. It turns out that local min points of a grid cell can be effectively inferred from its neighbor's. Based on local min points, we develop an algorithm of "semi-quadratic" $O(mn)$ complexity for grid trajectories, where n is the length of a trajectory T_1 and m is the average number of local min points of grid cell. In this subsection we focus on the details of this similarity comparison algorithm for grid representation of trajectories.

Before introducing the algorithm, it is necessary to mentioned that in the algorithm the most time consuming part is the distance computation for two grids, which accounts for up to 80% of CPU time in our experiments. So the number of calls of the grid distance computation is the major metric when we analyze the complexity of algorithms for grid representation.

The naive algorithm for OWD in grid representation requires computing the distance for all pairs of grids on different trajectories and has a $O(n^2)$ complexity, where n the length of a grid trajectory as defined in Section 3. But some computation of the distance between grids are not necessary. In this section we will show that a lower complexity $O(mn)$ can be achieved, where n is the length of trajectories and m is the number of local min points.

Before presenting the details of our algorithm, we first introduce the concept of Local Min Point.

DEFINITION 5. (Local Min Point) Given a grid cell g and a trajectories T, a grid cell $g' \in T$ is a *local min point* to g if $D(g'', g) > D(g', g)$ for each grid cell g'' in T adjacent to g'.

In other words, local min points of a grid cell g are those grids

whose distances to g are shorter than those of their neighbors. Figure 4 displays an example of local min points. g is a single grid cell and the grid cells marked with bold lines represent a trajectory. g has three local min points in the trajectory: g_1, g_2, and g_3. And among the three local min points, g_3 is the one whose distance to g is the shortest distance from g to the trajectory.

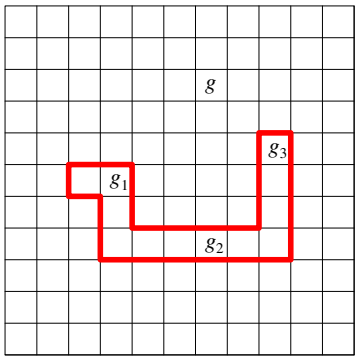

Figure 4: Examples of Local Min Point

According to the definition of local min points, it is clear that to compute $D^g_{\text{owd}}(g, T_2)$, it is only required to compute $D(g, g')$ for all local min points $g' \in T_2$ of g. A better news is that in grid representation local min points can be inferred without using the time consuming grid distance function. Algorithm 2 shows how to determine among two adjacent grid cells g_1, g_2, which one is closer to g. Before we explain this algorithm, it is necessary to establish the following easily verified fact first.

LEMMA 1. *For each pair of two adjacent grid cells g_1, g_2 on a trajectory, either $g_1.y = g_2.y$ and $|g_1.x - g_2.x| = 1$ or $g_1.x = g_2.x$ and $|g_1.x - g_2.x| = 1$.*

According to Definition 3, given any two adjacent grid cells g_1, g_2 on a trajectory, $D^g(g_1, g_2) = 1$ means that $(g_1.x - g_2.x)^2 + (g_1.y - g_2.y)^2 = 1$. Since the coordinates must be integers, either $g_1.y = g_2.y$ and $|g_1.x - g_2.x| = 1$ or $g_1.x = g_2.x$ and $|g_1.x - g_2.x| = 1$.

Therefore, given two adjacent grid cells g_1, g_2, either $g_1.y = g_2.y$ or $g_1.x = g_2.x$. Therefore, we only need to consider these two situations in Algorithm 2. When $g_1.y = g_2.y$ (Lines 1 to 7), it is only required to compare the distances on x-dimension; when $g_1.x = g_2.x$ (Lines 8 to 13), only the comparison on y-dimension is necessary. The distance computation function is invoked in neither case.

Based on Algorithm 2, we can easily find the local min points of a grid cell in a trajectory.

But actually a further improvement can be achieved—in most cases even the calling of Algorithm 2 is not necessary. The reason of this is that a grid cell's local min points can be inferred from its neighbors' local min points. Before showing this statement is true, we first prove the following theorem:

THEOREM 2. *Let T, T' be two trajectories and g_1, g_2 two adjacent grid cells in T. Suppose further that $g' \in T'$ is a local min point to g_1. If $g_1.y = g_2.y$ and $g_1.x \neq g'.x$ or $g_1.x = g_2.x$ and $g_1.y \neq g'.y$, then g' is also a local min point to g_2.*

PROOF. We first prove the case when $g_1.y = g_2.y \wedge g_1.x \neq g'.x$. According to Lemma 1 and Definition 5, for an adjacent grid cell g'' of g', there are only two possibilities:

Algorithm 2 Closer(g, g_1, g_2)

```
1:  if g1.y = g2.y then
2:      if |g1.x − g.x| < |g2.x − g.x| then
3:          return g1
4:      else
5:          return g2
6:      end if
7:  end if
8:  if g1.x = g2.x then
9:      if |g1.y − g.y| < |g2.y − g.y| then
10:         return g1
11:     else
12:         return g2
13:     end if
14: end if
```

(1) $g''.x = g'.x$ and $|g''.y - g'.y| = 1$, and

(2) $g''.y = g'.y$ and $|g''.x - g'.x| = 1$.

For Case (1), it is obvious that $D(g'', g_2) > D(g', g_2)$ since $D(g'', g_1) > D(g', g_1)$ and $g_1.y = g_2.y$. Thus g' is a local min point to g_2.

For Case (2), if $g''.x = g'.x + 1$, then

$$D(g'', g_1) < D(g', g_1) \Leftrightarrow g_1.x \leqslant g'.x$$

Since $g_1.x \neq g'.x$, so

$$g_1.x < g'.x \Leftrightarrow g_2.x \leqslant g'.x \Leftrightarrow D(g'', g_2) < D(g', g_2).$$

Similarly, we can also show that when $g''.x = g'.x - 1$, it is also true that $D(g'', g_2) < D(g', g_2)$. We conclude that g' is also a local min point to g_2.

The case when $g_1.x = g_2.x \wedge g_1.y \neq g'.y$ can be addressed similarly. □

Figure 5 illustrates the main idea of Theorem 2. In the figure g_1 has 4 local min points in T_2: g'_1, g'_2, g'_3, and g'_4. g'_1 is also a local min point of g_2 because $g_1.y = g_2.y \wedge g_1.x \neq g'_1.x$. Similarly, g'_3 and g'_4 are also local min points of g_2. But g'_2 may not be a local min point of g_2 since $g_1.x = g'_2.x$. Actually in this example g'_2 is not. For the same reason, g'_1, g'_2, and g'_3 are local min points of g_3; but g'_4 is not.

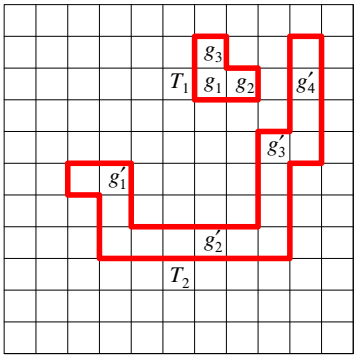

Figure 5: An Example of Theorem 2

Based on Theorem 2, we proceed to construct the OWD algorithm for grid representation. To compute the OWD distance from T_1 to T_2, the local min points of the first grid cell in T_1 are computed. Then we gradually infer other grid cells' local min points

based on their precedents' local min points. The shortest distance from a grid cell in T_1 to T_2 can be computed by comparing its distances to its local min points. The OWD distance from T_1 to T_2 is the sum of all grid cells' (in T_1) shortest distances to T_2 divided by the length of T_1.

The details of the algorithm is given in Algorithm 3. In the initialization (Lines 1 to 4), the local min points from the first grid cell g_1 of T_1 to T_2 are computed by going through T_2, and the shortest distance from g_1 to T_2 is computed as well. The outer "for" loop (Lines 5 to 23) computes the shortest distances from grid cells $g_i (2 \leqslant i \leqslant m)$ to T_2, one after another. When computing the shortest distance of g_i, the local min points of g_{i-1} are utilized (Lines 7 to 19). For each local min point g_p of $g_{i=1}$, if the conditions in Theorem 2 are met, it is also a local min point of g_i (Lines 8 and 9). Otherwise, Algorithm 2 is employed to check if there are local min points of g_i between g_{p-1} and g_{p+1} in T_2 (Lines 11 to 17). According to Theorem 2, it is only necessary to check the grid cells whose x-label (if $g_{i-1}.y = g_i.y$) or y-label (if $g_(i-1).x = g_i.x$) is equal to g_i's.

Algorithm 3 OwdGrid(T_1, T_2)

```
1: m = |T1|, n = |T2|
2: Find the local min points of the first grid cell g1 of T1 in T2
3: Compute the shortest distance d1 from g1 to T2
4: C ← d1, L ← 1
5: for each grid cell gi(2 ⩽ i ⩽ m) in T1 do
6:    Initialize an empty local min point set S
7:    for each local min point gp of gi−1 do
8:       if (gi−1.y = gi.y ∧ gp.x ≠ gi−1.x) ∨
            (gi−1.x = gi.x ∧ gp.y ≠ gi−1.y) then
9:          S ← S + {gp}
10:      else
11:         for each grid cell g′ between gp−1 and gp+1 in T2 do
12:            if (gi−1.y = gi.y ∧ g′.x = gi.x) ∨
                  (gi−1.x = gi.x ∧ g′.y = gi.y) then
13:               if g′ is a local min point of gi
                     (using Algorithm 2 to check) then
14:                  S ← S + {gp}
15:               end if
16:            end if
17:         end for
18:      end if
19:   end for
20:   Compute the shortest distance di from gi to T2 using S
21:   C ← C + di
22:   L ← L + 1
23: end for
24: Return C/L
```

Since the grid distance function is called only between grids of T_1 and their local min points in T_2, the complexity of Algorithm 3 is $O(mn)$, where n is the length of T_1 and m is the average number of local min points in T_2 for each grid cell in T_1. Although m is not completely independent of n, as shown in Section 6, it grows much slower than n. So compared to a quadratic complexity in DTW algorithm, Algorithm 3 achieves a better complexity bound.

5. SIMILARITY SEARCH

In Section 4, we present the OWD algorithms for comparing two PWL or grid trajectories (respectively). When performing similarity search on a large collection of trajectories, an index structure composed of multiple granularity levels can be employed to improve the efficiency. The index structure introduced here is similar to the index structure used in [14]. But we use a different definition of the lower bound distance, which is designed for grid representation, and prove the lower bounding feature of this distance definition.

The index structure is defined as follows.

$$\begin{array}{rcll} I & = & \langle L_1, L_2, ..., L_M \rangle & \\ L_i & = & \langle s_i, T_1^i, T_2^i, ..., T_N^i \rangle & (1 \leqslant i \leqslant M) \\ s_i & = & h \cdot s_{i+1} & (1 \leqslant i \leqslant M-1) \end{array}$$

where I is the index, M is the number of levels, L_i's $(1 \leqslant i \leqslant M)$ are levels with different granularity, N is the number of trajectories, s_i is the grid cell size at level i, T_j^i's $(1 \leqslant j \leqslant N)$ are trajectories at level i, and h is some positive integer. In the index structure, level 1 is the coarsest with the largest grid cell size $s_1 = h^{M-1} s_M$, while level M is the finest level (smallest cell size).

Intuitively, the index is composed of several levels of grid representations of trajectories, from coarsest to finest. The grid cell size of a level is h times of that of its successive level. Figure 6 shows two levels of an index of trajectories. Two linear trajectories (T_1, T_2) are shown, their grid representations consist of shaded grid cells. Note that the grid cell size of the upper level (Figure 6(left)) is twice of that of the lower level (Figure 6(right)).

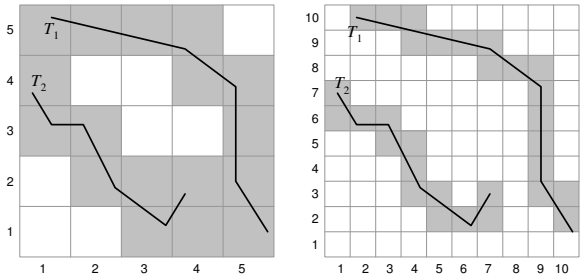

Figure 6: Illustration of Trajectory Index

The structure is similar to the index structure used in [14] for time series similarity. Key differences are: (1) time series data are segmented along the time dimension in [14] and thus their "grid" structure is one dimensional, while our index is based on 2 dimensional grid structures. (2) similarity distance functions used in this paper are very different from that in [14], and consequently, the search operations are different (though both use sequential search). The detail of the distance functions are presented later in this section.

Given a set of trajectories, an index should be built before processing similarity search. Each level is built through sequential search of trajectories. As we can see in the definition, different levels in an index are relatively independent. Insertion of a trajectory into an index can be translated to a series insertions into each level, which is straightforward. The building process is done in a similar fashion as in [14].

The process of a similarity search can be divided into two steps:

- (Step 1) Search k-nearest neighbors in the coarsest level. Compute these k candidates' OWD distance to the query trajectory in PWL representation. Make the largest one the threshold θ.
- (Step 2) Gradually enhance the accuracy by searching from coarser to finer levels. At each level, use θ to filter out non-promising trajectories to reduce the search space of the next level, and update the candidate list and θ accordingly. After searching through all levels, the final candidate list is the answer.

One problem is that the OWD distance in grid representation is not a lower bound of the OWD distance in PWL representation. This will cause false dismissals. Figure 7 demonstrates this problem. There are three trajectories in Figure 7: T_1, T_2, and T_3. It is obvious that the OWD distance between T_1 and T_2 is smaller than that between T_2 and T_3 in PWL representation. But in grid representation shown in the figure, the OWD distance between T_2 and T_3 is 0 because their grid representations are exactly the same. Hence T_3 is closer to T_2 than T_1 in grid representation, which is different from the PWL representation.

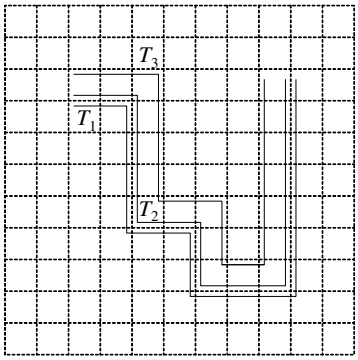

Figure 7: The Problem of OWD Distance

To address this problem, we introduce a *lower bound distance* (LBD) $D^g_{\text{lbd}}(g_1, g_2)$ between two grid cells g_1, g_2 as follows.

$$D^g_{\text{lbd}}(g_1, g_2) = \sqrt{l^2(g_1.x, g_2.x) + l^2(g_1.y, g_2.y)}$$

where

$$l(x_1, x_2) = \begin{cases} 0 & (x_1 = x_2) \\ |x_1 - x_2 - 1| & (x_1 \neq x_2) \end{cases}$$

Basically, the lower bound distance treats the distance between adjacent grid cells (whose x or y-label difference is at most 1) as 0. Lower bound distance is slightly less accurate than grid distance with the same granularity, but it carries a nice lower bound feature as shown in Lemma 3.

LEMMA 3. *Let p_1, p_2 be two points and s, s' two grid cell sizes such that $s' = hs$ for some positive integer $h > 1$. Suppose further that points p_1 and p_2 are mapped to g_1 and g_2 (respectively) in grid representation with grid cell size s, and to g'_1 and g'_2 (respectively) in grid representation with grid cell size s'. then*

$$h \cdot D^g_{\text{lbd}}(g'_1, g'_2) \leqslant D^g_{\text{lbd}}(g_1, g_2)$$

PROOF. We first show that $h \cdot l(g'_1.x, g'_2.x) \leqslant l(g_1.x, g_2.x)$. Note that

$$h \cdot l(g'_1.x, g'_2.x) = \begin{cases} 0 & (g'_1.x = g'_2.x) \\ h \cdot |g'_1.x - g'_2.x - 1| & (g'_1.x \neq g'_2.x) \end{cases}$$

If $(g'_1.x = g'_2.x)$, $h \cdot l(g'_1.x, g'_2.x) = 0 \leqslant l(g_1.x, g_2.x)$.

On the other hand, if $(g'_1.x \neq g'_2.x)$, then $(g_1.x \neq g_2.x)$ must be true since the cell size of g'_1, g'_2 is greater than that of g_1, g_2. Hence $h \cdot l(g'_1.x, g'_2.x) = h \cdot |g'_1.x - g'_2.x - 1| \leqslant |g_1.x - g_2.x - 1| = l(g_1.x, g_2.x)$, noting that $s' = hs$. Therefore, $h \cdot l(g'_1.x, g'_2.x) \leqslant l(g_1.x, g_2.x)$.

Using a similar reasoning, we can show that $h \cdot l(g'_1.y, g'_2.y) \leqslant l(g_1.y, g_2.y)$.

Finally, we have

$$\begin{aligned} & h \cdot D^g_{\text{lbd}}(g'_1, g'_2) \\ &= \sqrt{h^2 \cdot l^2(g'_1.x, g'_2.x) + h^2 \cdot l^2(g_1.y', g_2.y')} \\ &\leqslant \sqrt{l^2(g_1.x, g_2.x) + l^2(g_1.y, g_2.y)} \\ &= D^g_{\text{lbd}}(g_1, g_2) \end{aligned}$$

This concludes the proof. □

From Lemma 3, it is easy to establish Theorem 4 (proof omitted).

THEOREM 4. *Let T_1, T_2 be two trajectories and s, s' be two grid cell sizes such that $s' = h \cdot s$ for some positive integer $h > 1$. If T^g_1, T^g_2 are grid trajectories of T_1, T_2 (respectively) in grid representation with grid cell size s, and similarly, T'^g_1 and T'^g_2 are grid trajectories of T_1, T_2 (respectively) in grid representation with grid cell size s', then the following holds:*

$$h \cdot D^g_{\text{lbd}}(T'^g_1, T'^g_2) \leqslant D^g_{\text{lbd}}(T^g_1, T^g_2)$$

Since continuous trajectories can be treated as trajectories represented with tiny grid cells, we can easily establish Corollary 5.

COROLLARY 5. *Given two PWL trajectories T_1, T_2 and a grid cell size s, if T^g_1 and T^g_2 are grid trajectories of T_1 and T_2 in grid representation with grid cell size s, then*

$$s \cdot D^g_{\text{lbd}}(T^g_1, T^g_2) \leqslant D_{\text{owd}}(T_1, T_2)$$

Corollary 5 indicates that the OWD distance between trajectories in grid representation is always a lower bound of the OWD distance in continuous representation.

We now turn to the index structure. A key algorithm is to search for similar trajectories. The basic idea is to utilize the property of Corollary 5 to reduce the number of OWD distance computation. The tree is searched from coarser level to finer level. For each level, k nearest neighbors candidates are collected and the threshold is computed based on them. Then all trajectories with distance greater than the threshold are filtered out. Upon the end of the tree, we will get the answers to the query.

The detailed algorithm is given in Algorithm 4. Line 1 searches the k-nearest neighbors of Q in the coarsest level (i.e. the highest level) of the index structure I using Algorithm 1 to compare Q with each trajectory in this level and the results are stored in C. A temporary set P is used to keep intermediate trajectories filtered from the previous level. The "for" loop from Lines 3 to 11 implements Step 2 of the similarity search strategy. First, the threshold θ produced from the previous level is computed (Line 4). Then each trajectory in P is examined against the new threshold (Lines 5 and 6). Here $D^{g_i}_{\text{lbd}}(T, Q)$ means the LBD distance between T and Q on level i, and $I.L_i.s_i$ represents the grid cell size of level i. Promissing cadidates are compared with candidate trajectories in C (Line 7), while unpromising trajectories are filtered out (Line 9).

Although the search algorithm has the same worst case complexity as sequential search, it is expected that the index will reduce the complexity for real datasets. Experimental results with synthetic datasets clearly support this claim.

We conslude this section with the following remarks. The similary search technique developed in this paper works fine if the trajectory lengths are different. This is the consequence of the distance model and thus not surprising. Secondly, the technique remains effective if the similary measure is to match a sub-part of a trajectory.

Algorithm 4 Search(S, Q, k, I, M)
Input: S — a set of trajectories,
Q — the query trajectory,
k — number of nearest neighbors,
I — A set of grid representations of trajectories in S with different levels,
M — number of levels in I

1: Find the k-nearest neighbors of Q in $I.L_1$ and put in C their identifiers and distances to Q.
2: $P \leftarrow S$
3: **for** i from 1 to M **do**
4: $\theta \leftarrow \max_{x \in C} D_{\text{owd}}(x, Q)$
5: **for** each trajectory T in P **do**
6: **if** $I.L_i.s_i \cdot D^{g_i}_{\text{lbd}}(T, Q) \leqslant \theta$ **then**
7: $C \leftarrow k$ closest trajectories (and their distances) in $C \cup \{T\}$
8: **else**
9: $P \leftarrow P - T$
10: **end if**
11: **end for**
12: **end for**
13: return C

6. EXPERIMENTAL EVALUATION

In this section we evaluate the grid OWD algorithm (OWD in short in this section) by comparing it with the well known Dynamic Time Warping (DTW) algorithm and the Piecewise Linear OWD algorithm (PWL in short in this section).

The main findings from the experimental study are:

- OWD achieves better accuracy than DTW in spatial shape similarity search.
- OWD out-performs DTW in all settings and the gap between them increases rapidly with length of trajectories.
- OWD can approximate PWL fast with high accuracy.

6.1 Experiments setup and parameter settings

In the experimental evaluation, we choose Network generator—a moving object dataset generator developed by Brinkhoff [1]. Using this generator, we simulate two dimensional trajectories of vehicles on the road network in the city of San Francisco.

The number of trajectories varies from 10,000, 20,000, 50,000, to 100,000. The number of nearest neighbor (i.e., k) is 10, 20, 50, or 100. When evaluating performance, we use CPU time as the standard because all dataset sizes are small enough to fit into memory. All experiments are run on a linux PC with AMD Athlon 900MHz CPU, 256MB memory, and 20GB hard disk. A new parameter *granularity* is introduced to measure the fineness of grid representations, which is the number of divisions on each dimension (we apply the same number of divisions on both dimensions). So higher granularity means finer grid representation. Granularity varies from 10, 20, 40, to 80. The typical setting of experiments, if not explicitly mentioned, is 50,000 trajectories, 20 nearest neighbors, 80 granularity.

6.2 Precision

In this section we evaluate the precision of OWD and DTW by comparing them with PWL. We define the precision of OWD and DTW using the following formula:

$$Precision = \frac{|S \cap S_{pwl}|}{|S_{pwl}|}$$

where S is the result set of OWD or DTW, and S_{pwl} is the result set of PWL with the same setting.

Figure 8 (left) shows the precision results defined above with different granularity. Clearly OWD achieves better precision than DTW in the same grid representation. OWD's precision grows to near 100% when the grid granularity becomes finer, while DTW stays at the level less than 80%. The reason is that DTW ignores trajectories with similar spatial shape but different orientation, since DTW requires mapping between trajectories to be continuous and monotonic.

Figure 8 (right) displays the precision results with different k. It also shows that OWD achieves better precision than DTW. In this case, the parameter k has little impact of the precision, although smaller k has slightly better precisions.

Figure 9 gives another view of precision of OWD. The error percentage of OWD distance to PWL distance is displayed. In this figure we do not include DTW because its distance is in a different measure from PWL. The error percentage $P_{error}$is defined as follows.

$$P_{error} = \frac{|D_{\text{pwl}} - D_{\text{owd}}|}{D_{\text{pwl}}}$$

where D_{pwl} and D_{owd} are distance of PWL and OWD in the same setting, respectively.

It is easy to see that the error percentage decreases rapidly when granularity increases. When granularity equals 80, it is already very small. This also explains why OWD can achieve high precision rapidly.

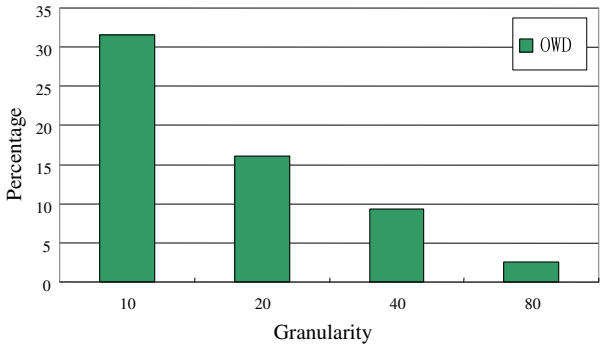

Figure 9: Distance Error Percentage of OWD to PWL

6.3 Scalability

In this set of experiments, we compare the comprehensive performance of OWD and DTW, with and without indexing, using datasets with different number of trajectories. The CPU time for one k-nearest neighbor query over the dataset is used as the comparison standard.

Figure 10 shows the performance of DTW and OWD without indexing. It is shown that OWD performs much better than DTW, especially when dataset size increases.

Figure 11 performs the comparison with indexing. We choose FTW [14] as the index method for DTW, because FTW is the best known index [14] for DTW so far. OWD uses the indexing technique introduced in Section 5. In all dataset sizes, OWD achieves much better performance than FTW. The difference of the performance increases with the size of the dataset.

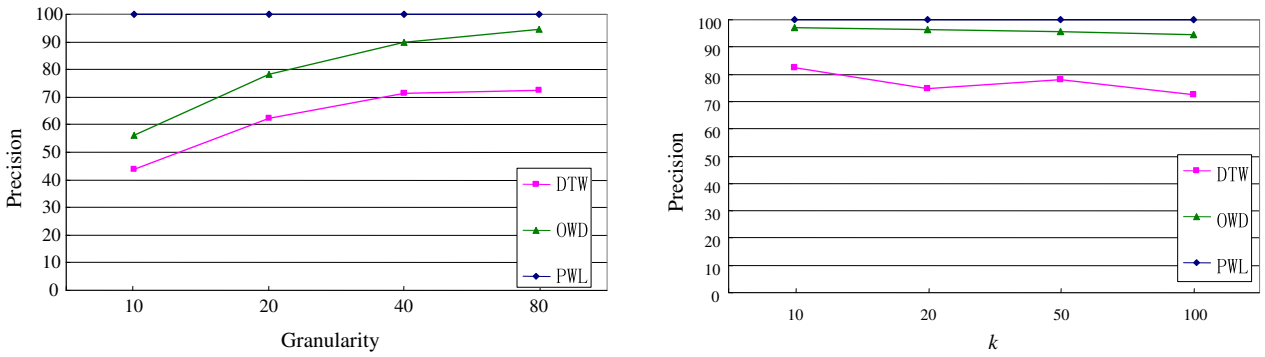

Figure 8: Precision of DTW and OWD with different granularity and k

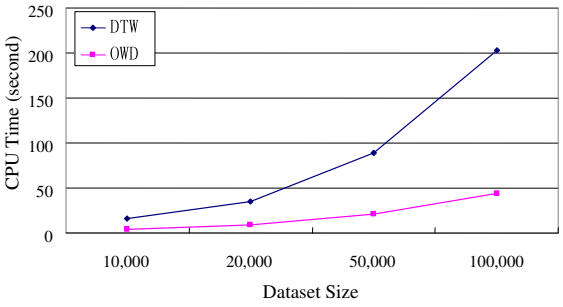

Figure 10: Performance of DTW and OWD without indexing

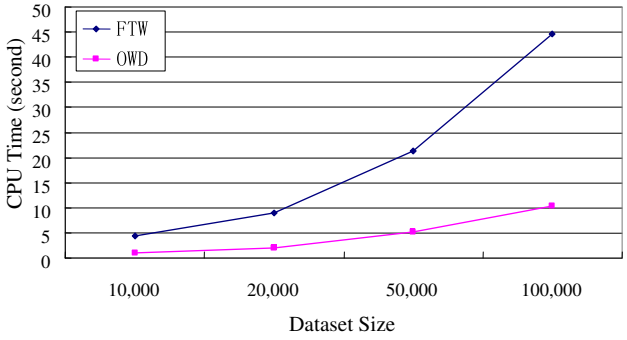

Figure 11: Performance of FTW and OWD with indexing

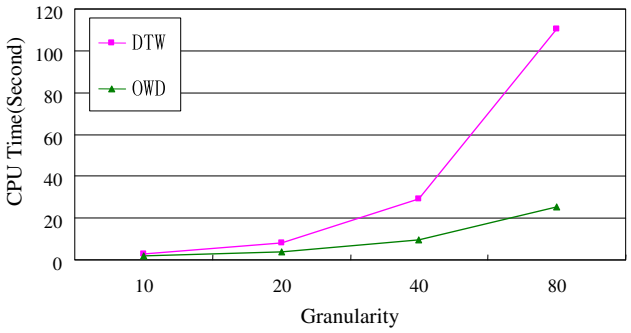

Figure 12: DTW and OWD with different granularity

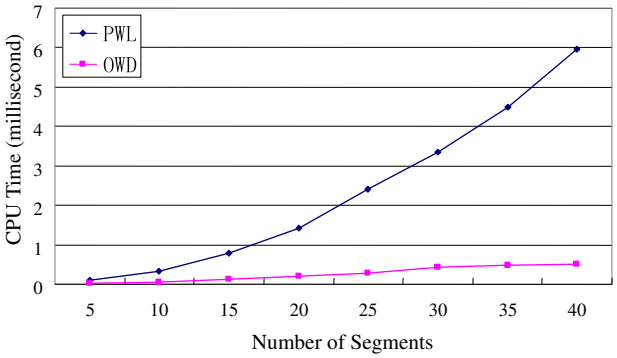

Figure 13: Performance Comparison of PWL and OWD with trajectories of different length

6.4 Performance with different granularity

This section shows the performance comparison between OWD, DTW, and PWL with different granularity.

Figure 12 shows the comparison of OWD and DTW with different granularity. It is clear that the computation cost of DTW is much higher than OWD in all settings. The time consumed by DTW increases much faster than OWD when the granularity grows. This is because DTW has a quadratic complexity while OWD's complexity is $O(mn)$, where n is the length of trajectories and m is the number of local min points (Subsection 4.2).

Figure 13 displays the performance comparison of PWL and OWD. In this figure, the average CPU time for a pair of trajectories similarity comparison is measured. The granularity of OWD is 80. It is shown that PWL is much slower than OWD in all cases. When the number of segments of trajectories increase, the gap between PWL and OWD grows as well. It is already shown in Subsection 5.2 that PWL can be approximated by OWD with reasonable high granularity. So OWD is a good substitute of PWL when performance is critical.

7. CONCLUSIONS

An interesting issue in moving objects databases is to find similar trajectories of moving objects. The similarity can be time sensitive or insensitive. In this paper we study the time independent similarity search problem of moving object trajectories. We introduced a new distance function (OWD) for comparing spatial shapes of trajectories, developed algorithms for computing OWD in both continuous (piecewise linear) and discrete (grid representation) cases, and an index structure which guarantees no false dismissals to enhance search performance. The discrete OWD algorithm achieves

a better complexity than known quadratic algorithms by utilizing local min points in grid representation. Different grid cell size can be chosen to trade-off the accuracy, performance, and sensitivity to noises. Our comprehensive experimental results show that OWD out-performs traditional Dynamic Time Warping (DTW) algorithm in terms of both precision and performance, with and without indexing. Moreover, the continuous OWD distance can be approximated with the discrete one efficiently with reasonable accuracy. Comparison of OWD and other existing algorithms (e.g. [17]) are under investigation. One interesting problem is the choice of granularity to further shorten the search process. It would be interesting to develop theoretical models or heuristic guidelines for the trade-off between running time of the algorithm and accuracy of results. It is also interesting to focus on more flexible representations and algorithms which can accommodate transformations such as spatial shift, rotation, and scaling etc.

8. REFERENCES

[1] T. Brinkhoff. Generating traffic data. *Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society*, 26(2):19–25, 2003.

[2] K. Chan and W. Fu. Efficient time series matching by wavelets. In *Proc. Int. Conf. on Data Engineering*, 1999.

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

[4] L. Chen, M. T. Ozsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proc. ACM SIGMOD*, 2005.

[5] M. de Berg, M. V. Krevel, M. Overmars, and O. Schwarzkopf. *Computational Geometry Algorithms and Applications*. Springer, 2000.

[6] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. ACM SIGMOD*, 1994.

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

[8] E. Keogh, T. Palpanas, V. B. Zordan, D. Gunopulos, and M. Cardle. Indexing large human-motion databases. In *Proc. VLDB*, 2004.

[9] S. L. Lee, S. J. Chun, D. H. Kim, J. H. Lee, and C. W. Chung. Similarity search for multidimensional data sequences. In *Proc. Int. Conf. on Data Engineering*, 2005.

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

[11] N. Meratnia and R. A. de By. Aggregation and comparison of trajectories. In *Proc. ACM Symp. on Advances in Geographic Information Systems*, 2002.

[12] R. Ng and Y. Cai. Indexing spatio-temporal trajectories with chebyshev polynomials. In *Proc. ACM SIGMOD*, 2004.

[13] V. Pan. Complexity of computations with matrices and polynomials. In *SIAM Review*, 1992.

[14] Y. Sakurai, M. Yoshikawa, and C. Faloutsos. FTW: Fast similarity search under the time warping distance. In *Proc. PODS*, 2005.

[15] D. Sankoff and J. B. Kruskal. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparisons*. Addison-Wesley, 1983.

[16] M. Vlachos, D. Gunopulos, and G. Das. Rotation invariant distance measures for trajectories. In *Proc. of SIGKDD*, 2004.

[17] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Proc. Int. Conf. on Data Engineering*, pages 673–684, 2002.

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