Efficient Similarity Join of Large Sets of Moving Object Trajectories

Hui Ding, Goce Trajcevski*and Peter Scheuermann†
Dept. of EECS, Northwestern University
2145 Sheridan Road
Evanston, IL 60208, U.S.A.

Abstract

We address the problem of performing efficient similarity join for large sets of moving objects trajectories. Unlike previous approaches which use a dedicated index in a transformed space, our premise is that in many applications of location-based services, the trajectories are already indexed in their native space, in order to facilitate the processing of common spatio-temporal queries, e.g., range, nearest neighbor etc. We introduce a novel distance measure adapted from the classic Fréchet distance, which can be naturally extended to support lower/upper bounding using the underlying indices of moving object databases in the native space. This, in turn, enables efficient implementation of various trajectory similarity joins. We report on extensive experiments demonstrating that our methodology provides performance speed-up of trajectory similarity join by more than 50% on average, while maintaining effectiveness comparable to the well-known approaches for identifying trajectory similarity based on time-series analysis.

1 Introduction

The advances in Global Positioning Systems, wireless communication systems and miniaturization of computing devices have brought an emergence of various applications in Location-Based Services (LBS). As a result, there is an increasing need for efficient management of vast amounts of location-in-time information for moving objects. An important operation on spatio-temporal trajectories, which is fundamental to many data mining applications, is the similarity join [5], [15]. Given a user defined *similarity measure*, a similarity join identifies all pairs of objects that satisfy the join predicate in which the condition is specified by the measure. Efficient similarity joins are especially desirable for spatio-temporal trajectories, because the distance calculation between trajectories is generally very expensive due to the intrinsic characteristics of the data.

*Research supported by the Northrop Grumman Corp., contract: P.O.8200082518

†Research supported by the NSF grant IIS-0325144/003

Previous research efforts on efficient similarity search in time series data sets mainly follow the GEMINI framework [9], [12], [17], [24]: given a similarity measure on the time series, each trajectory is transformed into a point in a high-dimensional metric space and an index is constructed in the transformed space using the defined measure (or the lower-bounding measure if one is proposed). These *transformed space approaches* have been proved efficient for a large number of different similarity measures in a variety of time series application domains.

When it comes to *moving object trajectories* which constitute a special category of time series data, we observe that one can perform the similarity join more efficiently using a different approach. The transformed space approaches [9], [12], [17], [24] incur extra overheads by building dedicated index structures and applying trajectory transformations. However, one can exploit the fact that trajectories are often already indexed in their *native space*, in order to facilitate processing of the common spatio-temporal queries such as range, nearest neighbor, etc. [7], [18], [20]. Existing spatio-temporal database prototypes have limited support for trajectory similarity join. Typically, joins are implemented as nested loop joins or sorted-merge joins, which require a large amount of expensive distance computation [14], [28]. The main focus of this work is to provide efficient and scalable similarity joins of spatio-temporal trajectories.

Our main contributions can be summarized as follows:

- We introduce a novel distance measure based on the Fréchet distance [1], which is highly effective in identifying similar trajectories.
- We propose lower and upper bounding approximations of the exact distance measure, which are straightforwardly applicable to the spatio-temporal indices and can prune a significant portion of the search space.
- We present an efficient trajectory similarity join in the native space, which combines the distance calculations with incremental accesses to the spatio-temporal indices.
- We conduct extensive experimental evaluations to show the efficiency and effectiveness of our proposed techniques.

1530-1311/08 $25.00 © 2008 IEEE
DOI 10.1109/TIME.2008.25

The rest of this paper is organized as follows. Section 2 provides the necessary background. Section 3 formally defines our distance metric and the approximation bounds. Section 4 elaborates on our index-based trajectory join framework. Section 5 presents our experimental results. Section 7 reviews related work and concludes the paper.

2 Preliminary

In this section, we introduce the concept of spatio-temporal trajectories, and discuss the existing similarity measures and the indexing of trajectories using R-tree.

2.1 Trajectories and Similarity Measures

We assume that objects move in a two-dimensional space, and that a trajectory Tr is a sequence of points $p_1, p_2, ..., p_i, ..., p_n$, where each point p_i represents the location of the moving object at time t_i, and is of the form (x_i, y_i, t_i), for $t_1 < t_2 < ... < t_i < ... < t_n$. For a given trajectory, its number of points is called the *point length* (p-length) of the trajectory. The time interval between t_1 and t_n is called the *duration* of the trajectory, denoted by ΔTr. The portion of the trajectory between two points p_i and p_j (inclusive) is called a *segment* and is denoted as s_{ij}. A segment between two consecutive points is called a *line segment*. If the sampling rates of trajectories are different, resulting in trajectories with positions sampled at different time stamps. we could simply perform a re-sampling and insert artificial locations by applying linear interpolation.

Several distance measures for trajectories have been proposed in the literature. The *L_p-norms* [12] are the most common similarity measures. For example, given two trajectories Tr_i and Tr_j of the same p-length, one can define the similarity measure based on the Euclidean distances between the corresponding points as: $\mathrm{L}_2(Tr_i, Tr_j) = \sqrt{\sum_{k\in[1,n]} dist(p_k^i, p_k^j)}$, where $dist(p_k^i, p_k^j) = (p_k^i.x - p_k^j.x)^2 + (p_k^i.y - p_k^j)^2$. While L_2 can be calculated in time linear to the length of the trajectories, it is sensitive to noise and lacks support for local time shifting, i.e., trajectories with similar motion patterns that are out of phase. The *Dynamic Time Warping* (DTW) distance [17] overcomes the above problem by allowing trajectories to be stretched along the temporal dimension, and is recursively defined as: $DTW(Tr_i, Tr_j) = dist(p_1^i, p_1^j) + min(DTW(Rest(Tr_i), Rest(Tr_j)), DTW(Rest(Tr_i), Tr_j), DTW(Tr_i, Rest(Tr_j)))$, where $Rest(Tr_i) = p_2^i, ..., p_n^i$. To reduce the impact of the quadratic complexity of DTW on large data sets, a lower-bounding function together with a dedicated indexing structure was used for efficient pruning [17]. Similar to DTW, other distance measures have also been proposed, e.g., the *Edit Distance on Real Sequence* (EDR) [9] and the distance based on *Longest Common Subsequence* (LCSS) [24]. The commonality is that they all follow the transformed space approach, and are not designed to utilize the spatio-temporal indices available in the native space. Recently,Pelekis *et al.* [19] identified several different similarity distance for trajectories, and argued that each of them is more appropriate than the others in different settings.

2.2 R-tree Based Indexing of Trajectories

The R-tree and its variants have been widely used for indexing arbitrary dimensional data [18]. An R-tree is a B+-tree like access method, where each R-tree node contains an array of *(key, pointer)* entries where *key* is a hyper-rectangle that minimally bounds the data objects in the subtree pointed at by *pointer*. In a leaf node, the *pointer* is an object identifier, while in a non-leaf node it is a pointer to a child node on the next lower level.

When indexing spatio-temporal trajectories with the transformed space approach, each trajectory is first transformed into a single point in a high-dimensional (metric) space and a high-dimensional indexing structure is used to index these points. Under this GEMINI framework [12], a high-dimensional R-tree is but one optional index structure.

However, spatio-temporal trajectories can also be indexed in their native space. Several such implementations have been developed in the moving object database literature [7], [18], [20] for processing various spatio-temporal queries. Directly indexing the entire trajectories may introduce large dead space and decrease the discriminating power of the index, hence the general idea is to split a long trajectory into a number of segments, and index the segments [18]. Each leaf node in the R-tree contains a number of 3-dimensional minimum bounding hyper-rectangles (MBR) that enclose the segments generated from splitting, together with unique identifiers that match each segment to its corresponding trajectory. The segments of the trajectories do not have to be of the same length, and a particular leaf node may contain segments from different trajectories. The problem of optimally splitting the trajectories to support efficient data mining has recently been investigated in [3] and is beyond the scope of this paper.

3 Spatio-Temporal Distance of Trajectories

In this Section, we introduce our new distance measure based on the classical Fréchet distance [1]. We observe that the commonly used similarity measures may not be appropriate for trajectories of moving objects and have a questionable applicability to sptio-temporal indices in the native space. To exemplify this, we paraphrase the popular *man walking dog* example [1] in Figure 1. If the Hausdorff distance [2] is used, one can assert that the motion of the man and the dog are always within a distance of ε. However, this measure pertains only to the *routes* of the man and the dog, completely ignoring the dynamics of their representative motions [22]. As illustrated in Figure 1, the locations of the man and the dog at some time instance

can be much larger than ε. Popularly, this means that the minimum length of leash is the fine measure of their distance.

A more general distance function is the Fréchet distance, defined for two continuous curves $f : [a_1, b_1]$ and $g : [a_2, b_2]$ as: $\delta_F(f,g) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \|f(\alpha(t)) - g(\beta(t))\|$, where α (β) ranges over all the possible continuous and monotonically increasing functions $[0,1] \to [a_1, b_1]$ ($[0,1] \to [a_2, b_2]$). Spatio-temporal trajectories in real settings consist of series of coordinate points at discrete time stamps, and the location of a moving object between these points is obtained via interpolation when needed. Hence it suffices to define a *discrete* version of the Fréchet distance as follows [11]:

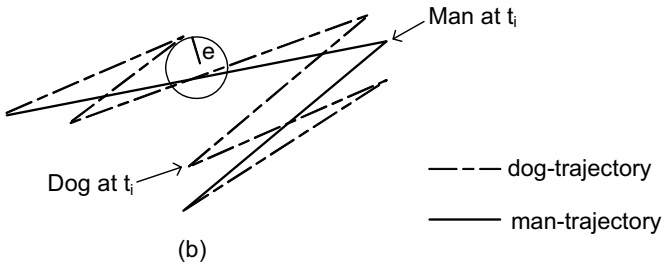

Figure 1: *Illustration of Fréchet Distance*

Let $Tr_1 = (p_1^1, \ldots, p_n^1)$ and $Tr_2 = (p_1^2, \ldots, p_n^2)$ be two trajectories. A *coupling* C between Tr_1 and Tr_2 is a sequence $\{(p_{a_1}^1, p_{b_1}^2), (p_{a_2}^1, p_{b_2}^2), \ldots, (p_{a_k}^1, p_{b_k}^2)\}$ of distinct pairs such that $a_1 = 1, b_1 = 1, a_k = n, b_k = n$ and for all a_i and b_i we have $a_{i+1} = a_i$ or $a_{i+1} = a_i + 1$, $b_{i+1} = b_i$ or $b_{i+1} = b_i + 1$, i.e., the matching is monotonically non-decreasing. The *length* $\|C\|$ of the coupling C is the maximum *link* of all the pairs in the coupling C, where a link is defined as the Euclidean distance between the two points in the pair. That is, $\|C\| = \max_{i=1,\ldots,k} dist(p_{a_i}^1, p_{b_i}^2)$. Finally, the discrete Fréchet distance between two trajectories Tr_1 and Tr_2 is defined as $\delta_{dF}(Tr_1, Tr_2) := \min\{\|C\| : C \text{ is a coupling of } Tr_1 \text{ and } Tr_2\}$. An important observation is that exploring all the possible couplings is exhaustive and costly. By considering all pairs of $(p_{a_i}^1, p_{b_i}^2)$ without paying attention to their temporal distances may distort the real spatio-temporal similarity of moving objects. Motivated by this, we further constrain the definition of the discrete Fréchet distance by considering only pairs of points whose temporal distances are bounded by a given window threshold. The *w-constrained discrete Fréchet distance* (*w*DF) is defined as follows:

DEFINITION 3.1. *Given two trajectories Tr_1 and Tr_2, their w-constrained discrete Fréchet distance $\delta_{wDF}(Tr_1, Tr_2) := \min\{\|C_w\| : C_w$ is a w-constrained coupling of Tr_1 and Tr_2, s.t. $\forall(p_{a_i}^1, p_{b_i}^2) \in C_w \Rightarrow \|p_{a_i}^1.t - p_{b_i}^2.t\| \le w\}$, where w is a parameter that determines the limit of the temporal matching window.*

The importance of the temporal dimension is emphasized by the matching window. An idea similar to ours (temporal matching window constraint) has also been used for other similarity measures [25], where a window size of $5\% - 20\%$ of the entire trajectory duration ΔTr is reported sufficient for most application in terms of finding similar trajectories. Further stretching the temporal matching window not only result in longer execution time of the distance function, but may deteriorate the accuracy of the distance measure due to over-matching. δ_{wDF} has the following properties:

(1) $\delta_{wDF}(Tr_1, Tr_1) = 0$, (2) $\delta_{wDF}(Tr_1, Tr_2) = \delta_{wDF}(Tr_2, Tr_1)$ and (3) $\delta_{wDF}(Tr_1, Tr_2) \le \delta_{wDF}(Tr_1, Tr_3) + \delta_{wDF}(Tr_3, Tr_2)$. Hence, we have:

PROPOSITION 3.1. *δ_{wDF} defines a pseudo-metric on the set of spatio-temporal trajectories.*

Due to space limit, the proofs of the claims are omitted from this paper and are presented in [10].

The *w*DF distance can be computed using dynamic programming and has a complexity of $O(\frac{w}{\Delta Tr} n^2)$. However, unlike DTW and EDR, *w*DF is a pseudo-metric and can utilize the triangular inequality for pruning during similarity search [8]. More importantly, *w*DF has led us to the derivation of two approximation distances that provide even greater pruning power, which we discuss next.

3.1 Efficiency and Approximation of *w*DF

For long trajectories, the brute force computation of *w*DF can be costly. We propose two efficient approximations that can bound the exact *w*DF distance and are much faster to compute: one that guarantees a lower-bound and one that guarantees an upper-bound for the exact *w*DF distance, respectively. The proposed approximations make use of a coarser representation of the spatio-temporal trajectories, obtained through splitting a given trajectory into segments and representing them by the sequence of MBRs that enclose the corresponding segments.

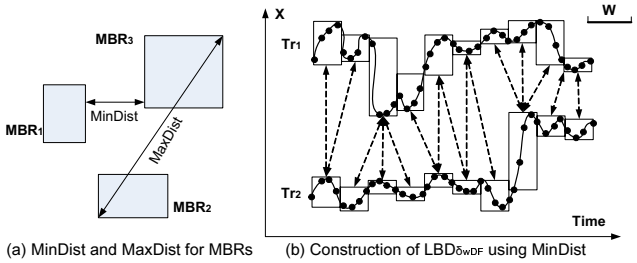

Figure 2: *Bounding the exact wDF distance with MBRs*

Consider two trajectories Tr_1 and Tr_2, each approximated by a sequence of MBRs, e.g., $M_1 = \{MBR_1^1, \ldots, MBR_t^1\}$, $M_2 = \{MBR_1^2, \ldots, MBR_s^2\}$, the *lower-bound coupling* C_w^L between M_1 and M_2 is defined as a monotonically non-decreasing matching between the

pairs of MBRs from each sequence. In particular, the link of a pair in the lower-bound coupling C_w^L is defined as the *MinDist* between the two composing MBRs, i.e., the minimum spatial distance between any two points from the respective MBR (c.f. Figure 2 (a)). The length $\|C_w^L\|$ of the lower-bound coupling C_w^L is $\max\{MinDist(MBR_{u_i}^1, MBR_{v_i}^2)\}$ where $u_1 = v_1 = 1$, $u_k = t$ and $v_k = s$. The w-constraining condition is specified over the time intervals of MBRs. Assuming that MBR_i^1 and MBR_j^2 enclose segment $(p_{i_1}^1, ..., p_{i_k}^1)$ and $(p_{j_1}^2, ..., p_{j_m}^2)$ respectively, they will be considered as a possible pair in a w-constrained coupling only if $\exists p_{i_a}^1, p_{j_b}^2$ *s.t.* $\|p_{a_i}^1.t - p_{b_i}^2.t\| \leq w$. As an example, consider the scenario depicted in Figure 2 (b), due to the limit of the warping window size w, every MBR can only be matched with its corresponding MBR on the other trajectory, and the two MBRs next to it, e.g., the first MBR of Tr_2 can only be matched with the first two MBRs of Tr_1, although its *MinDist* to the third MBR of Tr_1 is much smaller. Formally:

DEFINITION 3.2. *Given two sequences of MBRs M_1 and M_2 for trajectories Tr_1 and Tr_2 respectively, the lower-bound distance of Tr_1 and Tr_2 is: $LBD_{\delta_{wDF}}(M_1, M_2) := \min\{\|C_w^L\|$: C_w^L is a $w-$constrained lower-bound coupling of M_1 and $M_2\}$.*

Similarly, we define an *upper-bound coupling* C_w^U on the two sequences of MBRs M_1 and M_2, where the link of a pair is defined as the *MaxDist* between the two composing MBRs, provided that the temporal w-constraint holds:

DEFINITION 3.3. *Given two sequences of MBRs M_1 and M_2 for trajectories Tr_1 and Tr_2 respectively, the upper-bound distance of Tr_1 and Tr_2 is: $UBD_{\delta_{wDF}}(M_1, M_2) := \min\{\|C_w^U\|$: where C_w^U is a $w-$constrained upper-bound coupling of M_1 and $M_2\}$.*

The construction of $LBD_{\delta_{wDF}}$ between two trajectories from their MBRs is illustrated in Figure 2 (b), and the relationship of these two distance bounds and the exact distance is given by the following:

THEOREM 3.1. *Given two trajectories Tr_1 and Tr_2, and the corresponding sequences of MBRs, M_1 and M_2, that approximate them, for any matching window w the following holds: $LBD_{\delta_{wDF}}(M_1, M_2) \leq \delta_{wDF}(Tr_1, Tr_2) \leq UBD_{\delta_{wDF}}(M_1, M_2)$.*

We note that Theorem 3.1 applies to arbitrary trajectory splitting strategies, and the problem of optimally splitting the trajectories is beyond the scope of this paper [3]. For simplicity, we assume in the rest of this paper that the trajectories are uniformly split into segments of equal length l. From their definitions, $LBD_{\delta_{wDF}}$ and $UBD_{\delta_{wDF}}$ can be computed using the same dynamic programming algorithm for computing wDF, except that the MinDists/MaxDists between MBRs are used instead of Euclidean distance between points. However, the amount of distance computation involved can be greatly reduced because of the coarser representation. This can be illustrated by using the *warping matrix* concept [21] to describe the relevant coupling between two trajectories. The values in the cells of the warping matrix denote the distances between the corresponding matching MBRs/points. Figure 3 (b) shows the warping matrix between the MBRs of the two trajectories Tr_1 and Tr_2, and Figure 3 (c) shows the warping matrix between the actual points of the two trajectories. Intuitively, calculating $LBD_{\delta_{wDF}}(UBD_{\delta_{wDF}})$ or wDF is the process of finding a *path* [21] from the lower-left corner to the upper-right corner that minimizes the maximum value over all cells on the path. The amount of computation for $LBD_{\delta_{wDF}}(UBD_{\delta_{wDF}})$ is significantly less because of the reduced matrix size. Instead of computing the exact δ_{wDF} distance each time, we compute the $LBD_{\delta_{wDF}}$ and $UBD_{\delta_{wDF}}$ with only $O(\frac{w}{\Delta Tr}\frac{n^2}{l^2})$ time, and exclude unqualified candidates, substantially reducing the number of the wDF distance calculations. In the example of Figure 3 (b), we computed a path (shown in large grey cells) for the $LBD_{\delta_{wDF}}(UBD_{\delta_{wDF}})$, then in Figure 3 (c), we only need to consider the matrix cells that are covered by the grey path, and eventually obtained our warping path (shown in small black cells). The exact δ_{wDF} distance is then the maximum value of the black cells. Apparently the search space has been significantly reduced. Theorem 3.1 ensures that the MBR-based pruning will not introduce any false negatives.

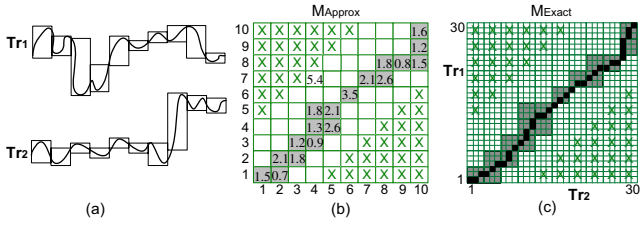

Figure 3: *Warping matrices for calculating $LBD_{\delta_{wDF}}$ /$UBD_{\delta_{wDF}}$ and wDF: X cells are automatically excluded by the temporal matching window, grey cells are potentially useful and black cells are on final path*

Moreover, we can do even better with the approximation distances by further limiting the search space, using an idea similar to *early abandoning* [27]. Consider the warping matrix M_{Approx} in Figure 3 (b) for calculating $LBD_{\delta_{wDF}}$ between Tr_1 and Tr_2. Initially, it only consists of "x" cells and white cells and all the white cells are assigned a value of ∞. We access MinDists between the MBRs in ascending order, and update the values of the corresponding cells, e.g., cell$(1,2)$=0.7, cell$(8,9)$ = 0.8, cell$(3,4)$ = 0.9, ...(grey cells). After each update, we invoke the dynamic programming algorithm and try to find a path for computing the $LBD_{\delta_{wDF}}$. At first there is no such path available, and the algorithm will quickly return ∞. After updating a cell (i, j), if we obtain the first path connecting the two corners in the matrix, then this is the optimal path (since any path formed

later will use a cell updated after cell (i, j) and will have a larger MinDist value than cell (i, j)). Consequently, the $LBD_{\delta_{wDF}}$ distance is equal to the MinDist value of cell (i, j). At this point the distance calculation has been completed and the rest of the cells can be pruned. In the example of Figure 3 (b), the critical cell after which we could find the path is $cell(6,6)$ and as a result, $LBD_{\delta_{wDF}}(Tr_1, Tr_2)$ equals 3.5. Note that cells such as $(7,4)$ are never considered at any time since its value is greater than 3.5. This important observation is formalized as follows:

THEOREM 3.2. *When calculating $LBD_{\delta_{wDF}}(UBD_{\delta_{wDF}})$ between two trajectories Tr_1 and Tr_2, if the pairwise distances between the MBRs are incrementally accessed in ascending order of their MinDists (resp. MaxDists), the number of MinDists (resp. MaxDists) calculation and accesses is minimum.*

Theorem 3.2 requires that the MinDists/MaxDists are sorted in ascending order, which may incur an extra overhead. However, the key observation is that such an ordering can be naturally obtained by maintaining a priority queue while accessing the MBRs in the R-tree [15]. The worst case complexity is still bounded by $O(m^2)$, where m is the number of MBRs in each trajectory. However, we observed in our experiments that in practice significant speed up can be achieved, since not all m^2 cells of the warping matrix need to be evaluated. In addition, although both $LBD_{\delta_{wDF}}$ and $UBD_{\delta_{wDF}}$ can be calculated using Theorem 3.2, in practice we only invoke dynamic programming once to calculate $LBD_{\delta_{wDF}}$. Instead of calculating the exact $UBD_{\delta_{wDF}}$, we use the path for $LBD_{\delta_{wDF}}$ to calculate an upper-bound on $UBD_{\delta_{wDF}}$, which in practice approximates the actual $UBD_{\delta_{wDF}}$ distance very well. More specifically, we straightforwardly compute the corresponding *MaxDist* values, using precisely the pairs of matching MBRs from the $LBD_{\delta_{wDF}}$ coupling. We then use the maximum of these *MaxDist*s to bound the $UBD_{\delta_{wDF}}$ from above. By definition of $UBD_{\delta_{wDF}}$, this bound is guaranteed to be greater than or equal to the value of $UBD_{\delta_{wDF}}$.

4 Index-Based Trajectory Join Under *w*DF

In this section, we present our framework for spatio-temporal similarity join of trajectories under the *w*DF distance measure. Assuming that each trajectory is uniformly split into segments that are indexed by a 3-dimensional R-tree, we describe the nearest neighbor join algorithm, and present several important variants.

4.1 Nearest Neighbor Join

Given two sets of spatio-temporal trajectories, the *nearest neighbor join* retrieves for each trajectory from the first set its most similar trajectory from the second set, using *w*DF as the similarity measure.

The inputs to the algorithm are the two trajectory sets $\mathbb{S}_1$ and $\mathbb{S}_2$, indexed by disk-based R-trees R_1 and R_2, respectively. The algorithm accesses both trees in a manner similar to the incremental distance join [15]: descending from their roots simultaneously, and concurrently assigning the segments from the second set to the closest trajectory from the first set. An important data structure used in the algorithm is a priority queue of the form $(elem1, elem2, mindist)$. The first item in each triple is from R_1 and the second one from R_2. Each item can be either a node of the R-tree, or a leaf node entry, i.e., the MBR of a particular segment. The third item in a triple is the *MinDist* between the first two items, and the dequeue operation will always remove the triple with the smallest *MinDist*. In addition to the priority queue, the algorithm uses two other data structures. The first is the *warping matrix directory* (WMD) that maintains an entry for each trajectory from $\mathbb{S}_1$, storing a list of incomplete $LBD_{\delta_{wDF}}$ and $UBD_{\delta_{wDF}}$ warping matrices between that trajectory and a trajectory from $\mathbb{S}_2$. Each entry in WMD also maintains an *upper bound distance*, which is the maximum possible distance allowed to become an answer candidate. In addition, each entry has a flag that indicates whether the nearest neighbor for this particular trajectory has been found. The second structure is the *candidates table* (CT) that stores for each trajectory from $\mathbb{S}_1$ its candidate answers in a sorted list, in ascending order of the $LBD_{\delta_{wDF}}$.

The join process is illustrated in Algorithm 1. After initializing the relevant data structures, the main body of the algorithm is a while loop that continuously processes the next triple dequeued:

- When both elements in the triple are MBRs of trajectory segments (line 4-17), it first checks whether the corresponding entry from WMD is *complete* and if so, simply discards the triple from further consideration. Otherwise, it performs early abandoning by checking whether the $LBD_{\delta_{wDF}}$ is less than the upper bound distance (line 7). This check uses the fact that $LBD_{\delta_{wDF}}$ is greater than the *MinDist* between the two MBRs. Then the relevant warping matrices in the corresponding WMD entry are updated and the algorithm examines whether the update generates a complete path in the $LBD_{\delta_{wDF}}$ warping matrix. If so, the $LBD_{\delta_{wDF}}$ and $UBD_{\delta_{wDF}}$ distances are calculated. $LBD_{\delta_{wDF}}$ is used to insert a new entry into the candidates table, and $UBD_{\delta_{wDF}}$ is used to update the upper bound distance of the entry. Finally, if the *MinDist* is greater than the entry's upper bound distance, this WMD entry is flagged *complete* since the corresponding $LBD_{\delta_{wDF}}$ will be greater than the upper bound distance, and the relevant warping matrices are discarded.
- When only the first element in the triple is the MBR of a segment (line 18-22), the algorithm checks whether the corresponding entry in WMD is flagged *complete*, and if so the triple is discarded since it (and any new triple formed by further descending the R-tree) may not produce a better answer than the existing candidate. Otherwise the second

Algorithm 1 Index-Based Trajectory Join

```
Input: R-tree R1, R2; Trajectory set S1, S2; temporal matching window w
    /* filtering stage */
 1: priority queue Q.ENQUEUE(R1.root, R2.root, 0)
 2: while ! Q.ISEMPTY do
 3:     (e1, e2, mindist) ⇐ Q.DEQUEUE
 4:     if both e1 and e2 are segment MBRs then
 5:         Tr1 ⇐ trajectory of e1, Tr2 ⇐ trajectory of e2
 6:         if WMD[Tr1] flagged incomplete then
 7:             if MinDist(e1,e2) ≤ E.upper_bound_dist then
 8:                 insert MinDist, MaxDist of e1, e2 into WMD[Tr1]
 9:                 if a path exists for the MinDist warping matrix between
                    Tr1 and Tr2 then
10:                     compute LBD_δwDF and an upper bound of UBD_δwDF
                        between Tr1, Tr2
11:                     if UBD_δwDF < E.upper_bound_dist then
12:                         E.upper_bound_dist ⇐ UBD_δwDF
13:                     insert Tr2 and LBD_δwDF into CT[Tr1]
14:             else
15:                 set flag of WMD[Tr1] as complete
16:         else if WMD[Tr1] flagged complete then
17:             discard the pair (e1, e2)
18:     else if e1 is segment MBR then
19:         if WMD[Tr1] flagged complete then
20:             discard the triple (e1, e2, mindist)
21:         else
22:             expandElement(e1, e2, Q)
23:     else if e2 is segment MBR then
24:         expandElement(e2, e1, Q)
25:     else if both e1 and e2 are node then
26:         expandBalancedElement(e1, e2, Q)
    /* refinement stage */
27: for every entry Tri in CT do
28:     compute δwDF(Tri, Trj) for each candidate Trj until the nearest
        neighbor is found
```

node is expanded by calling the function *expandElement*. *ExpandElement* expands one of the input nodes by pairing each one of its children with the other input element if they are temporally within w, and inserts the resulting triples into the priority queue Q.

- When only the second element in the triple is the MBR of a segment (line 23-24), the first node is expanded by calling *expandElement*, with *elem*1 and *elem*2 exchanged.
- When a pair of nodes is processed (line 25-26), the algorithm chooses to expand one of the nodes by calling *expandBalancedElement* which tries to keep the balance of the depth when descending the two trees. The node to expand is the one with a shallower depth or with a larger area if both nodes are at the same depth [15].

After the while loop terminates, the refinement step is performed on the CT using the triangular inequality of wDF for pruning (line 27-28). For every entry of the candidates table, we examine the candidate trajectories in ascending order of their $LBD_{\delta_{wDF}}$ and calculate the exact wDF distance, until either all the candidate trajectories have been examined, or the next $LBD_{\delta_{wDF}}$ is greater than the largest computed wDF distance value.

4.2 Variants of the Similarity Join

Algorithm 1 for nearest neighbor join requires minor modifications to calculate other similarity joins among trajectories in our framework.

- **k-nearest neighbor join** (kNN join) [5]: A kNN join finds for each trajectory from $\mathbb{S}_1$ its k nearest neighbors from $\mathbb{S}_2$ in terms of the wDF distance. For each trajectory from $\mathbb{S}_1$, after the first k candidates are added to the candidate table, the minimum of their $UBD_{\delta_{wDF}}$ is used as the upper bound. We continue to add new candidates as long as their $LBD_{\delta_{wDF}}$ distances are smaller than the current upper bound, and update the upper bound with the new tighter $UBD_{\delta_{wDF}}$ if necessary. This only requires that line 11-12 in Algorithm 1 are changed to maintain the *upper_bound_dist* with the minimum of the k candidates' $UBD_{\delta_{wDF}}$s. In the refinement stage, we calculate the exact wDF distance for every candidate and select the k trajectories with the smallest distance values. Again, we could use the triangular inequality of wDF to prune some of the distance computation.
- **Range Join** [5], [17]: A range join finds for each trajectory from $\mathbb{S}_1$ all the trajectories from $\mathbb{S}_2$ that are within a given wDF distance of it. For this extension, we simply need to fix the *upper_bound_dist* to the range query threshold in line 7 of Algorithm 1 and remove line 11-12 for updating the *upper_bound_dist*, i.e., we retrieve for each trajectory in $\mathbb{S}_1$ all the candidates whose $LBD_{\delta_{wDF}}$ is less than the given distance threshold during the filtering stage, and refine the answers using the exact wDF distance.

We also note that our framework can straightforwardly support the time interval join [4], where the kNN or range predicate is defined using only some portions (segments) of trajectories within a specified time interval of interest. In this case we retrieve only the index nodes and leaf node entries that intersect with the given time interval from the same index structure. This can be easily handled by changing line 3 of Algorithm 1 to check whether the two elements from the triple are temporally intersecting with the querying interval. If so we continue with the normal processing procedure, otherwise we simply discard the triple.

5 Experimental Results

In this section, we empirically evaluate the efficiency and effectiveness of our proposed techniques.

We have implemented our similarity join framework in Java. All our experiments are executed on a PC with a Pentium IV 3.0 GHz CPU and 1 GB of memory. To evaluate the efficiency of the proposed algorithms, we use the network-based traffic generator [6] and produce moving object trajectories based on the road networks of Oldenburg (OB) and San Francisco (SF). To obtain some quantitative observations about the potential use of our

framework for data mining applications, we use the *w*DF distance for classification in data sets provided by the UCR Time Series Collection [16]. We index the trajectories with a 3-dimensional R-tree and uniformly split each trajectory into segments. The resulting segments are then inserted into the R-tree, where each data entry in the leaf node contains one segment. The page size is set to 4KB, and an LRU buffer with 1000 pages is used. Unless stated otherwise, the *w* window size is set to 15% of the entire trajectory duration.

5.1 Efficiency of Similarity Join

Although our results are independent of the trajectory splitting strategy adopted, before evaluating the performance of our similarity join framework, we need to determine the trajectory splitting size for our data sets in order to remove its impact from further experiments. Increasing the number of splits implies tighter bounds but may also increase the costs for calculating them, whereas decreasing the number of splits deteriorates filtering effectiveness. We perform a nearest neighbor join for trajectories generated from the road networks of OB and SF, with 400 and 1000 points respectively. We generate 200 trajectories using the road networks and use this set as the first set for the join algorithm, we then add small perturbations to each trajectory, i.e., slightly offset the location of each point in the trajectory, and use the perturbed set as the second set of trajectories for the join algorithm. We vary the number of points contained in each segment from 5 to 200 and the results are shown in Figure 4 (a). Based on the results, we fix the number of points in each segment to 20 in the following experiments.

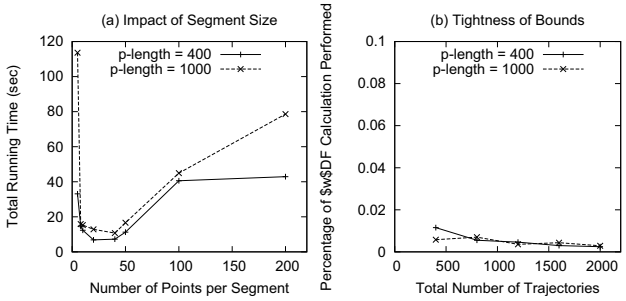

Figure 4: *Impact of Segment Size and Tightness of Bounds*

With the uniform split model, we then evaluate the tightness of the two distance bounds $LBD_{\delta_{wDF}}$ and $UBD_{\delta_{wDF}}$. We use the road networks of OB and SF to generate varying number of trajectories, and randomly pick one trajectory to perform a nearest neighbor query on the data set, using the two distance bounds for pruning. We record the total number of times the exact *w*DF distance is calculated, and divide this number by the total number of trajectories in the data set. The result ratio is shown in Figure 4 (b). Using our approximate distance bounds, we only need to perform less than 2% of the *w*DF distance calculation.

Next we evaluate the efficiency and the scalability of our trajectory join algorithm. Due to the limited space, we focus on the nearest neighbor join only. The next two sets of experiments compare the efficiency of three different approaches: (1) our framework using the *w*DF distance, (2) the metric-based join [26] (essentially a sequential scan over the entire data set but uses the triangular inequality for pruning as much as possible) with *w*DF as the distance metric and (3) similarity join on DTW distance with lower-bound indexing [17], as a representative of the transformed space approach. For the DTW based approach, we implement the join as a batch of similarity search queries where each query is a trajectory from the first data set that is used to search for its nearest neighbor in the second data set. We use the same parameters, e.g., the number of points in each segment/piece-wise approximation, the R-tree parameters and splitting strategy, etc. in both our framework and the DTW implementation. We also take into account the time it takes for approach (1) and (3) to build the index structure.

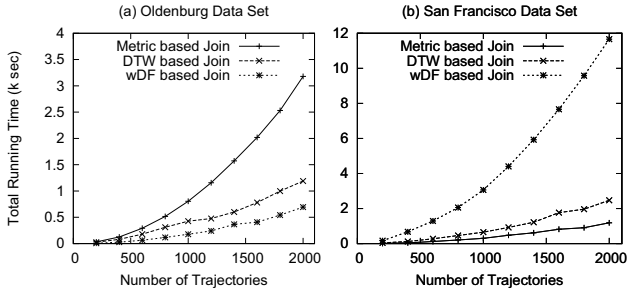

Figure 5: *Scaling with Number of Trajectories*

Our first set of experiments reports the total running time of the nearest neighbor join as a function of the number of trajectories. Figure 5 compares the performance of the three approaches on trajectories generated from road networks of OB and SF, respectively. Each OB trajectory contains 400 points and each SF trajectory contains 1000 points. We observe that our join framework clearly outperforms the metric-based join, yielding a speed-up of up to 10 times. Furthermore, our approach scales well to large trajectory sets since the running time grows linearly with respect to the number of trajectories, whereas the running time for metric-based join grows quadratically. This is because with our index-based join, the number of exact distance calculation grows only linearly with the number of trajectories, the rest of the distance calculation are pruned. On the other hand, metric-based join does not have this property. While the DTW based approach also outperforms the metric space based approach by a large factor, it is on average more than 2 times slower than our approach. This discrepancy becomes even larger on the SF data set. The main reason is that when the number of points in each trajectory increases, the dimensionality of the index

structure used to index the trajectories in the transformed space, i.e., the index of the DTW distance, also grows. This will reduce the selectivity of the index and admit more false positives that will need to be eliminated with the expensive DTW distance calculation. Increasing the number of points per segment/piece-wise approximation can not solve the problem, as it will yield a wider bounding envelop used by DTW and loosen the lower-bounds [17]. Working in the native space, our approach does not have this problem of dimensionality. When the number of points per trajectory increases, it only increases the total number of segments and the size of the R-tree structure. However, the extra accesses to the indices are paid off by the reduction of false positives because of the lower/upper-bounds.

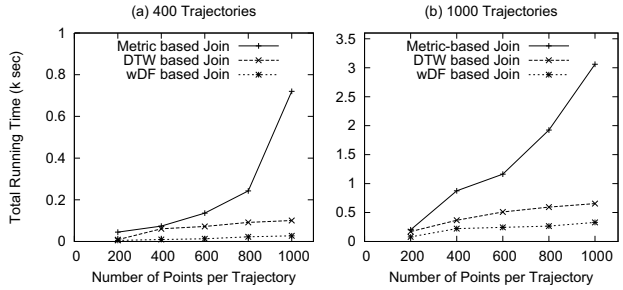

Figure 6: *Scaling with Trajectory Length*

Our next set of experiments investigates the similarity join performance with respect to the number of points per trajectory. We fix the number of trajectories in OB and SF to 400 and 1000 respectively and increase the number of points in each trajectory. From Figure 6, we can observe that our approach scales very well with the number of points per trajectory, and consistently delivers a speedup of more than 2 with respect to the DTW based approach. The speedup increases as the number of points per trajectory grows from 200 to 1000.

5.2 Effectiveness of *w*DF

In order to evaluate the effectiveness of our proposed similarity measure, we use a one-nearest neighbor classification algorithm as suggested by Keogh *et al.* on 20 different data sets [16]. These data sets cover various application domains, e.g., robotics, industry, botany etc. For each group of data, a training set and a testing set are provided together with the correct cluster labels. We compare the classification error ratio of *w*DF against that of L_2-norm and DTW from [16], as shown in Table 1.

The classification error ratio of *w*DF is obtained by finding the optimal warping window size for the purpose of this comparison (and so does DTW), and the percentages in parentheses indicate the ratio of matching window size w to the trajectory duration. We perform an exhaustive search using all possible matching window sizes, and report the one that yield the minimum classification error ratio. We

Data set	L_2-norm	DTW	*w*DF (w)
Synthetic Ctrl.	0.12	0.017	0.02 (13%)
Gun-Point	0.087	0.087	0.027 (0.7%)
CBF	0.148	0.004	0.027 (14.8%)
Face(all)	0.286	0.192	0.142 (2.3%)
OSU Leaf	0.483	0.384	0.421 (3%)
Swedish Leaf	0.213	0.157	0.182 (4.7%)
50 Words	0.369	0.242	0.301 (4.4%)
Trace	0.24	0.01	0 (4.4%)
Two Patterns	0.09	0.0015	0.0045 (14.5%)
Wafer	0.005	0.005	0.0045 (13.2%)
Face(Four)	0.216	0.114	0.307 (2.3%)
Lightning-2	0.246	0.131	0.229 (2.8%)
Lightning-7	0.425	0.288	0.329 (3.8%)
ECG	0.12	0.12	0.13 (1%)
Adiac	0.389	0.391	0.381 (6.25%)
Yoga	0.17	0.155	0.143 (2.1%)
Fish	0.217	0.16	0.181 (1.7%)
Beef	0.467	0.467	0.467 (2.3%)
Coffee	0.25	0.179	0.0714 (2.8%)
OliveOil	0.133	0.167	0.167 (0.5%)

Table 1: *Effectiveness of wDF Distance*

observe that the classification error rates yielded by *w*DF are clearly superior to L_2-norm, and is comparable with DTW (*w*DF wins in 7 data sets, ties in 2 data sets and loses the rest). This is because while *w*DF can handle local time shifting, it is more sensitive to noise than DTW. We note that using a uniform window size of 15% yields only slightly different results [10].

6 Related Work

The problem of turbo-charging data mining process by similarity join has been investigated in [5] for low-dimensional data. In this work, we focus on joining spatio-temporal trajectories and the main goal is to utilize the index structure to prune a large number of expensive distance calculation which dominates the join process. A trajectory join using the L_p-norms and a specialized index structure was presented in [4]. However, the approach can not be straightforwardly extended to support different spatio-temporal similarity join.

In [25] the indexing of LCSS and DTW using MBRs of trajectory segments is explored. However, the proposed lower-bound distance are calculated in conjunction with a query sequence, which makes the efficient extension to similarity join questionable. The issue of what is a semantically appropriate distance measure for trajectory similarity is addressed in [19]. [13] considers similarity search for trajectories using spatio-temporal indices and proposes a novel distance measure, however the work does not address the similarity join of trajectories. We note that the constructing MBRs over time series data for lower bounding has been explored for other similarity measures

along with the idea of early abandoning [17], [23], [27]. In this respect, we applied our *w*DF distance and combine similarity joins and spatio-temporal indices in the native space of moving object trajectories [7], [18], [20]

7 Concluding Remarks & Future Work

In this paper, we introduced a new similarity measure *w*DF for location-related time series data, based on Fréchet distance [1]. In order to compute the distance efficiently, we proposed two approximations for effective upper/lower-bounding. We then combined these approximations with spatio-temporal indices in the native space for pruning, and presented a similarity join framework under our distance measure that supports a number of different similarity join variants. Our experimental results have demonstrated the efficiency and scalability of our proposed technique in the context of moving object trajectories, and verified the effectiveness of our distance measure.

One immediate extension of this paper is to improve the robustness of our distance measure against outliers in the data. Since *w*DF is sensitive to noise, one can alleviate this problem by apply some filtering technique similar to EDR and LCSS [9], [24] when determining *w*DF. We have considered using a median filter to protect the warping matrix from noise. Our preliminary experiments indicate that the median filter substantially improves the effectiveness of *w*DF for classification purposes. However, there are two important issues that we need to address: (1) choosing the optimal filter size, or properly adjusting it (for adaptive algorithms); (2) median filters need not yield metric distance, which may slow down the refinement step of Algorithm 1. We will focus on these issues in the future work. Another interesting avenue of future work is to extend our approach towards more general types of motion and richer representations of the trajectory models.

References

[1] H. Alt and M. Godau. Computing the fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5, 1995.

[2] H. Alt and L. J. Guibas. "discrete geometric shapes: Matching, interpolation, and approximation". *Handbook of Computational Geometry*, 1999.

[3] A. Anagnostopoulos, M. Vlachos, M. Hadjieleftheriou, E. J. Keogh, and P. S. Yu. Global distance-based segmentation of trajectories. In *KDD*, 2006.

[4] P. Bakalov, M. Hadjieleftheriou, E. J. Keogh, and V. J. Tsotras. Efficient trajectory joins using symbolic representations. In *Mobile Data Management*, 2005.

[5] C. Böhm and F. Krebs. The *k*-nearest neighbour join: Turbo charging the kdd process. *Knowl. Inf. Syst.*, 2004.

[6] T. Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2), 2002.

[7] V. P. Chakka, A. Everspaugh, and J. M. Patel. Indexing large trajectory data sets with SETI. In *CIDR*, 2003.

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

[9] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD Conference*, 2005.

[10] H. Ding, G. Trajcevski, and P. Scheuermann. Efficient similarity join of spatio-temporal trajectories. In *Technical Report NWU-EECS-08-01, Northwestern University, http://www.eecs.northwestern.edu/ hdi117/publications.html*, 2007.

[11] T. Eiter and H. Mannila. Computing discrete fréchet distance. In *Technical Report CD-TR 94/64, Technische Universitat Wien*, 1994.

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

[13] E. Frentzos, K. Gratsias, and Y. Theodoridis. Index-based most similar trajectory search. In *ICDE*, 2007.

[14] R. H. Güting, V. T. de Almeida, D. Ansorge, T. Behr, Z. Ding, T. Höse, F. Hoffmann, M. Spiekermann, and U. Telle. "secondo: An extensible dbms platform for research prototyping and teaching". In *ICDE*, 2005.

[15] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *SIGMOD Conference*, 1998.

[16] E. Keogh, X. Xi, L. Wei, and C. Ratanamahatana. The UCR Time Series dataset. In *http://www.cs.ucr.edu/ eamonn/time_series_data/*, 2006.

[17] E. J. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowl. Inf. Syst.*, 7(3), 2005.

[18] Y. Manolopoulos, A. Nanopoulos, A. Papadopoulos, and Y. Theodoridis, editors. *R-trees: Theory and Applications*. Springer-Verlag, 2006.

[19] N. Pelekis, I. Kopanakis, G. Marketos, I. Ntoutsi, G. L. Andrienko, and Y. Theodoridis. Similarity search in trajectory databases. In *TIME*, 2007.

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

[21] Y. Sakurai, M. Yoshikawa, and C. Faloutsos. "ftw: fast similarity search under the time warping distance.". In *PODS*, 2005.

[22] G. Trajcevski, H. Ding, P. Scheuermann, R. Tamassia, and D. Vaccaro. Dynamics-aware similarity of moving objects trajectories. In *GIS*, 2007.

[23] M. Vlachos, D. Gunopulos, and G. Das. Rotation invariant distance measures for trajectories. In *KDD*, 2004.

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

[25] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. J. Keogh. Indexing multidimensional time-series. *VLDB J.*, 15(1), 2006.

[26] J. T.-L. Wang and D. Shasha. Query processing for distance metrics. In *VLDB*, pages 602–613. Morgan Kaufmann, 1990.

[27] L. Wei, E. J. Keogh, H. V. Herle, and A. Mafra-Neto. Atomic wedgie: Efficient query filtering for streaming times series. In *ICDM*, 2005.

[28] O. Wolfson, A. P. Sistla, B. Xu, J. Zhou, and S. Chamberlain. Domino: Databases for moving objects tracking. In *SIGMOD Conference*, 1999.