Parallel Subtrajectory Alignment over Massive-Scale Trajectory Data

Lisi Chen[1] , **Shuo Shang**[1*] , **Shanshan Feng**[2] and **Panos Kalnis**[3]
[1]University of Electronic Science and Technology of China
[2]Harbin Institute of Technology, Shenzhen, China
[3]King Abdullah University of Science and Technology, Saudi Arabia
{chenlisi.cs, jedi.shang}@gmail.com, victor_fengss@foxmail.com, panos.kalnis@kaust.edu.sa

Abstract

We study the problem of subtrajectory alignment over massive-scale trajectory data. Given a collection of trajectories, a subtrajectory alignment query returns new targeted trajectories by splitting and aligning existing trajectories. The resulting functionality targets a range of applications, including trajectory data analysis, route planning and recommendation, ridesharing, and general location-based services. To enable efficient and effective subtrajectory alignment computation, we propose a novel search algorithm and filtering techniques that enable the use of the parallel processing capabilities of modern processors. Experiments with large trajectory datasets are conducted for evaluating the performance of our proposal. The results show that our solution to the subtrajectory alignment problem can generate high-quality results and are capable of achieving high efficiency and scalability.

1 Introduction

With the continued proliferations of GPS-enabled devices (e.g., smartphones, vehicle navigation systems, wearable smart devices) and online location-based services (e.g., Uber, Lyft, Google Maps), trajectory data is being generated at an unprecedented scale. For example, the average number of for-hire vehicle trips per day from New York City during the first half of year 2020 is well over 400K.[1] The availability of massive-scale trajectory data enables trajectory similarity search [Chen *et al.*, 2010; Driemel *et al.*, 2020; Xie *et al.*, 2017; Yuan and Li, 2019; Shang *et al.*, 2014], which finds trajectories that are similar in some specific sense to a set of query locations. Trajectory similarity search is one of the fundamental problems in spatio-temporal data analytics. It has a broad range of applications, including in-depth trajectory data explorations, ridesharing service analytics, route planning and recommendations for energy saving and emission reduction, and other location-based services.

Given a collection of trajectory data D, existing studies formulate the trajectory similarity search problem as a query that finds a subset of D based on spatial (i.e., proximity to query locations) and temporal (i.e., travel time interval) constraints. However, it is likely that the quality of query results fails to meet users' expectation due to data sparsity, especially when a user defines a very stringent spatio-temporal constraint (i.e., a narrow time interval, a high similarity threshold). Consider a toy example in Figure 1, where v_s and v_e, denoted by red pentagon and star, respectively, are source and destination locations, $O = \{o_1, o_2, o_3\}$ are query locations, T = [8:30,9:30] denotes the query constraint of travel time interval. Let τ_1 (yellow), τ_2 (dark blue), τ_3 (green), τ_4 (purple), and τ_5 (light blue) be five trajectories in D. Traditional trajectory similarity search aims to find trajectories that are spatially and temporally close to the query. In this example, we may find that τ_1 is the only trajectory that meets departure (v_s), destination (v_e), and time interval ($[8:30, 9:30]$) requirements. However, this result may not be good enough in real applications like ridesharing recommendations because τ_1 is not close enough to query locations. Additionally, if a user set a stringent requirement on similarity threshold, it is very likely that no trajectory in D meets query requirements.

In this light, we study a novel problem of Sub-Trajectory Alignment (STA) over a collection of trajectory data D. The STA problem not only considers existing trajectories in D, but also generates and evaluates new trajectories by splitting and aligning exiting trajectories in D. Specifically, given source and destination locations, v_s and v_e, respectively, a set of query locations O, a travel time interval $T = [t_s, t_e]$, a similarity threshold θ, the maximum number of sub-trajectory alignments M, the STA problem finds all existing trajectories in D and all trajectories generated by sub-trajectory alignments from D such that: (1) Each result trajectory τ_r covers v_s and v_e and the arrival timestamps regarding v_s and v_e are within T; (2) The similarity between τ_r and O is no less than θ; (3) Trajectory τ_r is generated from the alignment of at most $M + 1$ existing trajectories from D. Consider the example in Figure 1, we may derive two feasible trajectories by aligning two existing trajectories: $\tau_{r_1} = v_s \rightarrow \tau_2 \rightarrow v_0 \rightarrow \tau_5 \rightarrow v_e$; $\tau_{r_2} = v_s \rightarrow \tau_2 \rightarrow v_3 \rightarrow \tau_3 \rightarrow v_e$. Compared to the original result trajectory τ_1, τ_{r_2} is more similar (closer) to query locations o_1, o_2, and o_3. Thus, we may acquire results with higher quality by aligning only two existing trajectories.

It is challenging to answer the STA problem efficiently. Given query location set O, we have $\sum_{i=0}^{M} S(|O|, i+1)$ dif-

*Corresponding author.

[1]https://data.cityofnewyork.us/browse?q=trip

ferent partitionings where $S(\cdot,\cdot)$ denotes the Stirling number of the second kind. Thus, it is computationally prohibitive to enumerate and evaluate all possible location partitionings. To address the challenge, we develop Parallel Sub-Trajectory Alignment Search (PSTAS) algorithm for answering the STA problem. The high-level idea of PSTAS works as follow. Let q be an STA query, which consists of source location v_s, destination location v_e, a set of query locations O, travel time interval $T = [t_s, t_e]$, similarity threshold θ, and the maximum number of sub-trajectory alignments M. The PSTAS has two phases: (1) Generation of trajectory candidates through network expansion; (2) Alignment of trajectory candidates through bottom-up merging.

Initially, we regard v_s, v_e, and each location in O as individual network expansion centers. We run Dijkstra's Algorithm [Dijkstra, 1959] to explore the road network and to find trajectory candidates that are close to each expansion center. Here, we propose a pruning technique to filter out unqualified candidates at an early stage. Note that the expansions from different locations are independent of each other so they can be executed in parallel.

Next, we generate partitionings of the set O. Each partitioning contains k disjoint expansion center sets ($k \in [1, M+1]$). For each expansion center set in each partitioning, we evaluate and merge the trajectory candidates associated with the expansion center. For each partitioning, we merge the candidates associated with each expansion center set in a bottom-up fashion. The computations in each group and in each partitioning occur in parallel. Similarly, the computations in different partitionings are independent of each other so they are capable of running in parallel.

Our contributions are summarized as follows. First, we define and study a novel problem of Sub-Trajectory Alignment (STA), aiming to address the limitation of low result quality for traditional trajectory similarity search with insufficient trajectory data. Second, we develop a PSTAS algorithm that is capable of answering the STA problem efficiently. Third, we conduct extensive experiments on large trajectory data. The experiment results confirm that: (1) Answering the STA problem is substantially more likely to find high-quality trajectory results in comparison to answering existing trajectory similarity search problem; (2) Our PSTAS algorithm is capable of answering the STA problem over a collection of up to 10M trajectories with user interaction time.

2 Preliminaries and Problem Definition

2.1 Road Network and Trajectory

We formulate a road network by a connected and undirected graph $G = (V, E, F, W)$, where V denotes a set of vertices and $E \subseteq \{\{v_i, v_j\} | v_i, v_j \in V \wedge v_i \neq v_j\}$ denotes a set of edges. Here, a vertex $v \in V$ denotes a road intersection or endpoint, an edge $e = \{v_i, v_j\} \in E$ denotes a road segment. Function $F : V \cup E \rightarrow Geometries$ denotes the following two mapping operations: (1) Mapping a vertex to the point location (road intersection or endpoint); (2) Mapping an edge to a polyline that represents the corresponding road segment. Function $W : E \rightarrow R$ assigns a weight $W(e)$ to an edge e, which denotes the corresponding road segment's length. For simplicity, we assume that the location points are located on vertices.

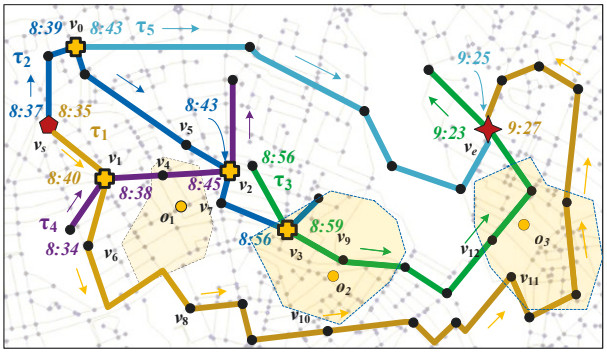


Figure 1: An example of STA problem

This modeling of road networks aligns with previous studies (e.g. [Chen *et al.*, 2010; Shang *et al.*, 2014]). Trajectory and sub-trajectory are defined as follow.

Definition 1 *(Trajectory) Trajectory τ is a sequence of vertex-timestamp pairs $\langle p_1, p_2, ..., p_n \rangle$. Note that $p_i = (v_i, t_i)$ where v_i represents the vertex and t_i denotes the timestamp when τ passes v_i.*

Definition 2 *(Sub-trajectory) A sub-trajectory of τ, denoted by $\tau(p_i \rightarrow p_j)$, is a segment of τ, where $p_i, p_j \in \tau$ and p_i and p_j are the start and end vertex-timestamp pairs of the segment, respectively.*

2.2 Problem Formulation

Definition 3 *(Valid Sub-Trajectory Alignment) Given a source location v_s, a destination location v_e, a time interval $T = [t_s, t_e]$, and a collection of trajectories D, a valid sub-trajectory alignment, denoted by $\langle \tau_1(p_1 \rightarrow p_2), \tau_2(p_2 \rightarrow p_3), ..., \tau_n(p_n \rightarrow p_{n+1}) \rangle$, satisfies the following conditions: (1) $n \geq 2$; (2) $\tau_i \in D$ $(i \in [1, n])$; (3) $p_1 = (v_s, t_1)$ where $t_1 \geq t_s$; (4) $p_{n+1} = (v_e, t_{n+1})$ where $t_{n+1} \leq t_e$.*

Definition 4 *(Sub-Trajectory Alignment (STA) Problem) Given a collection D of trajectories, the STA problem takes the following arguments as input: (1) Source and destination locations, v_s and v_e, respectively, and a set of query locations O; (2) A travel time interval $T = [t_s, t_e]$; (3) A similarity threshold θ; (4) The maximum number of sub-trajectory alignments M. The STA problem finds all valid sub-trajectory alignments R such that: (1) The similarity between O and each sub-trajectory alignment $\tau_r \in R$, denoted by $\mathrm{Sim}(O, \tau_r)$, is no less than θ (i.e., $\mathrm{Sim}(O, \tau_r) \geq \theta$); (2) The number of sub-trajectories in each alignment does not exceed M.*

2.3 Similarity Measures

Given a location v and a trajectory τ, the proximity $d(v, \tau)$ between v and τ is defined by Equation 1.

$$d(v, \tau) = \min_{v_i \in \tau} \{d_{min}(v, v_i)\}, \quad (1)$$

where $d_{min}(v, v_i)$ denotes the shortest network distance between v and v_i. Given a set O of locations and a trajectory

τ, the similarity $\mathrm{Sim}(O, \tau)$ between them is defined by the aggregate distance [Chen *et al.*, 2010; Shang *et al.*, 2017]:

$$\mathrm{Sim}(O, \tau) = \frac{\sum_{v \in O} e^{-d(v,\tau)}}{|O|} \quad (2)$$

3 Parallel Subtrajectory Alignment Search

Our PSTAS algorithm consists of two phases: (1) Generation of trajectory candidates (Section 3.1); (2) Alignment of sub-candidates (Section 3.2).

3.1 Generating Trajectory Candidates

This section presents our algorithm details regarding how to generate trajectory candidates associated with each location. Specifically, we present how to generate trajectory candidates associated with v_s, v_e, and each location in O, respectively. Trajectory candidates are potentially capable for alignment. To generate trajectory candidates associated with v_s and v_e, we retrieve all trajectories that pass v_s after t_s, and retrieve all trajectories that pass v_e before t_e. To generate trajectory candidates associated with each location in O (i.e., o_1, o_2, o_3, and o_4), we perform network expansion from each location based on Dijkstra's algorithm [Dijkstra, 1959].

We terminate the network expansion from each location o_i when we are unable to find potentially qualified trajectory for alignment through further network expansion. As such, we pre-identify a network expansion region r_i for each location o_i. Region r_i is a circle in network space. The center of r_i is o_i and the radius is the network distance from o_i to the expansion boundary. From Figure 1, we see that the expansion boundaries of o_1, o_2, and o_3 are illustrated by light yellow areas. Note that Dijkstra's algorithm iteratively selects the vertex from road networks with the minimum distance label for expansion. As such, given a trajectory τ, if $p_m.v_m$ ($p_m \in \tau$) is the first vertex visited by the expansion from o_i, v_m is the vertex nearest to o_i. It is worthy of noting that expansions from different locations are independent of each other. Thus, they can be performed in parallel.

Given an expansion center o_i, we proceed to present how to determine the radius of its expansion region r_i. Specifically, if a vertex v is visited during the Dijkstra's expansion from o_i and trajectory τ covers v, we calculate the similarity upper bound between O and τ, denoted by $\mathrm{Sim}^{\top}_{o_i}(O, v)$, based on Equation 3.

$$\mathrm{Sim}^{\top}_{o_i}(O, v) = \frac{|O| + e^{-d_{min}(o_i,v)} - 1}{|O|} \quad (3)$$

During the process of network expansion from o_i, if $\mathrm{Sim}^{\top}_{o_i}(O, v) < \theta$, we terminate the expansion from o_i because all unvisited trajectories so far are unqualified for alignment. As shown in Figure 1, when we run network expansion from o_1, we retrieve τ_1 and τ_4 as trajectory candidates because both trajectories spatially overlap with o_1's expansion region. Likewise, we retrieve τ_2 and τ_3 as trajectory as candidates associated with o_2, and retrieve τ_1 and τ_3 as candidates associated with o_3.

Data structure. During the expansion process, we maintain a candidate set $S(o)$ for expansion center o, including v_s, v_e, and all $o_i \in O$. Each trajectory candidate τ in $S(o)$ is represented by a *trajectory label*, denoted by $l_o(\tau, v_p) = \langle e, v_p, d \rangle$, where e is the entry of τ, v_p denotes the vertex covered by τ that is visited during the network expansion from o, and d denotes the network distance from v_p to o.

Algorithm for generating trajectory candidates. Algorithm 1 presents the pseudo code for generating trajectory candidates associated with v_s, v_e, and each location $o_i \in O$. The inputs are start and end locations (v_s and v_e), locations O, similarity threshold θ, and travel time interval T. The output is trajectory candidate sets of v_s, v_e, and each location in O (i.e., $\mathcal{S} = \{S(v_s), S(v_e), S(o_1), ..., S(o_{|O|})\}$). In particular, each $S(\cdot) \in \mathcal{S}$ is indexed by a min-heap consisting of trajectory labels associated with each location. Trajectory labels in $S(o)$ are sorted in ascending order of $l_o(\tau, v_p).d$ (i.e., the network distance between o and v_p). First, we initialize the result trajectory candidate set associated with v_s, v_e, and each $o_i \in O$ (lines 1–2). Next, we find trajectory candidates associated with v_s and v_e respectively (lines 3–10). Specifically, we first evaluate all trajectories that cover v_s. For each trajectory τ, we add τ into v_s's candidate pool if τ passes v_s after $T.t_s$ (lines 5–6). Likewise, we proceed to evaluate all trajectories that cover v_e. For each trajectory τ, we add τ into v_e's candidate pool if τ passes v_s before $T.t_e$ (lines 9–10). Next, we perform a Dijkstra's Algorithm based network expansion from eachinte location $o_i \in O$. We iteratively retrieve the next vertex that has not been scanned (line 12). Then we compute the upper bound of similarity between O and trajectories covering v_p (i.e., sim_i^{ub}) (line 14). If $sim_i^{ub} \geq \theta$, we insert the labels of trajectories that cover v_p to $S(o_i)$ (lines 16–17); Otherwise, we terminate the expansion process.

3.2 Alignment of Trajectory Candidates

Having trajectory candidates of v_s, v_e, and O, we need to find qualified alignment of these trajectory candidates. Our high-level idea works as follow. First, we derive all qualified partitionings for location set O. Each qualified partitioning, denoted by $\mathbb{P}$ ($|\mathbb{P}| \leq M$), is a set of disjoint location subsets O_s of O, and the union of these subsets is O. Second, for each partitioning we generate trajectory candidates associated with each location subset $O_s \in \mathbb{P}$. Finally, based on those trajectory candidates, we generate qualified sub-trajectory alignments regarding each possible partitioning.

We present the data structure we used for processing trajectory candidate alignment. In order to represent and index each trajectory candidate associated with each location subset $O_s \in \mathbb{P}$, we maintain a subset trajectory candidate label of each trajectory τ associated with each location subset O_s. The label is denoted by $l(\tau, O_s) = \langle e, H, s_u \rangle$ where e is the entry of τ, H is a hashmap where keys are locations in O_s and values are their corresponding vertices visited by the expansion algorithm, and s_u denotes the upper bound of similarity between O and τ, which is computed by Equation 4.

$$l(\tau, O_s).s_u = \frac{|O| + \sum_{\langle o_i, v_p \rangle \in H} e^{-d(o_i,v_p)} - |O_s|}{|O|}, \quad (4)$$

Algorithm 1: TrajCandidatesGen

Data: Start location v_s, end location v_e, query location set O, similarity threshold θ, travel time interval T
Result: $\mathcal{S} = \{S(v_s), S(v_e), S(o_1), ..., S(o_{|O|})\}$

1 **for** *each* $S(\cdot)$ *in* $\mathcal{S}$ **do**
2 $S(\cdot) \leftarrow \emptyset$;
3 **for** *each trajectory* τ *that covers* v_s **do**
4 $p \leftarrow \tau.p_1$;
5 **if** $p.t_1$ *is after* $T.t_s$ **then**
6 $S(v_s)$.add($l_{v_s}(\tau, v_s)$);
7 **for** *each trajectory* τ *that covers* v_e **do**
8 $p \leftarrow \tau.p_{|\tau|}$;
9 **if** $p.t_{|\tau|}$ *is before* $T.t_e$ **then**
10 $S(v_e)$.add($l_{v_e}(\tau, v_e)$);
11 **for** *each location* $o_i \in O$ **do**
12 **while** NetworkExpansion(o_i).hasNext() **do**
13 $v_p \leftarrow$NetworkExpansion(o_i).next();
14 $sim_i^{ub} \leftarrow |O| + e^{-d_{min}(o_i, v_p)} - 1$;
15 **if** $sim_i^{ub} \geq \theta$ **then**
16 **for** *each* τ *that covers* v_p **do**
17 $S(o_i)$.add($l_{o_i}(\tau, v_p)$);
18 **else**
19 **break**;
20 **return** $\mathcal{S}$;

where v_p denotes the vertex visited by the network expansion from o_i.

Algorithm of alignment. Algorithm 2 presents the pseudo code of trajectory alignment. We evaluate each qualified partitioning $\mathbb{P}$ of O where the number of subsets ranges from 1 to M. For each location subset $O_s \in \mathbb{P}$ we evaluate trajectory candidates associated with each $o_i \in O_s$ and derive trajectory candidates associated with O_s (lines 4–10). For each qualified trajectory candidate of O_s, we generate and index its label (lines 9–10). Next, we merge all subset trajectory candidates in a bottom-up manner. We evaluate each alignment (sequence) of subsets $\mathcal{P}$, denoted by $\mathcal{Q}$, by merging candidates from adjacent subsets hierarchically (lines 11–28). Specifically, given an adjacent subset pair $\langle O_s, O_s^{'} \rangle$ we evaluate each indexed subset trajectory candidate pair, τ_i and $\tau_i^{'}$, and check if they are mergeable (lines 19–21). In particular, τ_i and $\tau_i^{'}$ are mergeable if they have intersection vertex and their arrival timestamps on the intersection vertex meet temporal alignment requirement. If they are mergeable, we generate the label for τ_c, which is an aligned trajectories of τ_i and $\tau_i^{'}$ (lines 22–23). Next, we update $\mathcal{Q}$ by merging O_s and $O_s^{'}$ (line 24). The merging process terminates when $\mathcal{Q}$ only has one set. Upon termination, we have generated labels associated with the entire location set. Here, we add all trajectories that cover v_s and v_e into the result pool.

Algorithm 2: TrajAlignment

Data: Trajectory candidate sets $\mathcal{S}$, source location v_s, destination location v_e, location set O, similarity threshold θ, travel time interval T, maximum alignment count M
Result: Result sub-trajectory alignments R

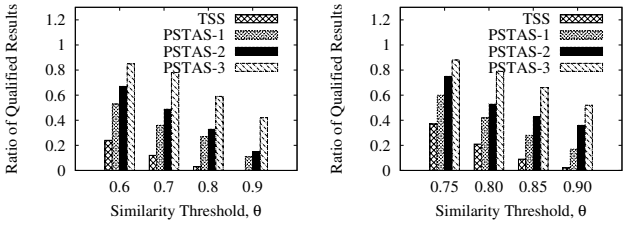
1 $k \leftarrow 0$;
2 **while** $k \leq M$ **do**
3 **for** *each partitioning* $\mathbb{P}$ *of* O *where* $|\mathbb{P}| = k$ **do**
4 **for** *each subset* O_s *in* $\mathbb{P}$ **do**
5 $S(O_s) \leftarrow \emptyset$;
6 **for** *each* $o_i \in O_s$ **do**
7 **for** *each* $\tau \in S(o_i)$ **do**
8 Generate $l(\tau, O_s)$;
9 **if** $l(\tau, O_s).s_u \geq \theta$ **then**
10 $S(O_s)$.add($l(\tau, O_s)$);
11 **for** *each alignment* $\mathcal{P}$ *of* $\mathbb{P}$ **do**
12 $\mathcal{Q} \leftarrow \mathcal{P}$;
13 Initialize O_c;
14 **while** $|\mathcal{Q}| > 1$ **do**
15 **for** *each disjoint adjacent subset pair* $\langle O_s, O_s^{'} \rangle$ *in* $\mathcal{Q}$ **do**
16 **for** *each label pair* $\langle l_i, l_i^{'} \rangle$ *where* $l_i \in O_s$ *and* $l_i^{'} \in O_s^{'}$ **do**
17 $O_c \leftarrow O_s \cup O_s^{'}$;
18 $\tau_i \leftarrow l_i.e$; $\tau_i^{'} \leftarrow l_i^{'}.e$;
19 **if** τ_i *and* $\tau_i^{'}$ *are mergeable* **then**
20 $\tau_c \leftarrow \tau_i + \tau_i^{'}$;
21 Generate $l(\tau_c, O_c)$;
22 **if** $l(\tau_c, O_c).s_u \geq \theta$ **then**
23 $S(O_c)$.add($l(\tau_c, O_c)$);
24 $\mathcal{Q}$.remove(O_s); $\mathcal{Q}$.remove($O_s^{'}$); $\mathcal{Q}$.add(O_c);
25 **for** *each label* l *in* $S(O_c)$ **do**
26 $\tau \leftarrow l.e$;
27 **if** τ *covers* v_s *and* v_e **then**
28 R.add(τ);
29 **return** R;

4 Experiments

4.1 Experiment Settings

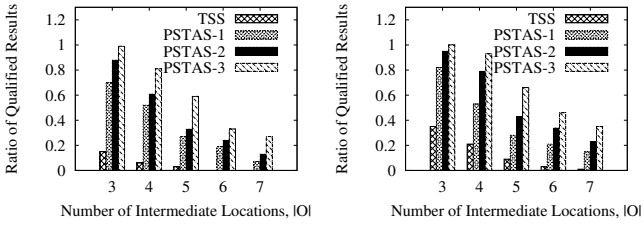
Datasets. We apply the following two road networks: Beijing Road Network (BN) and the New York Road Network (NYN)[2]. BN consists of ∼28K vertices and ∼27K edges. NYN consists of ∼95K vertices and ∼261K edges. For BN, we use taxi trajectory data collected by the T-drive project [Yuan *et al.*, 2013], which consists of 800K trajec-

[2]https://lab-work.github.io/data/



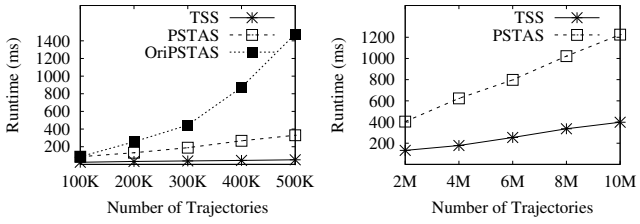
(a) BN (b) NYN

Figure 2: Effect of similarity threshold (effectiveness)



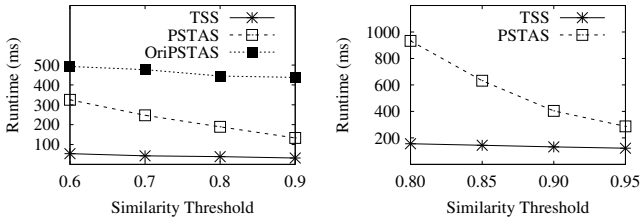
(a) BN (b) NYN

Figure 3: Effect of the number of locations (effectiveness)



(a) BN (b) NYN

Figure 4: Effect of the number of trajectories



(a) BN (b) NYN

Figure 5: Effect of similarity threshold, θ

tories. For NYN, we use taxi trip data from New York[3], which consists of 800M trips. Note that each taxi trip from New York only contains pick-up and drop-off locations and timestamps. As such, we generate a trajectory by deriving the shortest network path from the pick-up location to the drop-off location.

Implementations. The road networks, trajectories, and indices are memory resident. All algorithms are implemented in Java and run on a server with two Intel® Xeon® Processors Gold 5120 and 64GB RAM. Unless stated otherwise, experiment results are averaged over 100 independent trials using different v_s, v_e, and O for efficacy and efficiency evaluations.

Baselines. We evaluate the following methods. (1) *Trajectory Similarity Search without Alignment (TSS)*: Given a collection of trajectories, TSS returns a set of existing trajectories satisfying the similarity and temporal requirements. Note that TSS is equivalent to PSTAS where $M = 0$. (2) *Parallel Sub-Trajectory Alignment Search (PSTAS)*: Algorithm 1 and Algorithm 2. (3) *PSTAS without Pruning Technique (OriPSTAS)*: PSTAS without calculating the similarity upper bound for filtering unqualified trajectory candidates at an early stage.

Evaluation metrics. We use the following metrics to evaluate the effectiveness and efficiency of each method. (1) *Ratio of qualified results*: Given N independent trials, let N_q be the number of trials that return at least one qualified trajectory alignment. The ratio of qualified results is defined by $\frac{N_q}{N}$. (2) *Runtime*: The average time cost of query processing. The parameter settings are listed in Table 1.

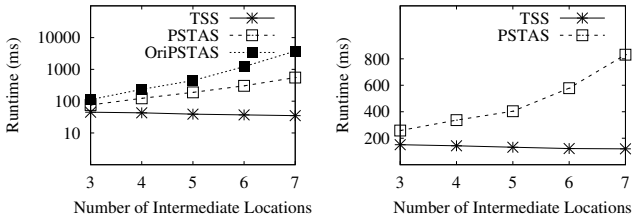
[3]https://data.cityofnewyork.us/browse?q=trips

	BN	**NYN**
Number of trajectories	100K–500K / default 300K	2M–10M / default 2M
Maximum number of alignments M	1–3 / default 2	1–3 / default 2
Number of intermediate locations $\|O\|$	3–7 / default 5	3–7 / default 5
Similarity threshold θ	0.60–0.90 / default 0.80	0.80–0.95 / default 0.90
Travel time interval T	randomly generated from 0.5 to 3 hours	randomly generated from 0.5 to 3 hours
Thread count	16–48 / default 48	16–48 / default 48

Table 1: Parameter settings

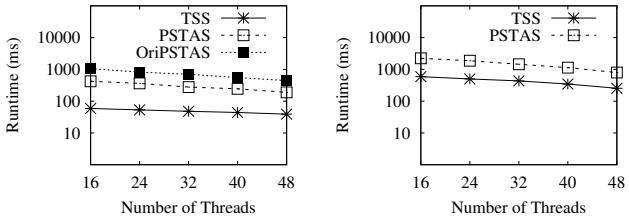
4.2 Experiment Results

Varying similarity threshold (effectiveness). First, we evaluate the ratio of individual trails that return qualified trajectories. We vary the similarity threshold θ on BN and NYN. In Figure 2, the "x" from "PSTAS-x" denotes the value of M. As we increase the similarity threshold, all methods exhibit a decreasing trend regarding the ratio of results with qualified trajectories. PSTAS-x performs consistently better than TSS. In particular, when we let θ be 0.8 in BN, TSS only has 3% trails (queries) that return qualified trajectories. When we increase θ to 0.9, none of the trails find qualified trajectories. In contrast, PSTAS-1, PSTAS-2, and PSTAS-3 have 27%, 33%, and 59% trails that return qualified results, respectively, when we set θ to 0.8. Such contrast clearly demonstrates that PSTAS, even with a small number of trajectory alignments, is substantially more likely to find high-quality results (i.e.,



(a) BN (b) NYN

Figure 6: Effect of the number of locations



(a) BN (b) NYN

Figure 7: Effect of thread counts

trajectories that are highly similar to query locations) than traditional trajectory similarity search method.

Varying the number of locations (effectiveness). Next, we evaluate the ratio of individual trails that return qualified trajectories as we vary the number of intermediate locations ($|O|$). From Figure 3, we see that all methods perform worse as we increase the number of intermediate locations.

Varying the number of trajectories. This set of experiments evaluates the efficiency of each method as we vary the number of trajectories. Note that OriPSTAS fails to answer the STA problem within interaction time on NYN. Thus, we only report the performance of OriPSTAS on BN in remaining experiments. From Figure 4 we see that all methods perform worse as we increase the number of trajectories in datasets. In particular, the time cost of OriPSTAS exhibits an exponential increasing trend as we increase the number of trajectories. In contrast, the time cost increment regarding PSTAS is close to a linear trend, which confirms the effectiveness of our pruning technique in Algorithms 1 and 2.

Varying similarity threshold θ. This set of experiments investigates the efficiency as we vary the similarity threshold θ. From Figure 5 we see that the performance of TSS and OriPSTAS is relatively consistent as we vary the value of θ. In contrast, the time cost of PSTAS exhibits a decreasing trend as we increase θ. This can be explained by the effect of our pruning technique. More unqualified trajectory candidates can be pruned when we increase θ, thus the time cost can be reduced substantially.

Varying the number of locations $|O|$. We proceed to study the efficiency of each method as we vary the number of intermediate locations $|O|$. Figure 6 shows that the time cost of TSS is consistent as we vary $|O|$. While the time costs of PSTAS and OriPSTAS increase when the number of locations increases. The reason is that a higher value of $|O|$ indicates more partitionings to be evaluated.

Effect of thread counts. Finally, we study the effect of thread count on the efficiency of all methods. Figure 7 show that all methods exhibit lowered time cost as we increase the number of threads. The results show that all methods are capable of parallel processing.

5 Related Work

Trajectory similarity search has been extensively investigated by existing studies. In particular, given a collection of trajectories, the problem of trajectory similarity search aims to find trajectories that meet query arguments defined by users [Frentzos *et al.*, 2007; Zheng *et al.*, 2013; Shang *et al.*, 2012; Driemel *et al.*, 2020; Zheng *et al.*, 2017; Xie *et al.*, 2017; Yuan and Li, 2019; Shang *et al.*, 2014]. The query arguments can be spatial requirement [Chen *et al.*, 2010], temporal requirement [Shang *et al.*, 2014], and text-based requirements [Shang *et al.*, 2012; Zheng *et al.*, 2013]. The problem of trajectory similarity search by multiple locations was first investigated by Chen et al. [Chen *et al.*, 2010]. Specifically, the problem takes a location set from road networks as argument and finds trajectories that are spatially close to the query location set on the basis of some specific metrics. Existing studies on this matter aim to develop spatial and network indexing structures for effectively organizing a large volume of trajectory data [Zheng *et al.*, 2018; Zhao *et al.*, 2018; Liu *et al.*, 2016; Liu *et al.*, 2014; Chen *et al.*, 2020]. However, the aforementioned studies are developed for supporting the retrieval of existing trajectories. They do not consider the problem of deriving new trajectories by splitting and aligning existing trajectories. [Chen *et al.*, 2019] target the problem of route recommendation by combining past travel routes. However, their proposal does not consider the temporal dimension in route combination.

6 Conclusions

We define and study the problem of Sub-Trajectory Alignment (STA), aiming to address the limitation of low result quality for traditional trajectory similarity search with insufficient trajectory data. To solve the STA problem, we develop a PSTAS algorithm consisting of two phases: (1) Generation of trajectory candidates through network expansion and (2) alignment of trajectory candidates through bottom-up merging. The experiment results confirm that: (1) Answering the STA problem is substantially more likely to find high-quality trajectory results in comparison to answering existing trajectory similarity search problem; (2) Our PSTAS algorithm is capable of answering the STA problem over a collection of up to 10M trajectories with user interaction time.

Acknowledgments

This work was supported by the NSFC (U2001212, 62032001, and 61932004).

References

[Chen *et al.*, 2010] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou, Yu Zheng, and Xing Xie. Searching trajectories by locations: an efficiency study. In *SIGMOD*, pages 255–266, 2010.

[Chen *et al.*, 2019] Lisi Chen, Shuo Shang, Christian S. Jensen, Bin Yao, Zhiwei Zhang, and Ling Shao. Effective and efficient reuse of past travel behavior for route recommendation. In *KDD*, 2019.

[Chen *et al.*, 2020] Lisi Chen, Shuo Shang, and Tao Guo. Real-time route search by locations. In *AAAI*, pages 574–581, 2020.

[Dijkstra, 1959] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Math*, 1:269–271, 1959.

[Driemel *et al.*, 2020] Anne Driemel, Petra Mutzel, and Lutz Oettershagen. Spatio-temporal top-k similarity search for trajectories in graphs. *CoRR*, abs/2009.06778, 2020.

[Frentzos *et al.*, 2007] E. Frentzos, K. Gratsias, and Y. Theodoridis. Index-based most similar trajectory search. In *ICDE*, pages 816–825, 2007.

[Liu *et al.*, 2014] Kuien Liu, Yaguang Li, Jian Dai, Shuo Shang, and Kai Zheng. Compressing large scale urban trajectory data. In *CloudDP@EuroSys*, pages 3:1–3:6. ACM, 2014.

[Liu *et al.*, 2016] Jiajun Liu, Kun Zhao, Philipp Sommer, Shuo Shang, Brano Kusy, Jae-Gil Lee, and Raja Jurdak. A novel framework for online amnesic trajectory compression in resource-constrained environments. *IEEE Trans. Knowl. Data Eng.*, 28(11):2827–2841, 2016.

[Shang *et al.*, 2012] Shuo Shang, Ruogu Ding, Bo Yuan, Kexin Xie, Kai Zheng, and Panos Kalnis. User oriented trajectory search for trip recommendation. In *EDBT*, pages 156–167, 2012.

[Shang *et al.*, 2014] Shuo Shang, Ruogu Ding, Kai Zheng, Christian S. Jensen, Panos Kalnis, and Xiaofang Zhou. Personalized trajectory matching in spatial networks. *VLDB J.*, 23(3):449–468, 2014.

[Shang *et al.*, 2017] Shuo Shang, Lisi Chen, Zhewei Wei, Christian S. Jensen, Kai Zheng, and Panos Kalnis. Trajectory similarity join in spatial networks. *PVLDB*, 10(11):1178–1189, 2017.

[Xie *et al.*, 2017] Dong Xie, Feifei Li, and Jeff M. Phillips. Distributed trajectory similarity search. *Proc. VLDB Endow.*, 10(11):1478–1489, 2017.

[Yuan and Li, 2019] Haitao Yuan and Guoliang Li. Distributed in-memory trajectory similarity search and join on road network. In *ICDE*, pages 1262–1273, 2019.

[Yuan *et al.*, 2013] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. T-drive: Enhancing driving directions with taxi drivers' intelligence. *IEEE Trans. Knowl. Data Eng.*, 25(1):220–232, 2013.

[Zhao *et al.*, 2018] Yan Zhao, Shuo Shang, Yu Wang, Bolong Zheng, Quoc Viet Hung Nguyen, and Kai Zheng. REST: A reference-based framework for spatio-temporal trajectory compression. In Yike Guo and Faisal Farooq, editors, *KDD*, pages 2797–2806, 2018.

[Zheng *et al.*, 2013] Kai Zheng, Shuo Shang, Nicholas Jing Yuan, and Yi Yang. Towards efficient search for activity trajectories. In *ICDE*, pages 230–241, 2013.

[Zheng *et al.*, 2017] Kai Zheng, Bolong Zheng, Jiajie Xu, Guanfeng Liu, An Liu, and Zhixu Li. Popularity-aware spatial keyword search on activity trajectories. *World Wide Web*, 20(4):749–773, 2017.

[Zheng *et al.*, 2018] Bolong Zheng, Haozhou Wang, Kai Zheng, Han Su, Kuien Liu, and Shuo Shang. Sharkdb: an in-memory column-oriented storage for trajectory analysis. *World Wide Web*, 21(2):455–485, 2018.