Indexing and matching trajectories under inconsistent sampling rates

Indexing and Matching Trajectories under Inconsistent Sampling Rates

Sayan Ranu [#1], Deepak P [*2], Aditya D. Telang [*2], Prasad Deshpande [*2], Sriram Raghavan [*2]

[#] *Dept. of CSE, IIT Madras, Chennai, India.*

[*] *IBM Research, Manyata Tech. Park, Bangalore, India.*

[1] sayan@cse.iitm.ac.in

[2] {deepak.s.p, aaditya.telang, prasdesh, sriramraghavan}@in.ibm.com

***Abstract*—Quantifying the similarity between two trajectories is a fundamental operation in analysis of spatio-temporal databases. While a number of distance functions exist, the recent shift in the dynamics of the trajectory generation procedure violates one of their core assumptions; a consistent and uniform sampling rate. In this paper, we formulate a robust distance function called *Edit Distance with Projections (EDwP)* to match trajectories under inconsistent and variable sampling rates through *dynamic interpolation*. This is achieved by deploying the idea of *projections* that goes beyond matching only the sampled points while aligning trajectories. To enable efficient trajectory retrievals using EDwP, we design an index structure called *TrajTree*. TrajTree derives its pruning power by employing the unique combination of bounding boxes with Lipschitz embedding. Extensive experiments on real trajectory databases demonstrate EDwP to be up to 5 times more accurate than the state-of-the-art distance functions. Additionally, TrajTree increases the efficiency of trajectory retrievals by up to an order of magnitude over existing techniques.**

I. Introduction

The last decade has witnessed an unprecedented growth in the availability of location-tracking devices. The widespread usage of these devices generates an abundance of data that are in the form of trajectories. Querying and mining these trajectories is essential for a multitude of spatio-temporal tasks such as tracking migratory patterns of animals [1], identifying "alleys" conducive to environmental disasters [2], etc. What lies at the core of any of these analytical tasks is a mechanism to compute the similarity between two trajectories. The importance of matching trajectories has been recognized in the computer science community and a number of techniques exist [3]–[7]. However, the recent explosion in the volume of trajectory data is fueled by the availability of cheap and heterogeneous location tracking devices that violate an assumption made by all of the existing trajectory matching techniques; the assumption of a uniform and consistent sampling rate. In this paper, we study the problem of matching trajectories in the presence of this sampling noise.

Generally, a trajectory $T = [s_1, \cdots, s_n]$ of a moving object is represented as a sequence of spatio-temporal points s_i that are sampled over a time duration. A spatio-temporal point $s = (x, y, t)$ encodes the spatial-attributes of the location, such as latitude and longitude, and the timestamp t at which the location was traversed. Two trajectories are considered similar if they remain spatially close to each other for the majority of their existence. A basic model to match trajectories is through the L^p-norm and create a one-to-one alignment between the sampled points. This model, however, suffers in cases of *local time shifts*. Consider two trajectories that traverse the same spatial contour, where one of them is slower in the first half of the distance, and the other is slower in the second half. At an uniform sampling rate, the first trajectory would have more points in the first spatial half than the second. Consequently, the true distance would not be captured. Dynamic Time Warping (*DTW*) [6] first recognized this issue and accounted for local time shifts using many-to-one mappings. Subsequently, DTW was further improved by better capturing the spatial semantics in Edit distance with Real Penalty (ERP) [4], Edit Distance on Real sequence (EDR) [5], DISSIM [7], and model-driven assignment (MA) [8].

While existing techniques can adapt to local time shifts, they are unable to cope well with non-uniform sampling rates. In real life, there are variations in sampling rates within and across trajectories induced through variable device settings, power constraints, intermittent signal disruptions, etc. For example, a recent study has shown that cab drivers alter the default sampling rate in their GPS-navigation systems to reduce power consumption [9]. Furthermore, sampling rates in trajectories generated from sources such as online "check-ins" (eg. FourSquare), gps-tagged photo albums, and call detail records are inherently non-uniform. These variations in inter-trajectory as well as intra-trajectory sampling rates pose a number of unique challenges that are not yet adequately addressed. To combat these issues, we make the following contributions in trajectory matching:

- We develop a robust distance function, called *Edit Distance with Projections (EDwP)*, to compute similarity between trajectories. EDwP employs a threshold-free approach and automatically adapts to non-uniform sampling rates through *dynamic interpolation* by using the idea of *projections*.
- We develop an index structure called *TrajTree* for efficient retrieval of k-NN queries. The proposed index structure combines the ideas of *Lipschitz embedding* [10] with *bounding boxes* [11] to drastically prune the search space.
- Extensive experiments performed on real datasets show EDwP as up to 5 times more accurate and robust than state-of-the-art trajectory matching techniques. Furthermore, TrajTree is efficient in pruning the search space, which results in an order of magnitude speed-up over the most recent trajectory matching technique [8] and 5X speedup over EDR [5].

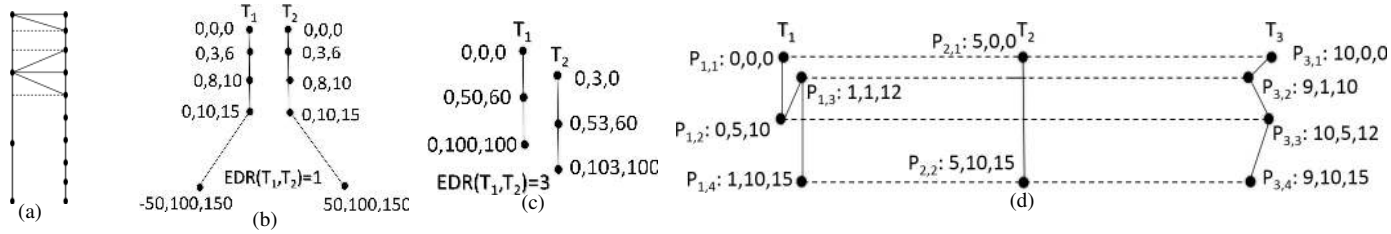


Fig. 1. Illustrates the weaknesses of existing techniques using EDR [5] as the representative metric against (a) inter-trajectory sampling rate variance, (b) intra-trajectory sampling rate variance and (c) "phase" shifts. The first two dimensions in each st-point represents the spatial location, and the third dimension represents the timestamp at which the location is recorded. The distances computed using EDR assumes a spatial threshold of $\epsilon = 2$. (d) Illustrates a scenario where the mapping produced by MA [8] is not logical.

II. Impact of inconsistent sampling rates

In this section, we analyze the impact of varying sampling rates on existing trajectory matching techniques.

1. Inter-trajectory Sampling Rate Variations: Consider Fig. 1(a) where the two trajectories represented by the vertical lines are being compared; the left one is sparsely sampled (only 4 points) whereas, the right one has a higher rate of sampling. Since most existing techniques (except DISSIM [7] and MA [8]) start by mapping sampled points between the trajectories, the best mappings include those as indicated by the slanted solid lines joining points across trajectories, which cause a larger spatial distance between mapped points than the more intuitive mappings indicated by the horizontal dotted lines.

MA [8] is more flexible in aligning points. While trying to align a sampled point p_1 in trajectory T_1 to another sampled point p_2 in T_2, it considers any non-sampled point in the straight line connecting p_2 to the last aligned point in T_2 prior to p_2. Fig. 1(d) illustrates an example. This technique, however, may produce alignments that are semantically inconsistent. Specifically, in Fig. 1(d), $p_{1,2}$ and $p_{1,3}$ in T_1 get mapped to two non-sampled points in T_2. Here, $p_{1,3}$ is mapped to a point that is traversed prior to the point where $p_{1,2}$ is mapped. In other words, it introduces alignments that goes backward in time, and thus, violates the basic premise of time-series matching. Consider another trajectory T_3 in the figure that is formed by points that are as far away from T_2 as those of T_1; the difference in the order of traversing points among T_1 and T_3 intuitively makes T_3 much more similar to T_2 than T_1. However, MA relies on aggregating the distance between matched points, and thus spuriously assigns the same similarity for the (T_1, T_2) pair as for (T_2, T_3).

DISSIM also incorporates non-sampled regions in the distance computation. However, DISSIM cannot cope with local time shifts due to only considering one-to-one mappings. Thus, DISSIM can detect similarity between trajectories under non-uniform sampling rates only if they travel at identical speeds.

2. Intra-trajectory Sampling Rate Variations: Besides inter-trajectory sampling rate variation, sampling rates also vary within a trajectory. This results from signal disruptions, power constraints, etc. Fig. 1(b) illustrates the issue by taking one of the most recent works, EDR, as the representative metric. EDR considers two points to "match" if they are within a spatial distance threshold ϵ. Let us assume $\epsilon = 2$. As can be seen, four out of the five points are identical and accordingly, EDR assigns a distance of 1 due to the dissimilarity between the fifth points. However, all of the four matched points represent a densely sampled region and in reality, for the majority of the two trajectories, they diverge from each other. As a result, a distance of 1 is not an accurate reflection of the significant dissimilarity existing between them.

One could certainly tackle varying sampling rates by interpolating points in the under-sampled regions. Such an approach however, suffers from three bottlenecks. First, to ensure a uniform sampling rate, interpolation should be performed such that the processed database of trajectories have a uniform density that is equal to the maximum density observed in the unprocessed

If the trajectories are constrained within a road network, one could consider map-matching [12], [13]. However, it has been shown that at low sampling rates, map-matching is not accurate [14]. More critically, many trajectories such as animal migrations, ship and airplane movements, pedestrian trajectories, etc. are not constrained within a network.

3. Sampling "Phase" Variations: Since st-points only represent a sample of the locations traversed, the choice of recorded samples can have a drastic impact on the computed distance even under uniform sampling. Fig. 1(c) depicts two trajectories that are sampled uniformly from their starting points. Although T_1 and T_2 overlap for the majority of their existence in the spatial region between $(0, 3)$ and $(0, 100)$, they are not represented by the same set of sampled points. As a result, under a spatial threshold of $\epsilon = 2$ for EDR (or LCSS), none of the st-points match, and consequently, the maximum possible distance of 3 is assigned.

4. Threshold Dependency: EDR, ERP and LCSS, use thresholds to determine similarity between locations. MA depends on four different thresholds. Generally, threshold-dependent techniques grid the physical space into regions, and two spatial locations "match" if they fall within the same grid. Determining the appropriate gridding threshold, however, is not straightforward and can have a large impact on the similarity. For example, in Fig. 1(c), at $\epsilon = 2$, EDR(T_1, T_2) assigns the maximum possible distance of 3. On the other hand, a small increase of $\epsilon = 3$ reduces EDR(T_1, T_2) to the lowest possible distance 0.

Table I summarizes the features of six of the most recent distance metrics. In this paper, our goal is to answer all of the outlined challenges without compromising on the positive aspects of any of the existing distance functions.

TABLE I. ROBUSTNESS OF PREVIOUS TRAJECTORY SIMILARITY METRICS

Technique	Local Time Shifts	Sampling Rate Variations			Threshold Free?
		Inter--trajectory	Intra-trajectory	Phase Variations	
DTW [6]	✓				✓
LCSS [3]	✓				
ERP [4]	✓				
EDR [5]	✓				
DISSIM [7]		✓			✓
MA [8]				✓	
EDwP	✓	✓	✓	✓	✓

III. PROBLEM FORMULATION

***Definition** 1:* TRAJECTORY: *A trajectory* $T = \{s_1, \cdots, s_n\}$ *is a temporally ordered sequence of spatio-temporal points (st-points). An st-point* $s = ([v_1, \cdots, v_d], t)$ *contains a* d*-dimensional feature vector describing the spatial attributes and a timestamp* t *encoding the time at which the location is recorded.*

For simplicity, we use trajectories embedded in a 2D plane. The first two dimensions in each st-point denotes the spatial co-ordinates and the third dimension denotes the timestamp.

***Definition** 2:* SUB-TRAJECTORY: T_1 *is a sub-trajectory of* T_2 *if* $\forall i, 1 \leq i \leq |T_1|\ T_1.s_i = T_2.s_{a+i}$*, where* $\exists\ a,\ 0 \leq a \leq (|T_2| - |T_1|)$*. The relationship is denoted using* $T_1 \subseteq T_2$*.*

To denote the specific sequence of st-points that define a sub-trajectory T_1 of T_2, we use the notation $T_1 = T_2[a, .., b]$.

***Definition** 3:* SPATIO-TEMPORAL SEGMENT: *A spatio-temporal segment* $e = [s_1, s_2, f(\cdot)]$ *represents a segment connecting two temporally consecutive st-points* s_1 *and* s_2 *through an interpolating function* $f(\cdot)$*.* $f(\cdot)$ *models the movement of the object in the intermediate time interval* s_{1t} *to* s_{2t}*.*

We use $e.s_1$ and $e.s_2$ to denote the two endpoints of e respectively. $s \in e$ denotes an st-point s lying within st-segment e. Spatio-temporal segments (*st-segments*) are better descriptors of trajectories, since they characterize the entire trajectory shape under a given set of observations. We assume $f(\cdot)$ to be a straight line connecting s_1 and s_2 and thus represent an st-segment as $e = [s_1, s_2]$. Certainly, more accurate functions can be used to interpolate points in the presence of additional information such as the underlying road network [15]. We choose linear since it approximates movements well [7], [16], [17].

With the introduction of an st-segment, hereon, a trajectory is represented as a sequence of segments rather than points. The notion of a sub-trajectory is extended analogously. The *length* of a trajectory T is defined as follows:

$$length(T) = \sum_{\forall e_i \in T} length(e_i) \tag{1}$$

where $length(e_i) = dist(e_i.s_1, e_i.s_2)$. The *speed* within e is defined as $speed(e) = \frac{length(e)}{e.s_{2_t} - e.s_{1_t}}$.

A. Edit Distance with Projections (EDwP)

Conceptually, given two trajectories T_1 and T_2 that are represented as segment sequences, EDwP computes the cheapest set of *edits* that make them identical. EDwP performs two kinds of edits: *replacements* and *inserts*.

Replacement: The replacement operation, denoted by $rep(e_1, e_2)$, represents the operation where the segment e_1 is matched with e_2. This match adds a cost, defined as follows:

$$rep(e_1, e_2) = dist(e_1.s_1, e_2.s_1) + dist(e_1.s_2, e_2.s_2) \tag{2}$$

where $dist(s_1, s_2)$ denotes the euclidean distance between points s_1 and s_2. $rep(e_1, e_2)$ is symmetric and the cost is proportional to the spatial distance between the endpoints of e_1 and e_2. Thus, if e_1 and e_2 are identical, the cost would rightly evaluate to zero.

Insert: While $rep(.,.)$ allows us to match segments, matching trajectories based only on already sampled points (and such segments) is inadequate due to reasons outlined in Sec. II. The $ins(e_1, e_2)$ operation introduces extra points to aid robust matching. In particular, $ins(e_1, e_2)$ denotes the operation that inserts a point $p^{ins(e_1,e_2.s_2)}$ into e_1, effectively splitting e_1 into two segments $[e_1.s_1, p^{ins(e_1,e_2.s_2)}]$ and $[p^{ins(e_1,e_2.s_2)}, e_1.s_2]$. The point $p^{ins(e_1,e_2.s_2)}$ is determined so that the first segment $[e_1.s_1, p^{ins(e_1,e_2.s_2)}]$ is best aligned with e_2; thus, $p^{ins(e_1,e_2.s_2)}$ is effectively the point on e_1 that is spatially closest to $e_2.s_2$:

$$p^{ins(e_1,e_2.s_2)} = \arg\min_{p \in e_1} \quad dist(p, e_2.s_2)$$

Given the construction, we refer to $p^{ins(e_1,e_2.s_2)}$ as the *projection* of $e_2.s_2$ on to e_1. The timestamp for the new spatial point is intuitively in proportion with the partition that it induces within e_1:

$$p_t^{ins(e_1,e_2.s_2)} = e.s_{1t} + \frac{dist(e.s_1, p^{ins(e_1,e_2.s_2)})}{speed(e_1)}$$

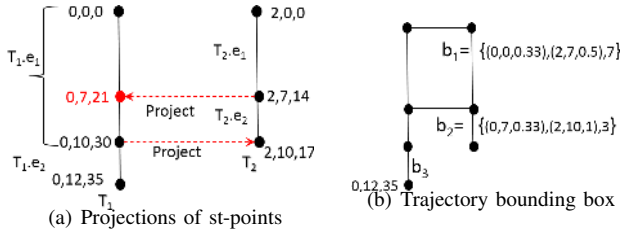
The $ins(e_1, e_2)$ operation does not involve any cost; the first part of the split segment of e_1 is expected to be matched to e_2 in the next step, wherein the cost would be incurred.

From the perspective of trajectories, the operation $ins(T_1.e_1, T_2.e_1)$ modifies T_1 by inducing the split on $T_1.e_1$ as outlined above. Following this operation, T_1 has one additional segment, where the first two segments are the partitions of $T_1.e_1$ from the split followed by segments $T_1.e_2$ to $T_1.e_{|T_1|}$. Hereon, we overload the $ins(.,.)$ notation to use trajectories, so that $ins(T_1, T_2)$ is meant to denote exactly the same split as $ins(T_1.e_1, T_2.e_1)$. Thus, $ins(T_1, T_2)$ denotes the insertion defined on the first segments of the trajectories.

***Example** 1: In Fig. 2(a), the* $ins(T_1, T_2)$ *splits* $T_1.e_1$ *by inserting the new point* $(0, 7, 21)$ *within* T_1*, as illustrated in the figure. Let* T_1' *denote the modified version of* T_1 *after the insert operation (i.e., the output of* $ins(T_1, T_2)$*); the following step would then invoke* $rep(T_1'.e_1, T_2.e_1)$ *so that the segments* $[(0, 0, 0), (0, 7, 21)]$ *and* $[(2, 0, 0), (2, 7, 14)]$ *be matched incurring the cost:*

$$dist((0,0,0),(2,0,0)) + dist((0,7,21),(2,7,14)) = 2 + 2 = 4$$

With the formalization of the edit operations, we next define the proposed distance function EDwP between trajectories



(a) Projections of st-points (b) Trajectory bounding box

Fig. 2. (a). Illustrates the idea of *projections*. (b). The trajectory bounding box constructed over T_1 and T_2 in Fig. 2(a). The tBoxSeq label for b_3 is omitted since it contains just one segment.

T_1 and T_2.

$$\text{EDwP}(T_1, T_2) = \begin{cases} 0 & \text{if } |T_1| = |T_2| = 0 \\ \infty & \text{else if } |T_1| = 0 \text{ or } |T_2| = 0 \\ min\{\text{EDwP}(Rest(T_1), Rest(T_2)) + & \text{otherwise} \\ \quad (rep(T_1.e_1, T_2.e_1) \times Coverage(T_1.e_1, T_2.e_1)), \\ \quad \text{EDwP}(ins(T_1, T_2), T_2), \\ \quad \text{EDwP}(T_1, ins(T_2, T_1)),\} \end{cases}$$

where $Rest(T)$ is the sub-trajectory $T[2, \cdots, |T|]$ containing all segments except $T.e_1$. *Coverage* quantifies the importance of an edit based on how representative the segments being edited are of the overall trajectories.

$$Coverage(e_1, e_2) = length(e_1) + length(e_2) \quad (3)$$

Thus, larger segments have more weight on the overall distance than smaller segments.

***Example** 2: The cheapest sequence of edits to convert T_1 to T_2 in Fig. 2(a) is illustrated in Fig. 3 where the different steps are represented in the sub-figures. Adding the cost of each edit, EDwP$(T_1, T_2) = 89.65$.*

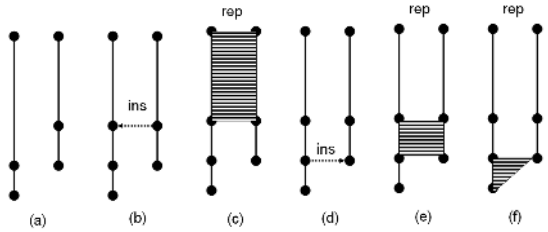


Fig. 3. Illustration of EDwP while matching the trajectories in Fig. 2(a).

Since EDwP computes the cumulative sum of each edit, in most cases, the distance is likely to increase monotonically with the length of the trajectories (unless the extra length improves the alignment, as could happen in certain cases). In certain situations, the average distance between matched points is more desirable than the cumulative distance. For such cases, we length normalize EDwP.

$$\text{EDwP}_{avg}(T_1, T_2) = \frac{\text{EDwP}(T_1, T_2)}{length(T_1) + length(T_2)} \quad (4)$$

Table II summarizes the novel features of EDwP that enables it to overcome all of the weaknesses of existing techniques without compromising on any of their positive aspects. At the same time, similar to the quadratic computation costs of DTW, LCSS, ERP, and EDR, the cost of EDwP(T_1, T_2) is $O\left((|T_1| + |T_2|)^2\right)$.

TABLE II. FEATURES OF EDWP THAT SOLVE ISSUES OUTLINED IN TABLE I.

Local Time Shifts	Sampling Rate Variations			Threshold Dependence
	Inter--trajectory	Intra-trajectory	Phase Variations	
Dynamic Programming	Projections	Coverage	Segment alignment	Parameter-free

***Example** 3: To establish how EDwP improves on the deficiencies of existing techniques, let us revisit Figs. 1(b) and 1(c). Here, EDwP(T_1, T_2) corresponding to Fig. 1(b) and Fig. 1(c) are 20591.26 and 582 respectively. Compared to EDR, EDwP correctly identifies that the trajectories in Fig. 1(c) are significantly more similar than those in Fig. 1(b).*

IV. INDEX STRUCTURE

Due to the quadratic computation cost of EDwP, performing sequential scans across entire databases is not scalable. Thus, we develop an index structure called *TrajTree* to answer k-NN queries efficiently on large trajectory databases.

Like LCSS, DTW and EDR, EDwP is non-metric due to violating triangular inequality.

***Theorem** 1: EDwP does not satisfy triangular inequality.*

PROOF: See Sec. A in Appendix.

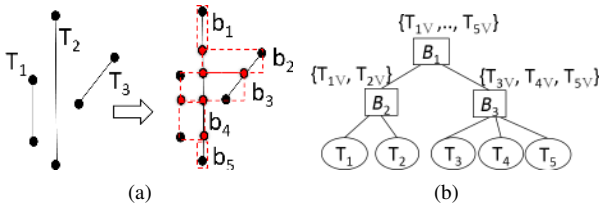
Due to Theorem 1, generic indexing techniques that are reliant on triangular inequality based pruning cannot be applied. Thus, we formulate the concept of *bounding boxes* for trajectories and organize the search space in a hierarchical fashion.

A. Indexing trajectories using bounding boxes

Generally, a bounding box, such as in R-trees [11], summarizes a set of objects such that given a query, its distance to the bounding box is lower than the distances to all of the constituent objects within the box. Generalizing this idea to trajectories, however, is not straightforward. A number of questions arise.

• *How do you construct a tight bounding box on a set of trajectories?* The most intuitive approach is to describe a set of trajectories as a sequence of bounding boxes. An example is shown in Fig. 2(b). While the bounding box for trajectories in Fig. 2(a) is straight-forward, the scenario in Fig. 4(a) is much more complex. As can be seen, to describe all three trajectories using a sequence of tight bounding boxes, we need to partition each trajectory into a large number of segments. Otherwise, the tightness is compromised. Generating the optimal partitioning scheme is difficult, since they are not based on any global alignment. Rather, the analysis needs to happen in the sub-trajectory space, where the optimal partitioning scheme allows us to identify sub-trajectories that are in close spatial proximity and thereby, allowing summarization through a sequence of tight bounding boxes.

• *Given a set of bounding boxes that have already been built, where should a new trajectory be inserted?* Ideally,



(a) (b)

Fig. 4. (a) The ideal bounding box on the shown three trajectories. Red points denote projected points and the red dashed boxes denote the bounding boxes. (b) Demonstrates the structure of TrajTree. The non-leaf square nodes represent tBoxSeqs, and the circular leaf nodes represent trajectories. T_{i_V} represents the vantage descriptors stored at each node.

it should be inserted on the bounding box that undergoes minimum expansion in volume.

• *How can bounding boxes be used to compute a lower bound?* In the space of high-dimensional points, this operation is trivial since comparing to the corners of the box is enough. The same strategy does not transfer in the space of trajectories. Additionally, computing the EDwP between a query trajectory and a bounding box does not provide a lower bound.

Existing literature do not answer the above questions, which necessitates a deeper analysis for a formal treatment of the above questions. We begin this analysis by introducing the following definitions.

***Definition** 4:* SPATIO-TEMPORAL BOX: *A spatio-temporal box (st-box) $b = (s_1, s_2, minL)$ is a bounding box constructed over a set of st-segments. The st-points s_1 and s_2 represent the spatial coordinates of the diagonals of b. minL denotes the minimum length of all segments enclosed in b.*

We use the notations $s \in b$, and $e \in b$ to denote that an st-point s or st-segment e is bounded within b.

***Definition** 5:* TRAJECTORY BOX SEQUENCE: *A trajectory box sequence (tBoxSeq) is a sequence of st-boxes. The operation of constructing a tBoxSeq over a set of trajectories $\mathbb{T} = \{T_1, .., T_n\}$ is denoted as $\mathcal{B}$ =tBoxSeq($\mathbb{T}$). The volume $Vol(\mathcal{B})$ of $\mathcal{B}$ is $\sum_i^{|\mathcal{B}|} Vol(\mathcal{B}.b_i)$. In 2D, $Vol(\mathcal{B}.b_i)$ is simply the area of the bounding box $\mathcal{B}.b_i$.*

An example of a tBoxSeq is shown in Fig. 2(b). Next, we extend the existing definitions and notations to tBoxSeqs.

- **Sub-trajectory:** Trajectory $T \subseteq \mathcal{B}$, if $\forall i, 1 \leq i \leq |T|$, $T.e_i \in \mathcal{B}.b_{a+i}$ for some $0 \leq a \leq (|\mathcal{B}| - |T|)$.
- **dist(s,b)**: The distance between an st-point s and an st-box b is $\min_{\forall p \in b}\{dist(s,p)\}$.
  Projection: The projection of a point s on an st-box b is defined as $p^{ins(b,s)} = \arg\min_{p \in b} dist(s,p)$. The reverse projection $p^{ins(e,b)} = \arg\min_{p \in b} dist(s,p) \forall s \in e$ of b on a segment e is defined analogously.
- **Replace and Insert:** By incorporating the generalized formulations of projection and $dist(\cdot,\cdot)$, replace and insert edits are computed accordingly.
- **Coverage:** $Coverage(T.e, \mathcal{B}.b) = length(e) + length(b.minL)$, where T is a trajectory and $\mathcal{B}$ is a tBoxSeq.

All existing notations such as $Rest(\mathcal{B})$, $\mathcal{B}[m, \cdots, n]$, etc. are extended analogously.

B. Constructing tBoxSeqs

To construct tBoxSeqs, the optimal partitioning scheme needs to be computed in the sub-trajectory space. Towards that goal, we define a *sub-trajectory distance function*, EDwP$_{sub}$. The goal in EDwP$_{sub}(T_1, T_2)$ is to identify the sub-trajectory in T_2 that is most similar to T_1. As in the global alignment using EDwP, EDwP$_{sub}$ uses the same edit operations of replace and insert to partition and align trajectories. However, the edits are used in a different manner to identify the most similar sub-trajectory rather than the optimal global alignment. Using the generalized formulations above, the arguments in EDwP$_{sub}$ can either be a trajectory or tBoxSeq. Formally, EDwP$_{sub}$ is defined as follows.

$$\text{PrefixDist}(T, \mathcal{B}) = \tag{5}$$
$$\begin{cases} 0 & \text{if } |T| = 0 \\ \infty & \text{if } |\mathcal{B}| = 0 \\ min\{\text{PrefixDist}(Rest(T), Rest(\mathcal{B})) + & \text{otherwise} \\ \quad (rep(T.e_1, \mathcal{B}.b_1) \times Coverage(T.e_1, \mathcal{B}.b_1),) & \\ \quad \text{PrefixDist}(ins(T, \mathcal{B}), \mathcal{B}) & \\ \quad \text{PrefixDist}(T, ins(\mathcal{B}, T))\} & \end{cases}$$

$$\text{EDwP}_{sub}(T, \mathcal{B}) = \min_{1 \leq i \leq |\mathcal{B}|}\{PrefixDist(T, \mathcal{B}[i, .., |\mathcal{B}|])\} \tag{6}$$

where T is a trajectory and $\mathcal{B}$ is a tBoxSeq. Note that the only differences between EDwP and PrefixDist are the initialization conditions for $|T| = 0$ and $|\mathcal{B}| = 0$. This modification allows PrefixDist to skip suffixes of $\mathcal{B}$ without incurring any penalty. Consequently, PrefixDist computes the prefix of $\mathcal{B}$ that best matches to T. Since our goal is to compute the best-matching sub-trajectory (or a sub-sequence of a tBoxSeq), along with skipping suffixes, we should also be able to skip prefixes. Skipping prefixes is achieved by the $min\{\ldots\}$ condition in the EDwP$_{sub}(T, \mathcal{B})$ equation; EDwP$_{sub}(T, \mathcal{B})$ explicitly considers suffixes of $\mathcal{B}$ as candidates for T to match with using PrefixDist. *Thus, overall, the proposed systematic combination allows for skipping any prefix as well as any suffix of $\mathcal{B}$, which allows the possibility of aligning T with any contiguous sub-sequence of $\mathcal{B}$.* Clearly, EDwP$_{sub}(T, \mathcal{B})$ is asymmetric. The computation cost of EDwP$_{sub}$ is same as EDwP, which is $O\left((|T| + |\mathcal{B}|)^2\right)$. Even though PrefixDist needs to be computed for each suffix of $\mathcal{B}$, by employing dynamic programming, the results for a suffix S can be reused while computing the distance for a larger suffix $S' \supseteq S$.

Equipped with EDwP$_{sub}$, to compute the tBoxSeq on two trajectories, first, the optimal subsequence alignment is identified, and then an st-box is computed corresponding to each replace operation. To generalize the operation over a set of trajectories $\mathbb{T} = \{T_1, \cdots, T_{|\mathbb{T}|}\}$, the following iterative procedure is followed.

1) Initialize tBoxSeq, $\mathcal{B} = createTBoxSeq(T_1)$
2) $\forall\ i, 2 \leq i \leq |\mathbb{T}|$
   a) $\mathcal{B} = createTBoxSeq(T_i, \mathcal{B})$

where the $createTBoxSeq(T_i, \mathcal{B})$ represents a tBoxSeq based on the EDwP$_{sub}(T_i, \mathcal{B})$ alignment, and creating an st-box for each aligned segment. The final $\mathcal{B}$ is the desired tBoxSeq over all trajectories in $\mathbb{T}$.

***Example** 4: Revisiting Fig. 2(a), $EDwP_{sub}(T_1, T_2) = 89.65$ and $EDwP_{sub}(T_2, T_1) = 80$. Fig. 2(b) demonstrates the tBoxSeq corresponding to the trajectories in Fig. 2(a). Although $EDwP_{sub}$ is not symmetric, the tBoxSeq constructed is identical in this example regardless of the order. For $EDwP_{sub}(T_2, T_1)$, first $ins(T_2.e_1, T_1.e_1)$ inserts an additional point in T_1. Let us denote this partitioned T_1 as T_1'. Following this, $rep(T_1'.e_1, T_2.e_1)$ and $rep(T_1'.e_2, T_2.e_2)$ are the two edits. $T_1.e_2$ is left unmatched.*

C. Computing lower bound from tBoxSeq

For tBoxSeqs to be useful in pruning the search space, we need to show that a lower bound on the distances to the constituent trajectories can be derived. It is easy to see that EDwP$(T, \mathcal{B}) \leq$ EDw$P(T, T_i) \forall T_i \in \mathcal{B}$, where $\mathcal{B}$ is a tBoxSeq. This follows from the fact that EDwP computes a global alignment. Thus, we focus on EDwP$_{sub}$ and re-use it for computing a lower bound as well. Specifically, we prove the following theorem.

***Theorem** 2: Given a set of trajectories $\mathbb{T} = \{T_1, .., T_n\}$ and a query trajectory Q, let $\mathcal{B} = tBoxSeq(\mathbb{T})$. We claim $EDwP_{sub}(Q, \mathcal{B}) \leq EDwP(Q, T)\ \forall\ T \in \mathbb{T}$.*

PROOF: We first establish the following lemma.

***Lemma** 1: Let $T_s \subseteq T$ be two trajectories where $T_s = T[1, .., n]$, $1 \leq n \leq |T|$, In other words, T_s is a prefix of T. Then, for any trajectory Q,*

$$\text{PrefixDist}(Q, T) \leq \text{EDwP}(Q, T_s)\ \forall T_s \tag{7}$$

PROOF BY INDUCTION:
Base Case: Consider, $|T| - |T_s| = 0$, i.e., $T_s = T$
It is easy to see that PrefixDist$(Q, T) \leq$ EDwP(Q, T_s) since all computations in PrefixDist(Q, T), and EDwP(Q, T_s) are identical except when $Q = \emptyset$ and $T_s = T \neq \emptyset$. In this case, PrefixDist$(Q, T) = 0$ and EDwP$(Q, T_s) = \infty$. Thus, PrefixDist$(Q, T) \leq$ EDwP(Q, T_s).
Induction Step: Assume, $\forall Q$, PrefixDist$(Q, T) \leq$ EDwP(Q, T_s), where $|T| - |T_s| = k$. We need to now show that PrefixDist$(Q, T) \leq$ EDwP(Q, T_s) holds for $|T| - |T_s| = k + 1$. Let,

$$T_{ss} \subseteq T \text{ be } T_{ss} = T[1, .., (|T| - 1)] \tag{8}$$

Thus,

$$|T_{ss}| - |T_s| = k \tag{9}$$

Now, let $\mathbb{E}$ be the optimal set of edit operations for PrefixDist(Q, T_{ss}). From induction hypotheses,

$$\text{PrefixDist}(Q, T_{ss}) \leq \text{EDwP}(Q, T_s) \tag{10}$$

Among all sequences of edits that will be explored in PrefixDist(Q, T), one of them will involve $\mathbb{E}$ followed by the optimal edits for PrefixDist $(\emptyset, T[|T|, .., |T|]) = 0$. This follows from the fact that to optimally align T, its immediate sub-trajectory T_{ss} needs to be aligned first, followed by the optimal alignment for the remaining portions. Thus, there exists a sequence of edits where

$$\text{PrefixDist}(Q, T) \leq \text{EDwP}(Q, T_s) \qquad \square$$

***Lemma** 2: For any two trajectories T_1 and T_2, $EDwP_{sub}(T_1, T_2) \leq EDwP(T_1, T_s)\ \forall\ T_s \subseteq T_2$.*

PROOF BY CONTRADICTION: Assume there is a sub-trajectory $T_s = T_2[a, .., b]$ such that EDwP$_{sub}(T_1, T_2) >$ EDwP(T_1, T_s). Thus, from the definition of PrefixDist,

$$\exists T_s' \supseteq T_s,\ \text{PrefixDist}(T_1, T_s') > \text{EDwP}(T_1, T_s) \tag{11}$$

where $T_s' = T_2[a, .., |T_2|]$. However, using Lemma 1, this result contradicts the base assumption. Thus, proved. $\square$

It is easy to see that Lemma 2 also extends to tBoxSeqs.

***Corollary** 1: For any trajectory T and tBoxSeq $\mathcal{B}$, $EDwP_{sub}(T, \mathcal{B}) \leq EDwP(T, T_s)\ \forall\ T_s \subseteq \mathcal{B}$.*

Corollary 1, however, is not enough to prove Theorem 2. More specifically, $tBoxSeq(\mathbb{T})$ does not guarantee that $\forall T \in \mathbb{T},\ T \subseteq tBoxSeq(\mathbb{T})$. To illustrate the issue, consider Figs. 2(a) and 2(b). As can be seen, $T_1 \not\subseteq \mathcal{B}$ since $T_1.e_1 \notin \mathcal{B}.b_1$. Although the shape of T_1 has been captured in $\mathcal{B}$, $\mathcal{B}$ divides T_1 into a higher number of partitions. Thus, we next proceed towards proving the following Lemma.

***Lemma** 3: Consider two trajectories T_1 and T_2 such that both contain one st-segment each. If a new trajectory $T_2' = [[T_2.e_1.s_1, p],\ [p, T_2.e_1.s_2]]$ is created by inserting an additional point p on $T_2.e_1$, then $EDwP(T_1, T_2') \leq EDwP(T_1, T_2)$.*

PROOF: See Sec. B in Appendix.
Lemma 3 can easily be extended to the following corollary.

***Corollary** 2: Given three trajectories T_1, T_2 and T_2', where T_2' is generated by inserting any arbitrary number of points in arbitrary st-segments $e \in T_2$, $EDwP(T_1, T_2') \leq EDwP(T_1, T_2)$.*

By combining Corollaries 1 and 2, Theorem 2 is proved. $\square$

D. Building the index structure

The structure of TrajTree is similar to existing tree-based spatial access methods such as R-trees [11]. More specifically, a tree is built, where the root represents the tBoxSeq over the entire database. Next, trajectories in the root are partitioned into bf different groups, and the tBoxSeq on each group forms a child node of the root. The process is repeated recursively till a node is reached that contains less than n trajectories. Thus, each non-leaf node represents a tBoxSeq, and leaves correspond to trajectories. The structure of the tree is shown in Fig. 4(b). There are two parameters: the branching factor bf and the minimum size of each group n. In this paper, we focus on an in-memory setting and n does not have a significant impact on the performance. For an on-disk implementation, existing techniques based on the disk page size can be used to optimize n. bf, on the other hand, drastically impacts the performance of the index structure. If bf is set too low, the lower bounds computed using Theorem 2 would be loose. On the other hand, for a large bf, the lower bounds would be tight, but more computations would be performed at each level of the tree. It is therefore critical to obtain the right balance between the branching factor and the tightness of the lower bounds. Towards that goal, we use Alg. 1 to partition trajectories at each node.

We build a subset of trajectories $\mathbb{P} \subset \mathbb{D}$, where $\mathbb{P}$ comprises of a set of *diverse* trajectories. $\mathbb{P}$ is initialized to a random trajectory from $\mathbb{D}$ (line 3). Next, we add trajectories to $\mathbb{P}$ by choosing the database trajectory that is most diverse from all trajectories in $\mathbb{P}$ (the max of min construction in line 5). The

Algorithm 1 Partition($\mathbb{D}, \theta, n$)

1: **if** $|\mathbb{D}| \leq n$ **then**
2: **return**
3: $\mathbb{P} \leftarrow \{T_1\}$, $T_1 \in \mathbb{D}$ is randomly selected
4: **repeat**
5: $T \leftarrow \arg\max_{\forall T' \in \mathbb{D}} \{min_{\forall T'' \in \mathbb{P}} \{\text{EDwP}_{sub}(T', T'')\}\}$
6: $drop \leftarrow 1.0 - \frac{min_{\forall T' \in \mathbb{P}}\{\text{EDwP}_{sub} \cdot \mathcal{S}(T,T')\}}{min_{\forall T', T'' \in \mathbb{P}, T' \neq T''} \text{EDwP}_{sub}(T', T'')}$
7: $\mathbb{P} \leftarrow \mathbb{P} \cup \{T\}$
8: **until** $drop > \theta$
9: $\mathbb{B} = \{\mathcal{B}_1, \cdots, \mathcal{B}_{|\mathbb{P}|} | \mathcal{B}_i = \text{tBoxSeq}(T_i),\ T_i \in \mathbb{P}\}$
10: **for** $\forall T \in \mathbb{D} \backslash \mathbb{P}$ **do**
11: $\mathcal{B} \leftarrow \arg\min_{\forall B_i \in \mathbb{B}} \{\text{Vol}(\text{tBoxSeq}(\{\mathcal{B}_i, T\})) - \text{Vol}(\mathcal{B}_i)\}$
12: $\mathcal{B} \leftarrow \text{tBoxSeq}(\mathcal{B} \cup \{T\})$

diversity within $\mathbb{P}$ is quantified using the minimum distance between any pair of its constituent trajectories (denominator in RHS of line 6), and we keep expanding $\mathbb{P}$ as long as the marginal fractional drop in its diversity (*drop* in line 6) is under a threshold θ. Each trajectory in $\mathbb{P}$ now acts as a *pivot* and is initialized to a tBoxSeq. Hereon, each of the remaining trajectories in $\mathbb{D}$ is added to the tBoxSeq that undergoes the minimum expansion in volume (lines 9-12). Thus, instead of bf, we use θ to dynamically adjust the branching factor based on the properties of the dataset.

E. Vantage Points

In this section, we further boost the pruning power of TrajTree by distributing a set of *vantage points (VP)*. By consolidating the unique viewpoints of the VPs, a *vantage descriptor* is generated for each trajectory, which is then used to compute an upper bound on the maximum possible distance between the query and any trajectory in the k-NN answer set. The upper bound is then used in a manner similar to *best-first* search to prune the search space. The idea of VPs is inspired from Lipschitz embedding [10].

***Definition** 6:* VANTAGE POINT: *A VP is a spatial point in the trajectory space. The distance between a trajectory T and a VP v is the distance between v and the point p in T that is closest to v. Note, p may not necessarily be a sampled point.*

$$\text{VP-dist}(T, v) = min_{\forall e \in T}\{\arg\min_{p \in e} \quad dist(v, p))\} \tag{12}$$

where e is an st-segment in T.

***Definition** 7:* VANTAGE DESCRIPTOR: *Given a set of VPs $\mathbb{V} = \{v_1, \cdots, v_d\}$, the vantage descriptor of T is a d-dimensional feature vector $T_{\mathbb{V}} = [$VP-dist$(T, v_1), \cdots,$ VP-dist$(T, v_d)]$. We use $T_{\mathbb{V}}[i]$ to denote the i_{th} dimension in $T_{\mathbb{V}}$.*

In essence, the vantage descriptor of a trajectory captures a feature space representation based on the viewpoints of VPs in $\mathbb{V}$. Next, we define *vantage distance* between two trajectories.

***Definition** 8:* VANTAGE DISTANCE: *The vantage distance between trajectories T_1 and T_2 is defined as the following:*

$$VD(T_1, T_2) = \frac{\sum_{i=0}^{|\mathbb{V}|} 1 - \frac{\min\{T_{1_{\mathbb{V}}}[i], T_{2_{\mathbb{V}}}[i]\}}{\max\{T_{1_{\mathbb{V}}}[i], T_{2_{\mathbb{V}}}[i]\}}}{|\mathbb{V}|} \tag{13}$$

Vantage distance builds on the intuition that if trajectories T_1 and T_2 are equidistant from most of the distributed VPs, then they are likely to have traveled through similar regions. In other words, EDwP(T_1, T_2) and $VD(T_1, T_2)$ are likely to be correlated. Now, let $\mathcal{A}$ be the true k-NN answer set with respect to a given query trajectory Q, and $\mathcal{A}_{\mathcal{VD}}$ be the k-NN based on VPs. We define an upper bound UB as the following

$$UB = \arg \max_{\forall T' \in \mathcal{A}_{\mathcal{VD}}} \text{EDwP}(T', Q) \tag{14}$$

It is easy to see that $\forall T \in \mathcal{A}$, EDwP$(T, Q) \leq UB$. Indeed, an upper bound can be computed from any random selection of k trajectories from the database. However, the tightness of the upper bound depends on how well the selected trajectories overlap with $\mathcal{A}$. VPs allow us to make a more informed selection of k database trajectories and thereby, producing a tight upper bound. Additionally, since $VD(T_1, T_2)$ operates in the feature space and the computation cost is linear, $VD(T_1, T_2)$ is much faster than EDwP(T_1, T_2). To take advantage of these properties, we integrate VPs with TrajTree.

While constructing the index, in addition to computing the tBoxSeq at each node n, d VPs are also distributed and the resultant vantage descriptors of all trajectories in the subtree rooted at n are stored (Fig. 4(b)). The VPs are chosen using the same mechanism used for selecting pivots while splitting a node. Based on the chosen VPs, the vantage descriptor for each trajectory in the subtree of n is stored. Note that the density of VPs increases as lower levels of the tree are reached since the number of trajectories under a subtree decreases with the level. Consequently, resultant vantage descriptors get more refined as the depth of a node in the tree increases.

F. Indexing costs and updates

Storage Cost: The storage cost depends on the branching factor, which is controlled by θ. Assuming a balanced branching factor of bf, the height of the tree is $\log_{bf} |\mathbb{D}| + 1$, where $\mathbb{D}$ is the trajectory database. The number of nodes at each level of the tree follows a geometric progression, and is thus bounded by $O(\frac{bf|\mathbb{D}|-1}{bf-1})$. Additionally, at each level of the tree, $|\mathbb{D}|$ vantage descriptors are stored. Therefore, total storage is $O(\frac{bf|\mathbb{D}|-1}{bf-1} + |\mathbb{V}||\mathbb{D}| \log_{bf} |\mathbb{D}|)$, where $|\mathbb{V}|$ is the number of VPs at each node.

Index Construction Cost: Except at the last level of the tree, a cost of O($p|\mathbb{D}|$) is incurred at each level to select the pivots, where p is the number of pivots. The number of pivots at any level l of the tree follows the equation bf^l. Thus, the total index construction cost is bounded by $O(\frac{|\mathbb{D}|^2}{bf})$.

Updates: The index construction algorithm *bulk-loads* all trajectories in the database. To insert a new trajectory, the algorithm remains identical except for omitting the initial step of selecting pivots at each node since existing pivot points are reused. For deleting a trajectory, the corresponding leaf node and the vantage descriptors at all nodes from the leaf to the root are deleted. The tBoxSeqs remain unchanged. As in most hierarchical index structures, updates decrease efficiency since each insert or delete is likely to make the affected tBoxSeqs loose. Thus, to tackle this issue, we track the diversity of the pivots at each node. A node is classified as "poor" if the increase in diversity due to re-computing the pivots is above a threshold. Finally, if the number of "poor" nodes is above a certain ratio, then the index is re-built.

Algorithm 2 Query(Q, k)

```
1:  cands ← priority queue containing the root node
2:  ans ← empty priority queue of maximum size k
3:  processed ← ∅
4:  while cands ≠ ∅ do
5:      C ← cands.dequeue()
6:      if (|ans| < k or C.dist< ans.head().dist) then
7:          if C is a non-leaf node then
8:              for each trajectory T ∈ C.getVPtopk(Q, k, processed) do
9:                  ans.insert(< T, EDwP(Q, T) >)
10:                 processed ← processed ∪ {T}
11:             for each child C' of C do
12:                 if |ans| < k or EDwP_sub(Q, C') < ans.head().dist then
13:                     cands.insert(< C',EDwP_sub(Q, C') >)
14:         else if C ∉ processed then
15:             ans.insert(C)
16:             processed ← processed ∪ {C}
17: return ans
```

G. Querying Algorithm

Alg. 2 outlines the pseudocode to answer k-NN queries. The algorithm proceeds in a manner similar to *best-first* search. Two priority queues, $cands$ and ans, are maintained to compute the answer set. $cands$ prioritizes the unexplored nodes in the index based on their distances to the query. Initially, $cands$ contains just the root node. ans maintains the k-NN trajectories found at any given state of the searching procedure (line 2). ans is populated in an iterative manner (lines 4-16). At each iteration, the node C with the lowest distance is extracted from $cands$ and processed if it satisfies the constraints (lines 5-6).

Case 1: If C is a tBoxSeq, then the following steps are performed.
1. Update upper bound based on VPs at C: First, the top-k nearest trajectories to Q are computed based on the VPs at C. This analysis searches only those trajectories that are under the subtree rooted at C. Next, the actual distance to each of the VP-based k-NN trajectories are computed to generate an upper bound UB as outlined in Eq. 14 (UB is ans.head().dist in Alg. 2). Additionally, the trajectories in the VP-based k-NN are inserted to ans if they are currently within the top-k (lines 8-10). UB is next used to prune the children of C.

2. Prune children of C based on the updated lower bound: Once the upper bound from step 1 is derived, EDwP$_{sub}$ for each children of C is computed and added to $cands$ if they have a distance below UB (lines 11-13).

Case 2: If C is a trajectory, ans is updated based on C's distance to query trajectory Q (lines 14-16). As can be seen, a hashset called $processed$ is maintained to track trajectories that have already been evaluated. If not for $processed$, due to possible overlaps between the VP-based k-NN sets of nodes in the same path, trajectories common to both sets would be processed multiple times.

Therefore, to summarize the features of TrajTree querying:

- The k-NN answer set is exact and optimal.
- The distances of query trajectory Q to tBoxSeqs prioritize the search order.
- VPs produce a tight upper bound UB by approximating the optimal k-NN answer set. Since VPs operate at a feature space, computing UB is fast.

V. Experiments

In this section, we demonstrate that:

- EDwP is more accurate and robust to noise than state-of-the-art trajectory matching techniques.
- TrajTree is efficient in indexing k-NN queries.

A. Experimental Setup

All our algorithms are implemented in Java 1.6.0 and benchmarked on a PC with 12GB memory and Intel i5 2.60GHz quad core processor running Ubuntu 12.10. We use the length normalized EDwP defined in Eq. 4.

Datasets: We use two real trajectory datasets. The first dataset is extracted from the Beijing cab dataset [18]. This dataset contains trajectories of $10,000$ cabs tracked over a period of one week in Beijing [18]. Since we would like each trajectory to represent a single trip, we partition a trajectory into two if either the cab is stationary for more than 15 minutes, or the time gap between two consecutive points is more than 15 minutes. Following this procedure, we use a subset $42,000$ trajectories to create our dataset.

Besides vehicular trajectories, we also use the Australian Sign Language (ASL) dataset, which contains trajectories of hand movements that denote 98 different signs such as "alive", "cold", "computer", etc. Thus, each trajectory in the ASL dataset is labeled with the sign it denotes.

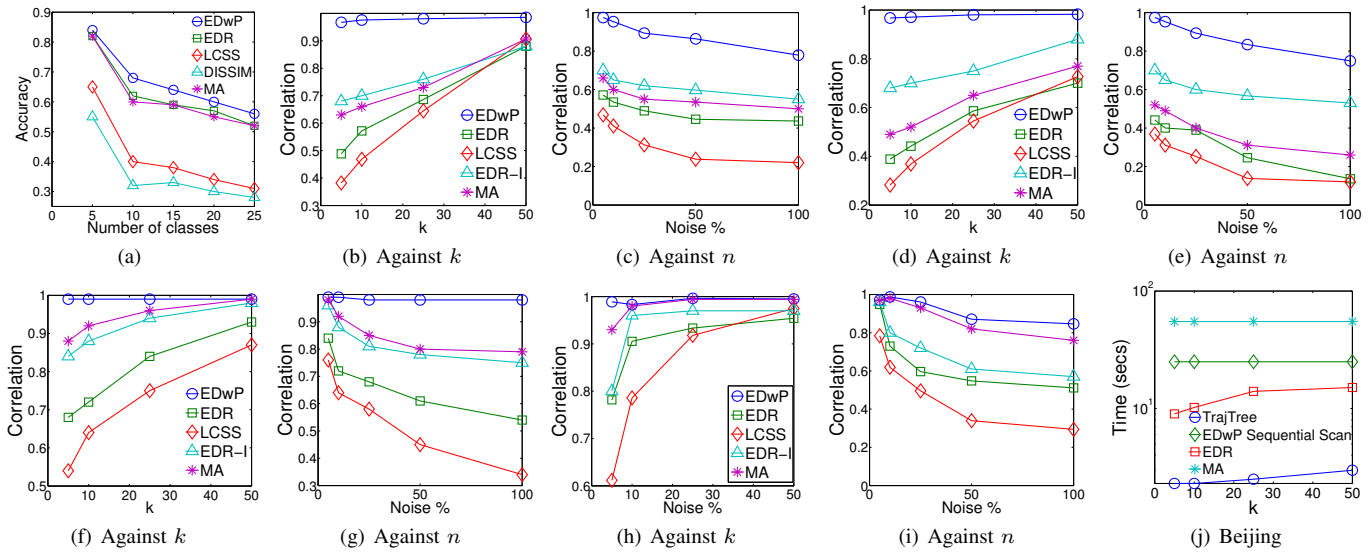
Benchmarking Techniques: We compare the accuracy of EDwP with LCSS [3], EDR [5], DISSIM [7] and the semi-continuous assignment model in MA [8]. EDR and LCSS have been shown to outperform DTW [6] and ERP [4] in [5].

Parameters: While EDwP is parameter-free, EDR and LCSS are dependent on a matching threshold. MA has as many as four parameters. We set these parameters as outlined by the respective papers. TrajTree, requires three parameters. The default values of θ in Alg. 1 and the number of vantage points are set to 0.8 and 80 respectively. The minimum size of a node in TrajTree is set to 10. The default k in the k-NN queries is set to 10.

B. Accuracy in clean data

First, we evaluate the accuracy of EDwP in clean data. The ASL dataset contains trajectories from 98 different classes with each class denoting a sign. These trajectories are recorded in controlled environments and are thus clean. To measure accuracy, we perform multi-class classification. We prepare the dataset for the classification experiment by first randomly selecting c classes, and then adding all trajectories under these classes to the classification dataset. Next, using each of the four distance metrics, we perform 10-fold cross-validation. The class labels of trajectories in the testing set is predicted using nearest neighbor classification. The process is repeated 100 times for consistency.

Fig. 5(a) demonstrates the results as the number of classes is varied from 5 to 25. EDwP achieves the highest accuracy across all numbers of classes. As the number of classes increases, the classification task gets harder and consequently, the accuracies of all distance metrics decrease. This rate of



(a) (b) Against k (c) Against n (d) Against k (e) Against n

(f) Against k (g) Against n (h) Against k (i) Against n (j) Beijing

Fig. 5. (a) Classification accuracy of EDwP, EDR, LCSS, DISSIM, and MA on the ASL dataset. Variance of Spearman's rank correlation of EDwP, EDR, EDR-I, LCSS, and MA in the Beijing dataset against (b-c) inter-trajectory sampling rate variance, (d-e) intra-trajectory sampling rate variance, (f-g) phase variations, (h-i) threshold dependency. The change in correlation is measured against k and noise percentage n. The legends for plot b to plot i are the same. (j) Growth rate of querying time with k.

decrease however, is the smallest in EDwP indicating better robustness to matching difficulty. The performance of DISSIM suffers since it is unable to cope with local time shifts.

C. Accuracy in noisy data

In this section, we benchmark the robustness of EDwP to sampling noise. We model each of the scenarios discussed in Sec. II and compare the performances of the distance metrics. For all of the experiments in this section, we construct two datasets, D_1 and D_2. While D_1 is clean, the second dataset D_2 is noisy. The process of injecting noise is dependent on the issue being evaluated. Now, to quantify robustness, we first construct a ground truth result set by computing the k-NN list for a randomly chosen query from the clean D_1 dataset. Next, we re-compute the k-NN list for the same query trajectory in the noisy D_2 dataset. A robust distance metric should adapt to the injected noise and produce an answer set that is close to the k-NN on the clean D_1 dataset. Based on this hypotheses, we compute Spearman's rank correlation coefficient [19] between the two k-NN lists. Since the elements in the two k-NN lists may not overlap completely, we form a single element set by taking their union. Next, for each element in the union set, we fetch its ranks in D_1 and D_2. Finally, we compute the correlation between the two ranked lists. The closer the correlation is to 1, the more robust the distance metric is.

For a thorough investigation of the performance, we also apply EDR on an interpolated dataset. As discussed in Sec. II, issues related to non-uniform sampling rates can be alleviated by interpolating the dataset with additional points so that all trajectories have a uniform sampling rate. As already discussed, this pre-processing step is extremely expensive and needs to be performed at query time. However, for the purposes of better understanding the impact of sampling rate, we ignore the scalability aspect and perform interpolation on both D_1 and D_2 to ensure a uniform sampling rate. Except this pre-processing, the subsequent steps for the k-NN query remain identical. We denote EDR on this interpolated scenario as *EDR-I*. DISSIM is not included in this experiment as it is unable to detect spatial similarity between trajectories moving at dissimilar speeds.

Inter-trajectory sampling variance: In this experiment, D_1 corresponds to the Beijing dataset. To model variance in inter-trajectory sampling rates in D_2, without altering the shape of a trajectory $T \in D_1$, we randomly select $n\%$ of its segments, and partition them into two by inserting a point. Thus, D_2 represents the same trajectories in D_1 at a higher sampling rate. n is the noise parameter that controls the difference in sampling rate between trajectories in D_1 and D_2. Fig. 5(b) demonstrates the results at $n = 5\%$ while k is varied between 5 and 50. Fig. 5(c), on the other hand, analyzes the performance at $k = 10$ as n is varied to increase the difference in sampling rate between D_1 and D_2. Across both k and n, EDwP achieves an accuracy that is up to 4 times better than existing techniques. As expected, the correlation increases with k since there is more overlap with the true k-NN list as k grows. Regardless of k however, EDwP achieves a rank correlation that is close to 1 due to dynamic interpolation achieved through projections. Against n, the performance difference is even more drastic. Even in the worst case scenario, EDwP's correlation with the true answer set is higher than 0.75. Despite interpolation, EDR-I performs worse than EDwP since it is limited to matching only the existing sampled (or interpolated) points. On the other hand, the alignment possibilities for EDwP are infinite due to projections. MA suffers since with the introduction of additional points, the trade off between 'gap points' and matched points change.

Intra-trajectory sampling variance: Similar to the previous experiment, D_1 corresponds to the Beijing dataset. To model intra-trajectory variance in sampling rates in D_2, we take only the first half of each trajectory, and within this half, we insert points in $n\%$ of the segments in a manner similar to the previous experiment. Thus, the sampling rate in

each trajectory in D_2 is higher in the first half. Figs. 5(d) and 5(e) demonstrate the results. Although the trends are similar to the results for inter-trajectory sampling variance, the performance gap between EDR-I and EDR is significantly higher. Due to the interpolation in the pre-processing step, the difference in sampling rate disappears in EDR-I, and the scenario degenerates to the case of only inter-trajectory sampling rate difference between D_1 and D_2. Overall, EDwP is up to 50% more correlated, since it weights each segment alignment based on its coverage and thus automatically adapts to the variance in the sampling rate.

Phase Variations: To model phase variations, we construct two datasets from Beijing. Specifically, for each trajectory, we randomly choose $n\%$ of its segments and partition each segment into two by inserting an additional point. This altered trajectory is next added to D_1. Now, the exact same set of segments are partitioned again and added to D_2. Thus, the sampling rate and the set of altered segments in D_1 and D_2 are identical; the only difference lies in the location of the inserted point. As can be seen in Figs. 5(f) and 5(g), although the trends are similar, generally, the performance of EDR and LCSS are better than inter and intra-trajectory sampling variances with MA performing best among existing techniques. Since MA considers aligning a sampled point to a non-sampled point, its performance is stable. EDwP performs the best due to projections and coverage.

Threshold Dependency: Finally, to highlight the impact of threshold dependency, we perturb the locations of $n\%$ of the st-points in each trajectory by a small amount and add it to D_2. D_1 represents the original Beijing dataset. To perturb a point, we draw a circle with the original location as the center and a radius equivalent to the distance traveled in 30 seconds based on the average speed of trajectories in the dataset. The perturbed location is then set to one randomly selected point within the circle. Figs. 5(h) and 5(i) demonstrate the results against k (at $n = 10\%$) and n (at $k = 10$) respectively. While the impact of threshold is not as drastic as with sampling rate variance, all techniques suffer due to relying on thresholds.

D. Index Performance

We now verify the efficiency of TrajTree in indexing k-NN queries. We compare the querying time of TrajTree with the index structure for EDR [5], and sequential scans over EDwP and MA. No index structure exists for MA [8]. For EDR, we ensure uniform sampling rates through interpolation since EDR-I is the closest to EDwP in terms of robustness.

1) Online costs: Fig. 5(j) demonstrates the growth rate of querying times with k (time in log-scale). TrajTree is more than an order of magnitude faster than MA and up to 5 times faster than EDR. Although the computational complexities of all three matching techniques are quadratic, MA is the slowest since it depends on 5 auxiliary functions, each of which has a quadratic computation cost. In other words, the constant factor in MA is 5 times higher. Compared to EDR, EDwP is faster since it performs dynamic interpolation through projections.

Next, we look at the growth rate of the querying time with database size. Fig. 6(a) demonstrates the result. The growth rate is sublinear for both TrajTree and EDR. EDwP is up to 5 times faster than EDR and 10 times faster than sequential scan, which establishes the efficiency of the pruning strategies. As we show later in Fig. 6(c), VPs allow us to derive a tight upper bound at the very beginning of the search process in the root node. As a result, a significant portion of the tBoxSeqs can be pruned out based on EDwP$_{sub}$. Furthermore, as the search process proceeds toward lower levels of the tree, the bounds get tighter.

Next, we study the performance dependence of TrajTree on the branching factor, which is controlled by θ. As outlined in Alg. 1, θ dynamically controls the branching factor by analyzing the marginal drop in the diversity of the selected pivots. Fig. 6(b) analyzes the optimal value for θ. The querying times are minimized at 0.8, which means the optimal balance between the tightness of tBoxSeqs and the number of EDwP$_{sub}$ computations is obtained. This result influences our default choice of $\theta = 0.8$.

The second parameter that impacts the performance of TrajTree is the number of vantage points. The tightness of the upper bound computed in Eq. 14 is influenced by the number of VPs distributed. We verify this tightness. Towards that goal, we introduce the notion of *UB-factor*.

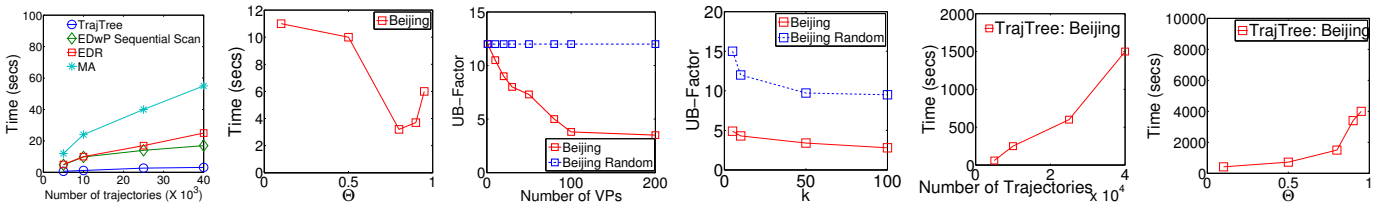
$$\text{UB-Factor} = \frac{\text{VP-based Upper Bound}}{k^{th} \text{ highest distance in actual } k\text{-NN}} \tag{15}$$

In the best case scenario, the optimal upper bound is the k^{th} highest distance in the actual k-NN answer set. We therefore set the optimal upper bound as the denominator of the UB-Factor. Thus, the closer the UB-Factor is to 1, the better is the performance.

Fig. 6(c) studies the variance in UB-Factor at the root node with increase in the number of VPs. Note that at lower levels of the tree, the UB-Factor can only get tighter since TrajTree continuously refines the upper bound by evaluating only unexplored trajectories. Additionally, the density of VPs increase with decrease in tree level. In other words, Fig. 6(c) depicts the worst case scenario. As expected from a theoretical standpoint, the UB-Factors improve with increase in the number of VPs.

To further investigate the tightness of the upper bounds derived using VPs, we also study the *Random UB-Factors* in Fig. 6(c), denoted as *Beijing Random*. A Random UB-Factor is simply the UB-Factor from a random subset of k trajectories in the database. Since the VP-based upper bound is derived from an approximate k-NN, comparison to the random UB-Factor allows us to judge the randomness in the k-NN approximation performed through VPs. If the k-NN approximation by VPs is simply a random subset of k trajectories, then the difference between the VP-based UB-Factor and the random UB-Factor would be low. As can be seen in Fig. 6(c), VPs provide an UB-Factor that is significantly tighter than random and thus, efficient in pruning a large portion of the search space.

To further establish the utility of VPs, we also study the tightness of UB-Factors with increase in k. Fig. 6(d) presents the results. The UB-Factors are almost constant across k. This is a natural consequence of the fact that both the numerator and denominator of UB-Factor increases with k. Nonetheless, the upper bounds through VPs are more than 3 times tighter than a random selection. In addition to the above studies, we also analyzed the Spearman's rank correlation coefficient between



(a) Beijing (b) (c) (d) (e) (f)

Fig. 6. Growth rate of running time with (a) dataset size and (b) θ. Tightness of the VP-based upper bound against the (c) number of VPs, and (d) k. Growth rate of index construction time with (e) database size and (f) θ.

VP-based k-NNs and the actual answer sets. The correlation ranges between 0.78 to 0.83 across all values of k in $[5, 100]$.

2) Off-line Costs: First, we look at the growth rate of the index construction cost with dataset size. As can be seen in Fig. 6(e), the growth rate is smaller than quadratic but higher than a linear distribution. This result is consistent with the theoretical analysis in Sec. IV. The branching factor, regulated by θ, also influences the index construction time. Fig. 6(f) studies the impact of this parameter. Consistent with the theoretical analysis in Sec. IV, the construction time increases with increase in θ as the computation cost at each level of the tree increases. This behavior is opposite to the relationship between θ and the querying time. However, since index construction is an one-time off-line computation, optimizing the querying time takes precedence.

VI. Related Work

An elaborate discussion on existing trajectory distance metrics and their weaknesses has already been done in Sec. II. Initial efforts on indexing trajectory retrieval were primarily directed towards indexing DTW [6], [20]. In subsequent works, LCSS [3], ERP [4] and EDR [5] proposed index structures for their respective distance metrics. Unfortunately, these index structures do not generalize to EDwP and thus, we develop TrajTree. Although we develop distance bounds specifically for EDwP, TrajTree generalizes the idea of bounding boxes for trajectories and can potentially be utilized for other trajectory operations. Applying bounding boxes on trajectories has also been explored by the TB-tree index [21] and SETI [22]. However, in contrast to TrajTree, both TB-tree and SETI partitions a trajectory into multiple segments, and a bounding box summarizes each of these partitions. As a result, a trajectory is distributed over various nodes in TB-tree and SETI. In TrajTree, each node is a sequence of bounding boxes that summarizes a set of trajectories. The design choice of TB-tree and SETI stems from their goal of identifying database trajectories that fall within a query temporal or spatial interval. In contrast, our goal is to index similarity queries for which TB-tree and SETI cannot be used to compute a lower bound on EDwP.

VII. Conclusion

In this paper, we studied the problem of matching trajectories under inconsistent sampling rates. To tackle this noise, we developed *Edit Distance with Projections (EDwP)*, which is insulated from the choice of sampled regions, non-uniform sampling rates, and local time shifts by employing the ideas of *projections* and *coverage*. Additionally, EDwP is parameter-free. EDwP not only allows robust matching, but its ability to interpolate dynamically enables us to generalize the concept of bounding boxes for trajectories. By integrating bounding boxes with *vantage points*, we developed an index structure called *TrajTree* for fast answering of k-NN queries. Each vantage point provides a unique viewpoint on the trajectory database and by consolidating their views, we are able to derive tight upper bounds on the true distance. These upper bounds, in conjunction with the lower bounds derived from bounding boxes, drastically reduces the search space and provides up to an order of magnitude speed up over the state-of-the art retrieval techniques. Furthermore, extensive experiments show EDwP to be 5 times more accurate and robust than existing trajectory matching techniques.

References

[1] "Movebank, http://www.movebank.org."

[2] M. Passe-Smith, "Exploring Local Tornado Alleys for Predictive Environmental bibtex," in *ESRI User Conference*, 2006.

[3] M. Vlachos, D. Gunopoulos, and G. Kollios, "Discovering similar multidimensional trajectories," in *ICDE*, 2002.

[4] L. Chen and R. Ng, "On the marriage of Lp-norms and edit distance," in *VLDB*, 2004.

[5] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *SIGMOD*, 2005, pp. 491–502.

[6] B. kee Yi, H. V. Jagadish, and C. Faloutsos, "Efficient Retrieval of Similar Time Sequences Under Time Warping," in *ICDE*, 1998.

[7] E. Frentzos, K. Gratsias, and Y. Theodoridis, "Index-based Most Similar Trajectory Search," in *ICDE*, 2007, pp. 816–825.

[8] S. Sankararaman, P. K. Agarwal, T. Mølhave, J. Pan, and A. P. Boedihardjo, "Model-driven matching and segmentation of trajectories," in *SIGSPATIAL*, 2013, pp. 234–243.

[9] L.-Y. Wei, Y. Zheng, and W.-C. Peng, "Constructing popular routes from uncertain trajectories," in *KDD*, 2012, pp. 195–203.

[10] J. Bourgain, "On lipschitz embedding of finite metric spaces in hilbert space," *Israel Journal of Mathematics*, vol. 52, pp. 46–52, 1985.

[11] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *SIGMOD*, 1984.

[12] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang, "Map-matching for low-sampling-rate gps trajectories," in *GIS*, 2009.

[13] J. Yuan, Y. Zheng, C. Zhang, X. Xie, and G.-Z. Sun, "An interactive-voting based map matching algorithm," in *MDM*, 2010, pp. 43–52.

[14] K. Zheng, Y. Zheng, X. Xie, and X. Zhou, "Reducing uncertainty of low-sampling-rate trajectories." in *ICDE*, 2012, pp. 1144–1155.

[15] P. Banerjee, S. Ranu, and S. Raghavan, "Inferring uncertain trajectories from partial observations," in *ICDM*, 2014.

[16] J.-G. Lee, J. Han, and X. Li, "Trajectory outlier detection: A partition-and-detect framework," in *ICDE*, 2008, pp. 140–149.

[17] M. F. Mokbel and W. G. Aref, "Indexing historical spatio-temporal data," in *Encyclopedia of Database Systems*, 2009, pp. 1448–1451.

[18] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *KDD*, 2011, pp. 316–324.

[19] "Spearman's rank correlation, http://en.wikipedia.org/wiki/spearman

[20] E. Keogh and C. A. Ratanamahatana, "Exact indexing of dynamic time warping," *Knowl. Inf. Syst.*, vol. 7, no. 3, pp. 358–386, 2005.

[21] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches in query processing for moving object trajectories," in *VLDB*, 2000, pp. 395–406.

[22] V. P. Chakka, A. Everspaugh, and J. M. Patel, "Indexing large trajectory data sets with seti," in *CIDR*, 2003.

APPENDIX

A. Triangular Inequality

THEOREM 1. *EDwP does not satisfy triangular inequality.* PROOF: Consider three trajectories $T_1 = [(0,0),(0,1)]$, $T_2 = [(0,0),(0,1),(0,2)]$, and $T_3 = [(0,0),(0,1),(0,2),(0,3)]$. We ignore the timestamps since the proof is independent from them. Now, EDwP$(T_1,T_2) = 1 \times 1$, EDwP$(T_2,T_3) = 1 \times 1$, and EDwP$(T_1,T_3) = 2 \times 2$. Thus, EDwP(T_1,T_2)+EDwP$(T_2,T_3) <$ EDwP(T_1,T_3).

B. Proof of Lemma 3

LEMMA 3 *Consider two trajectories T_1 and T_2 such that both contain one st-segment each. If a new trajectory $T_2' = [[T_2.e_1.s_1, p], [p, T_2.e_1.s_2]]$ is created by inserting an additional point p on $T_2.e_1$, then EDwP$(T_1,T_2') \leq$ EDwP(T_1,T_2).*

PROOF: We first define the following variables:

$p' = p^{ins(T_1.e_1,[T_2.e_1.s_1,p])}$,

$a = dist(T_1.e_1.s_1, T_2.e_1.s_1)$,

$b = dist(\ T_1.e_1.s_2, T_2.e_1.s_2)$,

$\Delta = dist(p, p')$,

$d_1 = dist(T_1.e_1.s_1, p') + dist(T_2.e_1.s_1, p)$,

$d_2 = dist(T_1.e_1.s_2, p') + dist(T_2.e_1.s_2, p)$.

Fig. 7 demonstrates the variables pictorially.

In this proof, we show that the edits $ins(T_1.e_1, [T_2.e_1.s_1, p])$ (followed by $rep([T_1.e_1.s_1, p'], [T_2.e_1.s_1, p])$) and $rep([p', T_1.e_1.s_2], [p, T_2.e_1.s_2])$ on T_2' is cheaper than $rep(T_1.e_1, T_2.e_1)$ on the original trajectories. Specifically,

$$(a+b) \times (d_1+d_2) - (a+\Delta) \times d_1 - (b+\Delta) \times d_2 \geq 0 \quad (16)$$

From the construction of T_2', it is clear that T_2''s shape is identical to T_2. Without loss of generality, we assume $b \geq a$. Due to symmetry, the proof for the opposite case follows analogously. Additionally, it is guaranteed $\nexists\ \Delta, \Delta > b$ and $\Delta > a$. Now, while p is projected to the closest point in $T_1.e_1$, in $rep(T_1.e_1, T_2.e_1)$, the distance between the endpoints of the st-segments are considered. Based on this observation, we divide the proof into two cases.

Case 1: Line pp' is perpendicular to $T_1.e_1$ It is known that the shortest distance from a point to a line is its perpendicular projection. Clearly, the left hand side (LHS) of Eq. 16 is minimized if a and b correspond to perpendicular projections of endpoints $T_2.e_1.s_1$ and $T_2.e_1.s_2$ respectively. Fig. 7 demonstrates an instance of this case. From geometry, b, Δ and a are parallel to each other and thus can be expressed as the following:

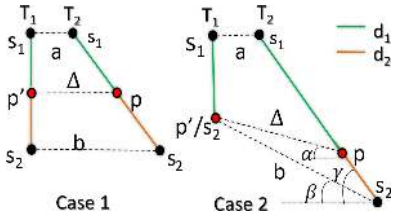


Fig. 7. Demonstrates the setup for the proof of Lemma 3. Although T_1 is assumed to be axis parallel, for arbitrary trajectories, it can always be rotated to simulate the same setup.

$$b = a + r \times (d_1 + d_2) \quad (17)$$

$$\Delta = a + r \times (d_1) \quad (18)$$

where $r = \frac{b-a}{d_1+d_2}$ defines the rate at which the projected distance for a point in $T_2.e_1$ grows. Therefore, LHS of Eq. 16 equals:

$$= ad_2 + (a + r(d_1+d_2))d_1 - (a + r \times d_1)(d_1+d_2) = 0$$

Case 2: Line pp' is not perpendicular to $T_1.e_1$ This case arises when $T_1.e_1$ does not contain the point that corresponds to the perpendicular projection of p. The non-optimality of the projected distance is maximized if p is projected to one of the endpoints of $T_1.e_1$. In such a case, it is guaranteed that either a or b is not a perpendicular projection either. Without loss of generality, we assume that p is projected to $T_1.e_1.s_2$, which in turn guarantees that b is the non-perpendicular projection. More precisely, $p^{ins(T_1.e_1,[T_2.e_1.s_1,p])} = p^{ins(T_1.e_1,T_2.e_1)} = T_1.e_1.s_2 = p'$. The proof for $p^{ins(T_1.e_1,[T_2.e_1.s_1,p])} = T_1.e_1.s_1$ follows analogously.

The LHS of Eq. 16 is minimized if a is the perpendicular projection of $T_2.e_1.s_1$. We now introduce the following variables. Let γ be the angle at which $T_2.e_1$ moves away from $T_1.e_1$. Furthermore, α and β are the angles between Δ and b with their corresponding imaginary perpendicular projections. Fig. 7 demonstrates each of these notations. Clearly, $\gamma \geq \beta \geq \alpha$. In the triangle, $\triangle p'\ p\ T_2.e_1.s_2$, $\angle T_2.e_1.s_2\ p'\ p = \beta - \alpha$, $\angle p T_2.e_1.s_2 p' = \gamma - \beta$, and $\angle p' p T_2.e_1.s_2 = 180 - \gamma + \alpha$. Using the law of sines, the following relationship is established.

$$\frac{d_2}{sin(\beta-\alpha)} = \frac{\Delta}{sin(\gamma-\beta)} = \frac{b}{sin(\gamma-\alpha)} \quad (19)$$

$$b - \Delta = d_2 \frac{sin(\gamma-\alpha) - sin(\gamma-\beta)}{sin(\beta-\alpha)} \quad (20)$$

Now, let $l_1 = dist(T_2.e_1.s_1, p)$. Therefore, from geometry,

$$\Delta - a = l_1 \frac{cos(\gamma)}{cos(\alpha)} \quad (21)$$

Combining these observations, the LHS of Eq. 16 is

$$= d_1 \times (b - \Delta) - d_2 \times (\Delta - a)$$

$$= d_1 d_2 \frac{sin(\gamma-\alpha) - sin(\gamma-\beta)}{sin(\beta-\alpha)} - d_2 l_1 \frac{cos(\gamma)}{cos(\alpha)}$$

$$\geq d_1 d_2 \frac{cos(\alpha)sin(\gamma-\alpha) - cos(\alpha)sin(\gamma-\beta) - cos(\gamma)sin(\beta-\alpha)}{sin(\beta-\alpha)cos(\alpha)}$$

$$\geq d_1 d_2 \frac{(cos(\alpha) - cos(\beta))\ sin(\gamma-\alpha)}{sin(\beta-\alpha)cos(\alpha)} \geq 0 \qquad \square$$