Knowledge and Information Systems (2020) 62:1279–1312
https://doi.org/10.1007/s10115-019-01382-x

REGULAR PAPER

Trajectory splicing

Qiang Lu[1] · **Rencai Wang**[1,3] · **Bin Yang**[2] · **Zhiguang Wang**[1]

Received: 23 September 2018 / Revised: 25 June 2019 / Accepted: 30 June 2019 / Published online: 18 July 2019

Abstract

With contin ed de elo ment of location-ba ed tem , la ge amo nt of t ajecto ie become a ailable, hich eco d mo ing object location ac o time. If the t ajecto ie collected b different location-ba ed tem come f om the ame mo ing object, the a e *spliceable trajectories*, hich cont ib te to e e enting holi tic beha io of the mo ing object. In thi a e , e con ide ho to efficientl identif liceable t ajecto ie . Mo e ecificall , e fi t fo mali e a liced model to ca t e liceable t ajecto ie he e thei time a e di joint, and the di tance bet een them a e clo e. Ne t, to efficientl im lement the model, e de ign th ee com onent : a di joint time inde , a di ected ac clic g a h of b-t ajecto location connection , and t o lice algo ithm . The di joint time inde a e a di joint time et of each t ajecto fo e ing di joint time t ajecto ie efficientl . The di ected ac clic g a h cont ib te to di co e ing g o of liceable t ajecto ie . Ba ed on the identified g o , the lice algo ithm *findmaxCTR* find ma imal g o containing all liceable t ajecto ie . Altho gh the lice algo ithm i efficient in ome actical a lication , it nning time i e onential. The efo e, an a o imate algo ithm *findApproxMaxCTR* i o o ed to find t ajecto ie hich can be liced ith each othe ith a ce tain obabilit ithin ol nomial n time. Finall , e e eriment on t o data et demon t ate that the model and it com onent a e effecti e and efficient.

Keywords T ajecto com tation · T ajecto f ion · T ajecto eco e · T ajecto linking

✉ Qiang L
l iang@c .ed .cn

Rencai Wang
c ang3@ifl tek.com

Bin Yang
b ang@c .aa .dk

Zhig ang Wang
c ang g@c .ed .cn

1 Beijing Ke Lab of Pet ole m Data Mining, China Uni e it of Pet ole m-Beijing, Beijing, China

2 De a tment of Com te Science, Aalbo g Uni e it , Aalbo g, Denma k

3 IFLYTEK CO.,LTD, Hefei, China

1 Introduction

Information technology is almost everywhere in our daily life, which collects various information from different digital devices [4,10]. Specially, the location-based systems based on mobile devices, such as GPS, mobile phones, and near-field communication (NFC) terminals, generate large amounts of trajectories of moving objects. Usually, each individual system uses its unique *ID* code to identify each trajectory. For example, a mobile phone network identifies a trajectory by its telephone number, while an NFC system identifies it by its device-id. Since multiple systems may capture a same moving object at different times and places, each system gathers the object's partial trajectories. Recovering a **complete trajectory** of a moving object from the partial trajectories collected in various systems, named **trajectory splicing**, is essential for many applications, such as anomaly behavior detection [21,22], data fusion, and trajectory data mining [46]. The following case shown in Fig. 1 elaborates trajectory splicing.

Every weekday, Alice and Bob go to work by walking and taking the subway, as shown in Fig. 1. Their movements generate six partial trajectories: W1, S1, O1, W2, S2, and O2, where the mobile sports software captures W1 and W2; the subway check-in systems capture S1 and S2; the office check-in systems capture O1 and O2. Their complete trajectories can be recovered based on spatiotemporal locations of these partial trajectories. For example, S2 is more likely to splice with W2 than W1, because end points' spatial positions of S2 are closer to those of W2, and the time interval of S2 [8:23,9:14] can be embedded into the time gaps of W2 (8:20, 9:16). Similarly, O2 can splice with W2. So, connecting W2, S2, and O2 can recover Bob's whole trajectory.

According to the above case, finding a group of spliceable trajectories must satisfy the following three requirements. The first is the **disjoint time constraint** that requires that time intervals of spliceable trajectories in the group should not overlap with each other. The second is the **spatial constraint** that requires that the distance between their end points should be nearby with each other. The third is the **maximal group constraint** that requires that the group of spliceable trajectories should be maximal and should not be contained by other groups. That means connecting as many spliceable trajectories as possible to recover a complete trajectory.

However, it is non-trivial to find spliceable trajectories to satisfy the above constraints owing to the following three challenges. The first challenge is that the process of finding trajectories that satisfy the disjoint time constraint is very time-consuming. The process includes two steps: querying sub-trajectories in all time gaps of a trajectory and counting the number of sub-trajectories that belong to the same trajectory. For example, in Fig. 1, W2 has three time gaps: $(-\infty$

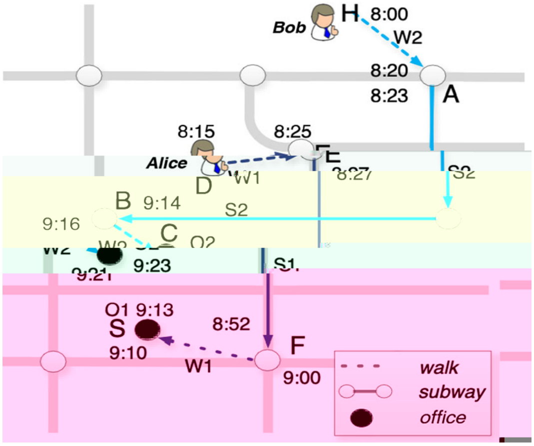


Fig. 1 The case of trajectory splicing

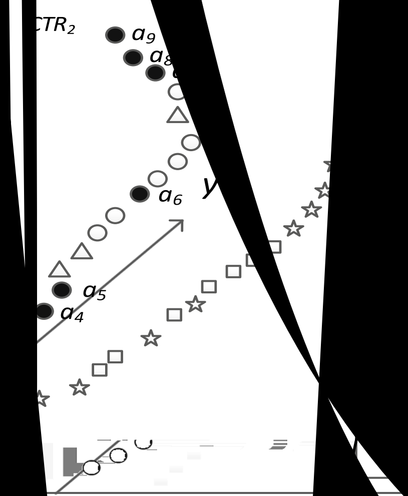
which connects two trajectories without using other spliceable trajectories. The other is the **indirect splice** which connects two trajectories by using other spliceable trajectories. For example, in Fig. 1, W2 and S2 are connected directly, while W2 and O2 are connected by S2. The indirect splices make the process of splicing trajectories complicated because it needs to find other trajectories to determine whether the two trajectories can be connected or not. To the best of our knowledge, known group pattern mining [8,9,19,24–27,36,45] or trajectory clustering [24,25] cannot find groups of spliceable trajectories, because they discover groups of trajectories according to the similarity between them rather than the relation of direct (indirect) splice. Although fuzzy trajectory linking [38] is close to the challenge, it can only find two direct splice trajectories and is not suitable for mining multiple trajectories that are the direct or indirect splice.

The third challenge is that it must find as many spliceable trajectories of a moving object as possible. In general, if a method wants to acquire a group of spliceable trajectories which are not contained by other groups, it needs to traverse all possible combinations of spliceable trajectories for a moving object. For example, in the above case, to recover Bob's trajectory, these groups, such as {W2, S2}, {W2, O2}, {S2, O2}, and {W2, S2, O2}, must be traversed. Namely, it needs to find a group of spliceable trajectories which most compactly fill up a specific spatiotemporal range. So, it is a bin-packing problem and is NP-hard [23]. The design of an approximation scheme or heuristic method is the key to deal with the problem.

In order to deal with the above challenges, a spliced model is defined to formalize the above requirements of spliceable trajectories. Based on the spliced model, trajectories are segmented into sub-trajectories according to a speed threshold. A B^+-tree [7] is used to save these sub-trajectories. For speeding up the process of finding disjoint time sets, the index of disjoint time called **DT-index** is constructed to keep intermediate results of searching the disjoint time sets in each time slice. Moreover, the DT-index is a multi-resolution structure like a quadtree and can save intermediate results of time slices with different lengths, supporting queries with different time intervals. For example, assuming that the DT-index consists of intermediate results of one, two, and four days, if a user time interval is 4.5 days, the DT-index can find disjoint time sets within the four days, and the B^+-tree can find disjoint time sets within the 0.5 days. Based on the above two indexes, an algorithm *e DT TR* is proposed to obtain all disjoint time sets within a specific time interval.

In order to find spliceable trajectories, a directed acyclic graph of sub-trajectory location connections called ***STLC-DAG*** is created to connect sub-trajectories by their time and locations. Once the algorithm ***createSTLC-DAG*** has created the graph, it can obtain the spliceable sets of trajectories that can splice with a specific trajectory. For example, in the above case, the algorithm can find S2's spliceable set {W2}, W2's {S2, O2}, and O2's {W2}. Moreover, these spliceable sets form a **splice graph**, where each node is a trajectory, and the edge between two nodes represents that the two trajectories are spliceable. For instance, the node S2 has one edge which connects the node W2, and W2 has edges which connect S2 and O2. Thus, in the splice graph, a clique is a group of spliceable trajectories. For addressing the third challenge, an algorithm ***FindMaxCTR*** is proposed to find all maximal e 1 N

tion	Definition
	A *TR* databa e
	A am le oint ⟨*id*, *lc*,
	The *i*th *TR* in *Ω*
	A TR_i di joint time et
	A et of *TR* that can be
	The *j*th b-t ajecto of
	he time inte al of e
	he n mbe of *TR* in *T*
	e n mbe of all *STR* in *T*
	om lete t ajecto con i t
	n the fi t element in e
	n the la t element in e
	E clidean di tance bet e
	clidean di tance bet e
	ne inte al of a b-t aj
	e inte al et of t ajec
	T bet een t o b-t aj
	Th et of $ti(TR_i)$



moving objects: $CTR_1 = \{TR_A, TR_B, TR_C\}$, which includes the trajectories with identifiers A, B, and C, and $CTR_2 = \{TR_D, TR_E\}$, which includes the trajectories with identifiers D and E.

In a trajectory, two sample points, p_i and p_{i+1}, are **connectable** if $speed(p_i, p_{i+1}) \geq e$, where e is a speed threshold and

$$speed(p_i, p_{i+1}) = \frac{d(p_i, p_{i+1})}{|p_{i+1}.t - p_i.t|} \quad (1)$$

where $d(p_i, p_{i+1})$ returns the Euclidean distance between sample points p_i and p_{i+1}. Given a sequence of sample points in a trajectory TR_i, if any two consecutive sample points in the sequence are connectable, the sequence is **connectable** in that it shows one continuous movement. Moreover, if other connectable sequences do not contain a connectable sequence, the connectable sequence is called **sub-trajectory** (denoted as ***STR***). In particular, we use STR_i to denote the jth sub-trajectory in trajectory TR_i. For example, trajectory TR_A in Fig. 2 has 4 sub-trajectories: $STR_A^1 = \langle a_1, a_2, a_3\rangle$, $STR_A^2 = \langle a_4, a_5\rangle$, $STR_A^3 = \langle a_6\rangle$, and $STR_A^4 = \langle a_7, a_8, a_9\rangle$. A sub-trajectory is the **atomic** computational unit in this paper.

The **time interval** of the sub-trajectory, denoted as $\boldsymbol{ti}(STR)$, is $[first(STR).t, last(STR).t]$, where the functions $first(\cdot)$ and $last(\cdot)$ return the first and last sample points in the sub-trajectory STR, respectively. The **time interval** of the trajectory is the set of time intervals of all its sub-trajectories, denoted as $\boldsymbol{ti(TR_i)} = \bigcup_{STR_i^j \in TR_i} ti(STR_i^j)$.

The **gap** between two sub-trajectories STR_i^j and STR_m^n, denoted as $gap(STR_i^j, STR_m^n)$, is defined by Eq. 2.

$$gap(STR_i^j, STR_m^n) = (last(STR_i^j).t,\ first(STR_m^n).t) \quad (2)$$

Moreover, the **gap** of trajectory TR_i in the time interval T, denoted as $\boldsymbol{gap(TR_i)}$, is defined by Eq. 3.

$$gap(TR_i) = T - ti(TR_i) = T - \bigcup_{STR_i^j \in TR_i} ti(STR_i^j) \quad (3)$$

For example, the time interval of trajectory TR_A, denoted as $ti(TR_A)$, is $\{[t_1, t_2], [t_3, t_4], [t_5, t_5], [t_6, t_7]\}$. Given $T = [t_0, t_8]$, we have $gap(TR_A) = \{(t_0, t_1), (t_2, t_3), (t_4, t_5), (t_5, t_6), (t_7, t_8)\}$.

2.2 Spliceable trajectories

If two trajectories TR_i and TR_j can be spliced into a complete trajectory, they must meet the **disjoint time constraint** that requires that their interval times should not overlap each other, namely $ti(TR_i) \subset gap(TR_j)$. Given a trajectory TR_i, all the trajectories that meet the disjoint time constraint with TR_i constitute the **disjoint time set** of TR_i, denoted as DT_i. In Fig. 2, since $ti(TR_B) \subset gap(TR_A)$ and $ti(TR_C) \subset gap(TR_A)$, we have $DT_A = \{TR_B, TR_C\}$.

In addition to the aforementioned temporal constraint, if TR_i and TR_j are spliceable, they must also meet the **spatial constraint**, meaning that the sub-trajectories from TR_i and TR_j must be close enough to each other. To formally define the spatial constraint, we introduce two concepts: ***spliceable pair*** and ***spliceable trajectories***.

Definition 1 Given two sub-trajectories STR_i^j and STR_m^n from two trajectories, respectively, and a distance threshold γ, if they do not overlap each other on the time dimension and the distance between them is less than γ[1], the two sub-trajectories STR_i^j and STR_m^n form a ***spliceable pair***, denoted as $\langle STR_i^j, STR_m^n \rangle$.

Definition 2 Given some trajectories, if the sub-trajectories in the given trajectories can constitute a sub-trajectory sequence $\langle STR_i^j, \ldots, STR_m^n \rangle$ such that any two neighbor sub-trajectories are a spliceable pair, these trajectories are called ***spliceable trajectories***.

Based on the above two definitions, we first introduce the concept of ***complete trajectory*** to formulate the maximal group constraint, which ensure that the groups of spliceable trajectories should not be contained by other groups. Then, we define the ***splice degree*** to quantify the complete trajectory.

Definition 3 If other groups of spliceable trajectories do not contain a group of spliceable trajectories, the group forms a **complete trajectory**, denoted as ***CTR***.

Definition 4 The ***splice degree***, which consists of two factors: the ratio of the sum of the distance between different trajectories to the distance of CTR and the ratio of the sum of time gaps to the time interval of CTR, is used to quantify the compactness level of connection between trajectories in a CTR, defined by Eq. 4.

$$dg(CTR) = \frac{\sum_{\langle STR_i^j, STR_m^n \rangle \in CTR} d(STR_i^j, STR_m^n)}{distance(CTR)} \times \frac{\sum_{\langle STR_i^j, STR_m^n \rangle \in CTR} gap(STR_i^j, STR_m^n)}{time(CTR)} \quad (4)$$

where $\langle STR_i^j, STR_m^n \rangle$ is a *spliceable pair* in the CTR; $d(STR_i^j, STR_m^n)$ is the distance between two sub-trajectories STR_i^j and STR_m^n; $distance(CTR)$ is the sum of distances between consecutive sample points in CTR, namely $distance(CTR) = \sum_{p_i \in CTR} d(p_i, p_{i+1})$, in which p_i and p_{i+1} are two consecutive sample points in the CTR; $time(CTR) = last(CTR).t - first(CTR).t$.

Based on the definition, $dg(CTR) \in (0, 1)$ and the smaller the splice degree $dg(CTR)$, the closer trajectories in the complete trajectory CTR. For example, in Fig. 2, assuming that the distance factors in Alice and Bob are the same value 0.02, $dg(\text{Alice}) = 0.02 \times (((8:27 - 8:25) + (9:00 - 8:52) + (9:13 - 9:10))/(9:13 - 8:15)) \approx 0.0448$, and $dg(\text{Bob}) = 0.02 \times ((8:23 - 8:20) + (9:16 - 9:14) + (9:23 - 9:21)/(9:23 - 8:00)) \approx 0.0017$. So, due to $dg(\text{Bob}) < dg(\text{Alice})$, the complete trajectory of Bob is better than that of Alice.

2.3 Problem definition

According to the above definitions, we formulate the problem of trajectory splicing by the trajectory splicing query.

Definition 5 From a dataset of trajectories, according to a user time interval, the ***trajectory splicing query*** discovers a complete trajectory sequence $CTRS = \langle CTR_1, \ldots, CTR_n \rangle$, where each complete trajectory CTR is ranked by its *splice degree*.

[1] Namely $(ti(STR_m^n) \subset gap(STR_i^j, STR_i^{j+1})) \cap (ti(STR_i^j) \subset gap(STR_m^{n-1}, STR_m^n)) \cap (d(last(STR_i^j), first(STR_m^n)) \le \gamma)$.

[redacted]

$_i$ in

(5)

hat a ear in the

$_A^{d} = \{B, C, D, E\}$

$E\}, \neg DT_A^{2,d} = \{D, E\}$

n Fig. 4.

a DT_i of each trajectory TR_i

proof in Appendix A).

$\cup \ldots \cup DF_i^{n,d}]$ (6)

Fig.

where $|T| = n \times d$, d is the length of the time slice, n represents the nth time slice, and P_i is a set which contains all trajectories that appear in T except the trajectory TR_i. For example, in Fig. 4, if $T = [0, 3d]$, $DT_D(T) = P_D^{0,3d} - [(P_D^{0,3d} - DT_D^{1,d}) \cup DF_D^{2,d} \cup DF_D^{3,d}] = \{A, B, C, E\} - [(\{A, B, C, E\} - \{E\}]) \cup \{A\} \cup \phi] = \{E\}$.

If T is too long, there are many time slices in T, and Eq. 6 contains many union operations of DF so that the computation of Eq. 6 is time-consuming. To alleviate the situation, we partition the time dimension into multiple levels of time slices. For instance, one level of time slices is a day, and another level is a week or month. So, if $|T|$ is one month, Eq. 6 can be computed by only one DF on the month level of time slices rather than by about 30 DF on the day level.

(2) *The structure of disjoint time index*

Based on the above analysis, we design the disjoint time index (called ***DT***-index), which includes a *DT-tree* and a *DF-tree* that save the disjoint time sets DT of each trajectory and its recomputation DF on different levels of time slices, respectively, as shown in Fig. 4. The two trees have the same structure. The *DT-tree* (*DF-tree*) consists of a single root node, leaf nodes, and non-root, non-leaf nodes. The detailed data structures of these nodes are as follows.

A root node, which may have multiple children, saves their *ID*s. An *ID* is both a time interval and a filename, when querying a time interval T, its children and their files are located quickly.

A leaf node stores pairs of $\langle i, DT_i \rangle$ or $\langle i, DF_i \rangle$ in a specific time slice. For example, in Fig. 4, $DT^{3,d}$ records pairs $\langle A, \{B, C\} \rangle$, $\langle B, \{A, C\} \rangle$ and $\langle C, \{A, B\} \rangle$.

A non-root, non-leaf node only has two children. It stores its children IDs and pairs of $\langle i, DT_i \rangle$ or $\langle i, DF_i \rangle$, where DT_i or DF_i can be computed by Eqs. 6 or 5, respectively.

3.2 Processing query

With the B^+-*tree* and the DT-index, we implement an algorithm *QueryDTsTR* which quickly finds the disjoint time set *DT* of each trajectory and all sub-trajectories (denoted as *STRSet*) in a time interval T, as shown in Algorithm 1.

Algorithm 1: *queryDTsTR*

Input: B^+*-tree, DT-Index, T*
Output: *DT(T), STRSet*
1 *STRSet*, $DT(T_1), R(T_1), R(T_2)$, P=*readsTR*(B^+-*tree*, T) ;
2 $DT(T_2)$= *Equation 7* ;
3 $DT = (DT(T1) \cup R(T_1)) \cap (DT(T2) \cup R(T_2))$;
4 return *DT,STRSet* ;

The query time interval T consists of two parts: One is a set of two time intervals without any time slices in the DT-index, denoted as $T_1 = \{t_1, t_2\}$; the other is the time interval that contains n time slices in the DT-index, denoted as T_2. For example, given $T = [8:35\ 11:25]$ and the minimal time slice is an hour, $T_1 = \{[8:35\ 9:00], [11:00\ 11:25]\}$, and $T_2 = [9:00\ 11:00]$. With the B^+-*tree*, it is easy to find all trajectories P and their sub-trajectories *STRSet* in T. Meanwhile, searching these sub-trajectories can obtain a trajectory set $R(T_1)$ where each trajectory appears in T_1 but not in T_2, a trajectory set $R(T_2)$ where each trajectory appears in T_2 but not in T_1, and a disjoint time set $DT(T_1)$ in the part T_1. The function *readSTR* at Line 1 implements the above process. Then, with the DT-index, the code at Line 2 computes the disjoint time set $DT(T_2)$ by Eq. 7. At last, the code at Line 3 gets the disjoint time set DT in T.

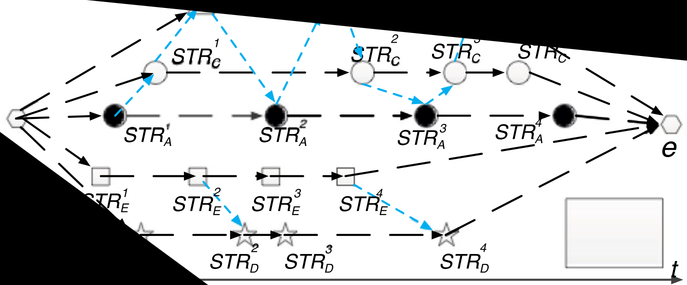
The algorithm can run very fast based on the following two reasons. One is that, in general, compared with the part T_2, the part T_1 is very short such that there are fewer sub-trajectories (*STRs*) in T_1. Hence, finding the disjoint time set $DT(T_1)$ is fast. The other is that, since the disjoint time set DT of each trajectory has been saved based on multiple time scales in the *DT*-index, only a small amount of nodes need to be searched from the index in order to compute the disjoint time set $DT(T_2)$ by Eq. 7. So, finding $DT(T_2)$ is also fast.

3.3 Splicing trajectory

3.3.1 Finding spliceable trajectories

We design an algorithm *createSTL-DAG* to discover spliceable trajectories by constructing a directed acyclic graph of sub-trajectory location connection (***STLC-DAG***), which is defined as *STLC-DAG* $= (V, E)$, where

- the vertex set V consists of all sub-trajectories (*STRSet*), a start vertex s, and an end vertex e, namely $V = \{STRSet\} \cup \{s, e\}$;
- the edge set E consists of two categories of directed edges. One, denoted as E_s, is the directed edge that connects two sub-trajectories in the same trajectory. The other, denoted as E_d, is the directed edge that connects a spliceable pair $\langle STR_i^j, STR_m^n \rangle$, as shown in Fig. 5.



STR_C^1
STR_C^2
STR_A^1
STR_A^2
STR_A^3
STR_A^4
e
STR_E^2
STR_E^3
STR_E^4
STR_D^2
STR_D^3
STR_D^4
t

Since there are the multiple sequences in the graph, all first vertexes from the sequence constitute a **candidate vertex set** (***CVS***), which is defined by Eq. 8.

$$CVS(STR_i^j) = \{STR_m^n | STR_m^n = first(\{ti(STR_m^k) \subset gap(STR_i^{j+1}, {}_m^k, STR_i^j)\}), m \in DT_i\} \tag{8}$$

For example, in Fig. 5, $CVS(STR_A^1) = \{STR_B^1, STR_C^1, STR_D^1, STR_E^2\}$.

Lemma 3 shows that when a trajectory cannot splice with another trajectory, the edge between the two trajectories can be deleted. Moreover, the deletion does not cause the result of spliceable trajectories to change.

The pseudocode of constructing the graph *STLC-DAG* is shown in Algorithm 2. The input arguments: the sub-trajectory set *STRSet* and the disjoint time set DT, are results of running the algorithm *queryDTsTR*, and γ is a distance threshold. The algorithm 2 will return a set $SP = \{SP_1, \ldots, SP_n\}$, where each SP_i is a group of spliceable trajectories.

Algorithm 2: *createSTLC-DAG*

```
   Input: STRSet, γ, SP = DT
   Output: SP
1  sortByStartTime(STRSet) ;
2  DAG.V = STRSet ∪ {s, e} ;
3  DAG.E.E_s = createEsEdge(STRSet, s, e) ;
4  C = φ ;
5  for k = 0; k < len(STRSet); k++ do
6      STR_i^j = STRSet[k] ;
7      for each STR_k^v ∈ sortByDes(C.get(STR_i^j)) do
8          sg = 0 ;
9          repeat
10             if !existPath(STR_k^v, STR_i^j, SP_k, DAG) then
11                 DAG.E.E_d.delEdges(TR_k, TR_i) ;
12                 SP_i = SP_i − k ;
13                 SP_k = SP_k − i ;
14                 C.del(⟨TR_i, TR_m⟩) ;
15                 sg = |C|;
16             else
17                 sg = sg − 1 ;
18             ⟨STR_k^v, STR_i^j⟩ ← C.next(STR_k^v, STR_i^j);
19         until ⟨STR_k^v, STR_i^j⟩ ≠ φ && sg > 0;
20     canTRSet = CVS(STR_i^j) ;
21     for each STR_m^n ∈ canTRSet do
22         if d(STR_i^j, STR_m^n) ≤ γ then
23             DAG.E.E_d.addEdge(STR_i^j, STR_m^n) ;
24         else
25             C.add(⟨STR_m^n, STR_i^j⟩) ;
26 return SP ;
```

Initially, the algorithm sorts all sub-trajectories in *STRSet* by their start time, creates all vertexes, and connects the vertexes that belong to the same trajectory (Line 1 3). C is a container that saves pairs of sub-trajectories which are likely to be indirectly spliced by others sub-trajectories (Line 4). For each sub-trajectory STR_i^j in $STRSet$, its candidate vertex set $CVS(STR_i^j)$ is firstly obtained by Eq. 8. Then, the algorithm creates a directed edge between the two sub-trajectories STR_i^j and STR_m^n

After Algorithm 2 finishes its running, if there exists an edge between two trajectories in the graph *STLC-DAG*, the two trajectories can be spliced according to Theorem 1. At the same time, the algorithm can find groups of spliceable trajectories SP, where each SP_i is a set of trajectories that can be directly or indirectly spliced with the trajectory TR_i based on Theorem 2.

Theorem 1 *If there exists a directed edge between two trajectories in the graph STLC-DAG, the two trajectories can be spliced.*

Theorem 2 *For each $SP_i \in SP$, where SP is one of the output parameters of algorithm 2, SP_i is a set of trajectories that can splice with the trajectory TR_i.*

The above two proofs are provided in Appendix B.

3.3.2 Finding maximum

Algorithm 5: *findApproxMaxCTR*

```
   Input: SP, SUBG = V, CAND = V, d, k, c = 0, fCTR = φ
   Output: fCTRSet: a fCTR set
1  if SUBG != φ then
2      if c = k then
3          if |CAND| ≤ (d − k) then
4              fCTR ← CAND;
5          else
6              fCTR ← takeFirst(CAND, d − k);
7          fCTRSet ← fCTR;
8          return ;
9      i = subscript(max|SUBG ∩ SP_i|), i ∈ SUBG;
10     branch = CAND − SP_i;
11     while branch != null do
12         b = takeFirst(branch) ;
13         fCTR ← b;
14         SUBG_b = SUBG ∩ SP_b;
15         CAND_b = CAND ∩ SP_b;
16         fCTRSet = findApproxMaxCTR(SP, SUBG_b, CAND_b, d, k, c + 1, fCTR);
17         CAND = CAND − {b};
18 else
19     fCTRSet ← fCTR;
20 return fCTRSet;
```

Based on the above analysis, we design an algorithm *findApproxMaxCTR* to find approximate maximal spliced paths quickly. The detailed pseudocode of *findApproxMaxCTR* is listed in Algorithm 5. The algorithm is similar to Algorithm 4 except the code on Lines 2–8. The additional parameters are as follows: d, k, and c, where d is used to limit the number of spliceable trajectories in one complete trajectory; k, which is used to limit the time of intersection between two *SP*s, is a recursive depth of the algorithm; and c records the current time of computing intersection in a spliced path *fCTR*. The code on Lines 2–8 shows how to deal with trajectories in *CAND* when $c = k$. If the size of *CAND* is less than $d - k$, all trajectories in *CAND* are added into *fCTR* (Lines 3–4). If the size is more than $d - k$, the first $(d - k)$ trajectories are added into *fCTR* (Line 6).

4 Time complexity analysis

In this section, we quantify the running time of the above algorithms and ignore algorithms in the preprocessing steps, such as the construction of B^+-tree and DT-index, because they can run offline. Let $\boldsymbol{T}$(*function*) be the running time of the *function*, M be the number of sub-trajectories, and N be the number of trajectories.

Lemma 7 *For the algorithm queryDTsTR, if the query time interval T consists of time slices from the DT-index, namely $T_1 = 0$ and $T_2 \neq 0$, the running time of queryDTsTR is $O(N^2)$; if the query time interval T does not contain the time slice for the DT-index, namely $T_2 = 0$ and $T_1 \neq 0$, the running time of queryDTsTR is $O(M^2)$.*

Proof Since all sub-trajectories are indexed by B^+-tree, the time of querying m sub-trajectories is $O(log_b^{|\Omega|} + M)$. $|\Omega|$ and b are constant. And, $log_b^{|\Omega|} \ll M$. So, the running time of reading all sub-trajectories in T is $O(M)$. At the same time, $R(T_1)$ and $R(T_2)$ can be obtained. If $T_1 = 0$, $DT(T_1)$ does not need to be computed. Therefore, $T(readSTR) = O(M)$. If $T_1 \neq 0$, the running time of computing $DT(T_1)$ is $O(M^2)$. And, $T(readSTR) = O(M^2)$. If $T_2 = 0$, Eq. 7 does not need to be computed. So, $T(queryDTsTR) = O(M^2)$.

If $T_2 \neq 0$, given that T_2 consists of k time slices which are in different levels in DT-index, k nodes in the DT-index need to be read. Each node contains no more than N items in which there are at most N TRs. According to Eq. 7, $T(\text{Eq. } 7) = O(kN^2)$. The running time of intersection between $DT(T_1)$ and $DT(T_2)$ is $O(N^2)$. So, *T(queryDTsTR)* is $O(N^2)$. □

Lemma 8 *The running time of the algorithm createSTLC-DAG is* $O(M^2N^2)$.

Proof Let $P = \sum_{i=1}^{N} |DT_i|$, where $DT_i \in DT$. So, $N \le P \le N^2$. The running time of creating vertexes (Line 3) and edges (Line 4) both are $O(M)$. In each loop (Line 5), $T(getCandSet) = O(m_k)$, where $m_k = |CVS(i, j)|$. And, the number of loops between Line 21 and 25 also is m_k. $T(addEdge)$ and $T(add)$ both are $O(1)$. The number of creating all edges in E_d (Lines 20–25) is $\sum_{k=1}^{M} m_k$ since $len(STRSet) = M$. According to $CVS(STR_i^j)$ (Eq. 8), $m_k \le DT_i$.

Since more sub-trajectories in TR_i result in less $|DT_i|$, the number of all edges is $\sum_{k=1}^{M} m_k$ and $\sum_{k=1}^{M} m_k \le \frac{kM}{N} \times P$, where $k \ll N$. Moreover, running time of $pseudocode$ on Lines 20–25 is $O(\frac{M}{N} \times P)$. If all edges are added into DAG (Line 23), C is empty. If all edges are added into C (Line 25), the longest time that $existPath$ runs is $\frac{M}{N} \times P$ because $delEdges$ (Line 11) can delete some edges. $T(existPath)$ depends on the number of vertexes and edges between the two sub-trajectories STR_k^v and STR_m^n. So, $T(existPath) = O(M + \frac{M}{N} \times P)$. The running time of operations on Lines 11–17 all is $O(1)$. The running time of $pseudocode$ on Lines 5–19 is $O(\frac{M}{N} \times P \times (M + \frac{M}{N} \times P)) = O(\frac{M^2}{N} \times P + \frac{M^2}{N^2} \times P^2)$.

Thus, $T(createSTLC\text{-}DAG) = O(M + \frac{M}{N} \times P + \frac{M^2}{N} \times P + \frac{M^2}{N^2} \times P^2) = O(\frac{M^2}{N} \times P + \frac{M^2}{N^2} \times P^2) = O(\frac{M^2}{N} \times (P + \frac{P^2}{N}))$. Owing to $P \le N^2$, $T(createSTLC\text{-}DAG) = O(M^2N^2)$ □

Lemma 9 *The running time of the algorithm findMaxCTR is* $O(3^{N/3})$.

Proof See Theorem 3 of [34]. □

Lemma 10 *Let D be a maximal degree of vertexes in the SP-set graph. The running time of the algorithm findApproxMaxCTR is* $O(N(N-D)C_{k-1}^{D-1})$. *Moreover, if k in Eq. 11 is a small numerical value, the running time of the algorithm findApproxMaxCTR is* $O(CN^2)$, *where C is a constant.*

Proof When the algorithm executes (depth 0) the code on Line 11 for the first time, $|branch| = N - D$. The algorithm will go to the branch SP_b, where the maximal degree of vertex b is D. Therefore, $|SUBG_b| \le D$. When it executes (depth 1) the code on Line 11 for the second time, $|branch| \le D - 1$. When it executes the code on Line 11 for the third time, $|branch| \le D - 2$.

Each branch repeats the above process until the depth of iteration reaches k. As the depth increases, $|branch|$ decreases. Moreover, in depth $k-1$, $|branch| \le D - k + 1$. According to Theorem 1 of [34], the algorithm generates all maximal cliques without duplication. So, each branch in the depth 1 is looked at as a combination C_{k-1}^{D-1}. The running time of $SUBG \cap SP_i$ on Line 9 is $O(N)$. Thus, $T(findApproxMaxCTR) = O(N(N-D)C_{k-1}^{D-1})$. When k is small, C_{k-1}^{D-1} is also small. Then, $T(findApproxMaxCTR) = O(CN^2)$. □

Table 2 Parameters

Notation	Definition
γ	The threshold of the distance between *STR*
d	The maximal length of a spliced path
p	Eq. 10
k	Top k complete trajectories (*CRTs*) sorted by Eq. 4

5 Experiments

In this section, we present the evaluation of the trajectory splicing query (Definition 5) and its algorithm based on two large real-world trajectory datasets. The first one is Geolife [47, 48], which is used to verify the effectiveness of our algorithm because it records labeled trajectories. The other is camera trajectory, which contains trajectories generated by the road safety cameras. Moreover, camera trajectory is mainly used to test the running time of algorithms, especially the algorithm *queryDTsTR* based on the *DT*-index, because it has large amounts of trajectories.

We use the two algorithms *findMaxCTR* and *findApproxMaxCTR* to implement the trajectory splicing query, respectively. Moreover, we implement the above two algorithms in Java language on a Linux server with Intel Xeon quad-core and 8 GB of main memory. The parameters used in the following experiments are defined in Table 2.

5.1 Evaluation on geolife

5.1.1 Data set and parameter setting

In the experiment, we extract trajectories from GeoLife in 2008 as the test data set. This test data set contains 4405 trajectories from 32 users. Each segment of those trajectories has been labeled by one of 11 transportation modes, which are bike, boat, bus, car, run, subway, taxi, train, walk, airplane, and others. These segments are considered from 11 different data sets. So, segments from the same user with the same label makes up the trajectory defined in the paper, denoted as *TR*. Each segment is the sub-trajectory defined in the paper, denoted as *STR*. The test data set contains 138 *TRs* and 4405 *STRs*, listed in Table 3.

The function $dist(i, j)$ is the Euclidean distance between two *TRs* with two labels i and j, respectively. Table 4 lists maximum, mean, and variance of $dist(i, j)$. For example, the first row in Table 4 represents the mean, variance, and max. distance between bike-*TRs* and other-*TRs*, which are 109,477 m, 146,006 m, and 212,719 m, respectively. We set four values

Table 3 Composition of *TR* Dataset

Id	Dataset	*TR*	*STR*	Id	Dataset	*TR*	*STR*
1	Airplane	1	2	7	Subway	7	108
2	Bike	14	301	8	Taxi	13	71
3	Boat	1	1	9	Train	4	12
4	Bus	22	426	10	Walk	28	756
5	Car	16	337	11	Other	30	2383
6	Run	2	8				

Table 4 Mean, Variance and Max. in $dist(i, j)$

Dist	*Mean* (m)	*Var* (m)	*Max* (m)	*Dist*	*Mean* (m)	*Var* (m)	*Max* (m)
1, 11	109, 477	146, 006	212, 719	4, 9	133, 446	173, 046	255, 808
1, 4	14, 576	0	14, 576	5, 10	55, 642	328, 973	2, 415, 622
1, 8	293, 078	0	293, 078	5, 11	34, 362	118, 063	1, 063, 245
2, 10	1500	2777	12, 075	5, 7	8564	39, 313	267, 034
2, 11	11, 257	84, 761	1, 023, 086	5, 8	11, 348	20, 908	76, 762
2, 4	2549	3654	12, 689	5, 9	13, 957	0	13, 957
2, 5	10, 001	17, 305	52, 276	7, 10	5850	7080	31, 996
2, 7	13, 171	20, 661	44, 042	11, 7	41, 265	132, 648	637, 270
2, 8	58, 703	118, 024	269, 712	7, 8	2265	4143	11, 631
3, 4	59, 156	73	59, 207	8, 10	15, 221	26, 122	77, 098
4, 10	12, 583	84, 028	986, 741	11, 8	223, 333	1, 214, 825	8, 328, 956
4, 11	23, 340	110, 415	1, 066, 120	8, 9	761, 691	951, 360	1, 828, 952
4, 5	124, 336	548, 462	2, 517, 981	9, 10	66, 511	98, 627	235, 890
4, 6	601	1315	5516	11, 9	468, 275	466, 053	1, 245, 493
4, 7	5894	11, 273	56, 182	11, 10	20, 986	109, 772	1, 125, 060
4, 8	6966	18, 875	77, 229				

for the parameter γ, which are $\gamma = m$, $\gamma = m + v$, $\gamma = m + 1.5v$ and $\gamma = max$, where m, v, and max are *mean*, *var*, and *max* in Table 4, respectively.

5.1.2 *findMaxCTR* vs *findApproxMaxCTR*

In order to evaluate the effectiveness of the two algorithms that splice trajectories from the above 11 datasets, we define ***recall***, ***precision***, and ***completeness*** as Eqs. 12, 13, and 14. *recall* represents the ability of which the two algorithms can recover complete trajectories (*CTR*) from the above 11 datasets; *precision* can show the degree of which top *k CTR*s contain user trajectories in Geolife; *completeness* is the degree that one complete trajectory recovers a user trajectory.

$$recall = num_a/num_b \tag{12}$$

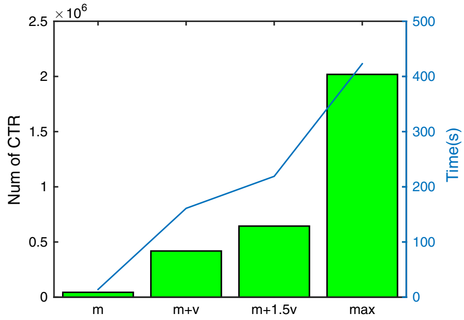
where num_b is the number of user trajectories in the test dataset and num_a is the number of user trajectories found by one of the two algorithms. In this experiment, $num_b = 32$ due to total 32 user trajectories in the dataset.

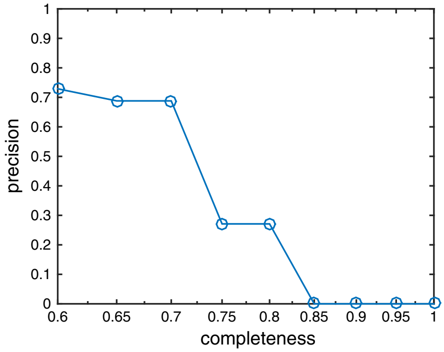
$$precision = num_c/k \tag{13}$$

where num_c is the number of complete trajectories that contain a user trajectory; k refers to top k complete trajectories ranked by Eq. 4.

$$completeness = \frac{|label(CTR) \cap (userTra)|}{|label(userTra)|} \tag{14}$$

where the function $label(.)$ returns the set of transportation modes in a trajectory; $|label(userTra)|$ is the number of labels that appear in a user trajectory *userTra* in the dataset; and $|label(CTR) \cap label(userTra)|$ is the number of labels that appear both in *CTR* and *userTra*.


×10^6
2.5
2
1.5
1
0.5
0
Num of CTR
500
400
300
200
100
0
Time(s)
m
m+v
m+1.5v
max


precision
completeness
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1

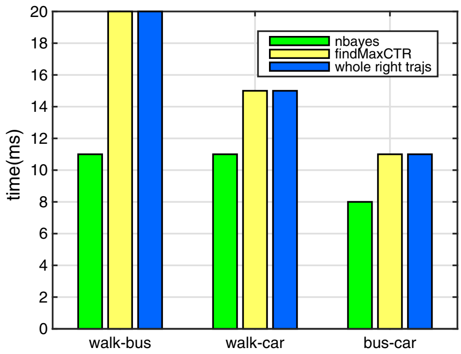


Fig. 11 *nbayes* vs. *findMaxCTR* on right trajectories

5.2 Evaluation on CameraTrajectory

5.2.1 Data set and parameter setting

In the data set, a trajectory consists of sample points that are generated by road safety cameras, which record information of vehicles that pass by them. The data set has 10,104 trajectories and 12,741,728 sample points over three months at Guyuan, China. Since we do not know which trajectories in the data set can be spliced in advance, for computing effectiveness of the algorithms, we manually select 104 trajectories from the data set as test trajectories and randomly split these trajectories into 568 trajectories. After the two algorithms run, we observe how many complete trajectories (*CTR*s) contain these test trajectories. Thus, we can compare *recall*, *precision*, and F_1 between the two algorithms. By setting thresholds $speed = 1$ (m/s) and $distance = 10{,}000$ (m), all trajectories in the data set are split into sub-trajectories. So, there is a total of 10,568 trajectories (*TR*s) and 1,812,568 sub-trajectories (*STR*s) in the data set.

5.2.2 *findMaxCTR* vs *findApproxMaxCTR*

With the parameter $\gamma = 5000$ m, the results of *findMaxCTR* versus *findApproxMaxCTR* are shown in Fig. 12, where ($d = 7, p = 0.9$), ($d = 14, p = 0.9$), ($d = 28, p = 0.9$), and ($d = 38, p = 0.9$) are the four groups of parameters in *findApproxMaxCTR*. *findMaxCTR* finds total 13,581 groups of spliceable trajectories. However, its *recall* is about 20% as shown in Fig. 12a, because many spliceable trajectories found by it do not satisfy the function *isSplicePath* so that they are discarded.

Compared with *findMaxCTR*, *findApproxMaxCTR* finds approximate maximal spliceable trajectories which are not checked by *isSplicePath*. Therefore, it has a higher *recall* than *findMaxCTR* when d is bigger. For example, when $d = 38$ and $p = 0.9$, its *recall*s are 82% on *completeness* = 1 and 93% on *completeness* = 0.85, respectively. However, when $d = 7$, it has a lower *recall* because the code on Lines 2–8 prunes many branches that contain spliceable trajectories in Algorithm 5. So, if d is in a reasonable range, *findApproxMaxCTR* is more robust than *findMaxCTR* because its approximate results are not filtered by Definition 5.

When selecting the first 4000 results found by the two algorithms, the precisions of the two algorithms are illustrated in Fig. 12b. Compared with *findApproxMaxCTR*, *findMaxCTR* can find more super trajectories although it has a poor ability to find super trajectories with high

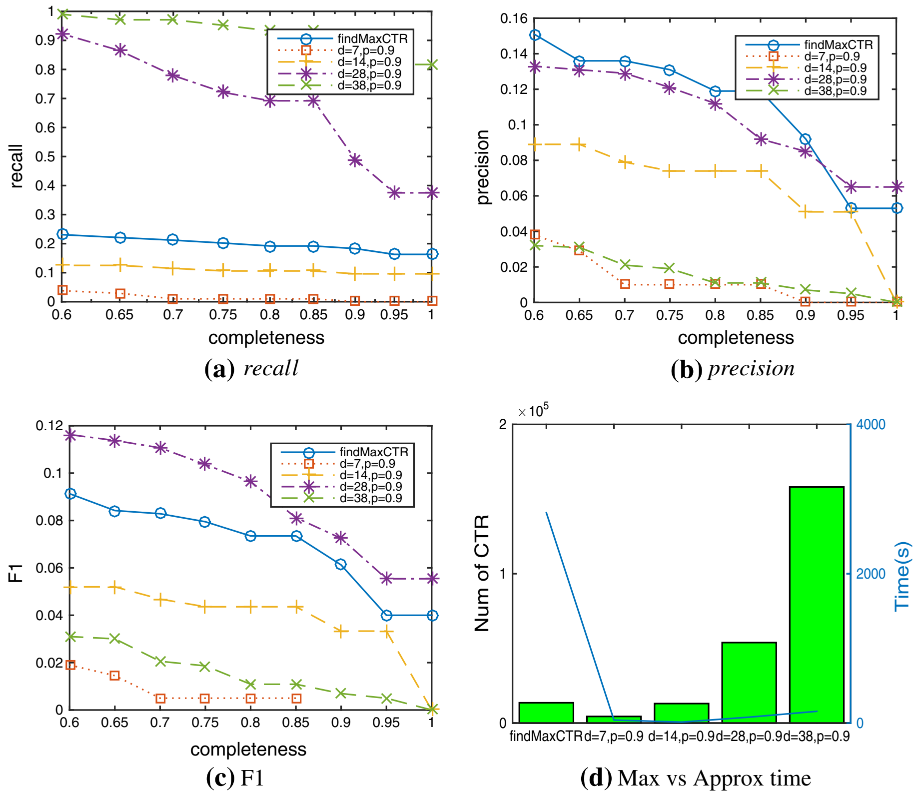


Fig. 12 *findMaxCTR* vs. *findApproxMaxCTR*

completeness. According to the F1 score on Fig. 12c, *findApproxMaxCTR* with the fitted parameters ($d = 28$ and $p = 0.9$) is better than *findMaxCTR*. However, searching for the right parameter values is very troublesome since it needs to try many different parameter values. So, from the view of simplicity, *findMaxCTR* is a good choice.

The time of *findMaxCTR* running on GeoLife (138 *TR*s) is about 160*s*, while its time on CameraTrajectory (10568 *TR*s) is about 2816*s*, as shown in Figs. 7b and 12d. However, it not

Table 5 Components in *DT*-tree

Level	*DT*-tree		*DF*-tree	
	# of *DTNode*	*Avg size*(*kb*)	# of *DFNode*	*Avg size*(*kb*)
1	13	39,002	12	33,124
2	6	39,831	5	43,695
3	3	37,905	2	87,141

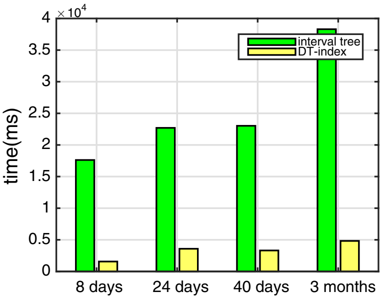


Fig. 13 B^+-tree vs. *DT*-index on computing *DT*

DT-tree and the *DF*-tree both have three levels of nodes except their root nodes. The sizes of the B^+-tree and the *DT*-index are 137 Mb and 1.65 Gb, respectively, after constructing the two indexes. Table 5 lists the details of the *DT*-index. The sizes of *DTNodes* in different levels are almost the same because, according to Eq. 15, longer the time, smaller the change in the disjoint time set of a trajectory. However, the change of sizes between *DFNodes* at different levels is big, because there is a significant difference between the two neighboring $\neg DT_i$ so that the size of DF_i is large based on $DF_i^n = \neg DT_i^n - \neg DT_i^{n-1}$. Although the size of the *DT*-index is very large, some lossless data compression algorithms, e.g., Lempel-Ziv (LZ) compression algorithm, can decrease its size. By *LZ*78 algorithm, the size of the *DT*-index changes from 1.65 Gb to 700 Mb.

As mentioned earlier in *queryDTsTR*, if $T_2 = 0$, it will search the disjoint time sets of all trajectories in the B^+-tree (called ***ITQuery***). If $T_1 = 0$, it will search all the disjoint time sets in the *DT*-index (called ***DTQuery***). After *ITQuery* and *DTQuery* run 10 times in different time intervals (8, 24, 40 days, and 3 months), their average time is shown in Fig. 13.

Apparently, *DTQuery* runs faster than *ITQuery* because the time complexity of *DTQuery* is $O(N^2)$ while the time complexity of *ITQuery* is $O(M^2)$, and $M \gg N$. As the users' time grows, M becomes bigger but N does not change. So, the main factor that affects the running time of *DTQuery* is only the I/O time of reading the disjoint time sets from the *DT*-index.

which is built based on the time, supports to retrieve sub-trajectories in these time intervals. Moreover, the indexes based on B^{+}-tree [37] and R-tree [18,33,35,40] can efficiently process the queries of time intervals. Although these indexes can process the queries, they cannot efficiently deal with the queries of time-disjoint sets because, in each query, they only support to search in a specific time interval not in multiple time intervals so that they need many queries of time intervals to discover these trajectories whose times are disjoint.

In addition to the disjoint time constraint on trajectories, spliceable trajectories require that the gap distances between them are close enough that they constitute a complete trajectory. Symbolic trajectories [13], which give us a conceptual view to understand various behaviors of the moving objects [30], can capture these spliceable trajectories by a sequence of time-dependent labels. The symbolic trajectory of a moving object is represented as a sequence of units $\langle u_1, u_2, \ldots, u_n \rangle$, where u_n is a pair $\langle t, s_b, s_e, l \rangle$ in which i is a time interval, s_b and s_e are the locations of two end points of the unit, and l is a label. For example, for the case in Sect. 1, the symbolic trajectory of *Bob* is the sequence $\langle([8:00-8:20], H, A, walk), ([8:23-9:14], A, B, subway), ([9:16-9:21], B, C, walk), \ldots\rangle$.

Güting et al. [13,29,35,40] create the data model of symbolic trajectories and their indexes to offer operations to search trajectories by the above sequence of time-dependent labels. More especially, these operations support to retrieve symbolic trajectories which satisfy the conditions of the time interval, spatial distance, and a sequence of labels. For example, the retrieval SQL of Bob transition from walk to subway is '$select\ pid\ from\ Case1\ where\ trans\ matches\ '* X(\_walk)\ Y(\_subway)*//\ Y.start-X.end \leq duration(0\ 9000000)'\ and\ pid = Bob$'. In order to match the symbolic trajectories from the database, the users must know the sequence of labels in advance. However, in the paper, the sequence of *label* is unknown before the users begin to retrieve spliceable trajectories. So, symbolic trajectory methods do not apply to queries for the spliced mode.

Spatiotemporal joins [32,49] find close pairs of trajectories from two datasets, respectively, based on the distances between the pairs of trajectories. Based on these close pairs, the trajectory join [1] retrieves groups of moving objects that have similar movements at a different time. Kexin Xie et al. [39] propose a spatiotemporal join method to associate segments of a trajectory with points of interest (POI) according to the distance between a POI and a trajectory and duration which a trajectory is geographically near a POI. However, the distances in these spatiotemporal join methods are the similarity between the two trajectories, while the gap distance between two trajectories is the Euclidean distance. So spatiotemporal joins are not fit to find spliceable trajectories defined in this paper because these spliceable trajectories are not similar.

6.2 Trajectory pattern analysis and mining

The spliced model needs to find groups of spliceable trajectories from different systems. Group pattern mining and trajectory clustering both find groups of moving objects based on similarity of their trajectories in a specific time interval, such as flock [8,9,36], convoy [19], swarm [27], group [26], gathering [45], and trajectory clustering methods [24,25]. These methods define different distance functions to evaluate the similarity between trajectories, and design corresponding cluster algorithms to discover groups of similar trajectories. However, these methods are not fit to find groups of spliceable trajectories because they find similar trajectories while spliceable trajectories are not similar. Another line of research on frequent trajectory mining targets at assigning travel cost-based weights to edges [15,16,42] and paths [3,44] that are frequently traversed by trajectories, where the travel costs can be travel time or fuel consumption [11,12]. However, only frequently traversed edges and paths are identified, which cannot be used directly to identify spliceable trajectories.

From the view of recovering complete user trajectories, a spliceable trajectory is one of the transportation modes in the user complete trajectory. So, discovering spliceable trajectories needs to decide whether other trajectories can splice with the current trajectory based on their information about time, location, and transportation mode. Trajectory inference methods [5,28,31,46] seem to be able to make the above decisions since these methods can predict a user location, infer his transportation mode, and predict when and where he will change mode [28] based on the known trajectory information. However, these methods are not good at dealing with the problem of splicing multiple trajectories owing to the two following reasons. One is that the problem of trajectory splicing acts on the different data sources while trajectory inference methods act on a single data source. In multiple data sources, each data source has a different ID code and contains trajectories of one transportation mode, and it is difficult to know in advance whether trajectories from different data sources belong to a user movement. So, the model of the problem is not built on a user history trajectory. More specifically, it is impossible to count the probability that one user switches one transportation mode to another. But, a single data source makes trajectory inference methods know user complete trajectory so that they can create their model based on user history trajectory.

The other is that they have different goals. The goal of our work is to match trajectories so that they can form one group, while the goal of trajectory inference methods is to predict a user location, infer his transportation mode, and so on. From the view of statistical learning, our work is the clustering problem, while trajectory inference methods are the regression problem. Preference learning is able to identify drivers with similar driving preferences and thus group their trajectories together [2,12,43]. However, it is unable to identify individual drivers.

The fuzzy trajectory linking(FTL) [38] is close to our work. It finds pairs of trajectories that belong to the same moving object by the two methods: (α_1, α_2)-filtering and naive Bayes matching. Compared with our methods, FTL can link (splice) two trajectories based on the distribution of distances between any two time-order points from the two trajectories, respectively. So, it avoids the disjoint time constraint in our work so that it can splice two trajectories even if their sub-trajectories overlap with each other in time. However, it does not support multiple trajectories splicing efficiently because the two above methods will be invalid as more trajectories are involved in a spliced process. Nevertheless, our method can splice multiple trajectories. Driver identification is also similar to our work in the sense that it also tries to identify trajectories from different drivers. However, it focuses on learning distinctive representations of driving behaviors and then clusters the representations [20], but ignores disjoint time and spatial closeness.

7 Conclusion

In this paper, we study the problem of trajectory splicing, which reconstructs individual com-

For f t re work, it i of intere t to extend the liced degree b con idering other factor , ch a the n mber of the b-trajectorie , and the ha e of the b-trajectorie , to e al ate the alit of the recon tr cted indi id al com lete trajector . It i al o of intere t to ar al-leli e [41] the ro o ed algorithm to im ro e the efficienc and to relax the time-di joint con traint to extend the liced model to incl de more indi id al artial trajectorie .

Acknowledgements We o ld like to thank Profe or Chri tian S. Jen en for ef l di c ion and comment . Thi work a orted b National Science and Technolog Major Project (no. 2017ZX05018-005), National Nat ral Science Fo ndation of China (no. 61402532), Science Fo ndation of China Uni er it of Petrole m-Beijing (no. 01JB0415), and China Scholar hi Co ncil.

Appendix A Computing disjoint time set

Lemma *In the query interval time T, the disjoint time set DT_i of each trajectory TR_i can be computed by Eq. 6.*

Proof Let $Q_i^{k,d}$ be a trajector et here each trajector TR a ear in T and it time inter al et $ti(TR)$ doe not o erla ith TR_i

Proof Let P_c which is found by *existPath* be a path from STR_k^v to STR_i^j. We first prove there must exist a path P_l from STR_k^v to STR_i^j in the current graph *STLC-DAG*. P_l is an time-ordered sequence, where each $STR \in \{STR_m^n | ti(STR_k^v).st < ti(STR_m^n).st < ti(STR_i^j).st, m \in M(P_c)\} \cup \{STR_k^v, STR_i^j\}$. And, $M(P_c)$ is a set of *TR* that P_c has passed through except i and k. We prove the problem according to the following situations.

If $|M(P_c)| = 0$ or $|M(P_c)| = 1$, P_c must be P_l.

If $|M(P_c)| \geq 2$, suppose P_l does not exist in the current *STLC-DAG*. Let P_a be the path contains the maximum number of *STRs* from P_l, where $M(P_c) \subseteq M(P_a)$. Then, at least one vertex STR_m^n from P_l is not on P_a. According to time, let STR_m^n be between $P_a[i]$ and $P_a[i+1]$, namely $ti(P_a[i]).st < ti(STR_m^n).st < ti(P_a[i+1]).t$, where $P_a[i](P_a[j])$ is a ith or jth *STR* in P_a, $m_i(m_{i+1})$ is the subscript of $P_a[i](P_a[i+1])$, and $m_i, m_{i+1} \in m(P_c)$. Therefore, before running the current pair, the algorithm has executed evaluation of the two pairs $\langle P_a[i], STR_m^n \rangle$ and $\langle STR_m^n, P_a[i+1] \rangle$. The evaluation generated two following results. One is that, if there does not exist a path between $\langle P_a[i], STR_m^n \rangle$ or $\langle STR_m^n, P_a[i+1] \rangle$, it shows TR_m and $TR_{m_i}(TR_{m_{i+1}})$ cannot be spliced. So, $m_i \notin SP_m$ or $m_{i+1} \notin SP_m$. According to $existPath$ (Algorithm 3), it cannot find that a path contains $STR_{m_i}(STR_{m_{i+1}})$ and STR_m. It contradicts with P_c. The other is that, if there does exist both above paths, STR_m^n can be added into P_a. It contradicts with P_a that has the maximum number of *STRs* from P_l. Therefore, P_l must exist in the current *STLC-DAG*.

Then, since P_l from STR_k^v to STR_i^j exists in *STLC-DAG*, it implies that there must exist a path P_b from the start vertex to STR_k^v in the current *STLC-DAG*. And, P_b contains all *STRs* of *TRs* between the start vertex and STR_k^v (P_c has passed through these *TRs*). This is because the algorithm has processed previous pair $\langle STR_t^r, STR_k^v \rangle$. And, there must exist a path P_t similar to P_a between STR_t^r and STR_k^v owing to the path found by *existPath*. And so on, these previous pairs form the P_b. Therefore, the P_b and P_l can form a spliced path. □

Lemma 5 If and only if a path found by algorithm 3 contains sub-trajectories from two different trajectories, the two trajectories can be spliced.

Proof If there exists a path, which is found by Algorithm 3, between any two sub-trajectories from two trajectories, respectively, according to Lemma 4, the trajectories that the path passed through can be spliced with the two sub-trajectories. So, the two two sub-trajectories can be spliced. According to the definition 6, if two sub-trajectories are spliceable sub-trajectories, there exists a spliced path that can pass through all sub-trajectories of the two trajectories. □

Theorem 1 *If there exists a directed edge between two trajectories, the two trajectories can be spliced.*

Proof Suppose there is an edge between STR_i^j and STR_m^n, which the two *STRs* belong to TR_i and TR_j, respectively, and TR_i cannot be spliced with TR_m. According to Lemma 5, at least one pair of *STRs* from the two *TRs*, respectively, cannot be connected by a path that is found by $existPath$. But, a Algorithm 2 (Line 10) must have deleted all edges between TR_i and TR_j if it finds that a pair between them cannot be connected by a path. Therefore, there is not an edge between them. It contradicts the assumption that there is an edge between STR_i^j and STR_m^n.

Theorem 2 *For each $SP_i \in SP$, where SP is one of output parameters of Algorithm 2, SP_i is a set of trajectories that can be spliced with the trajectory TR_i.*

Proof At initialized phase of Algorithm 2, $SP = DT$. Suppose one SP_i has a subscript m, and its corresponding TR_m cannot be spliced with TR_i. According to Lemma 5, there is not a path between one pair $\langle STR_i^j, STR_m^n \rangle$. And, $SP_i = SP_i - m$ (Line 12 in Algorithm 2), has been executed. It contradicts with SP_i because SP_i contains m. □

Lemma 6. In *SP*-set graph, a clique is a group of spliceable trajectories, a maximal clique is a complete trajectory.

Proof A group of spliceable trajectories can be directly or indirectly spliced with each other. Therefore, there exists an edge between any two of them. So, the group of spliceable trajectories is a clique in the graph. If the clique is the maximal clique, the group of spliceable trajectories on the maximal clique cannot be contained by other groups. So, the maximal clique in the graph is a complete trajectory *CTR*. □

References

1. Bakalov P, Hadjieleftheriou M, Tsotras VJ (2005) Time relaxed spatiotemporal trajectory joins. In: Proceedings of the 13th annual ACM international workshop on geographic information systems, ACM, New York, NY, USA, pp 182–191
2. Dai J, Yang B, Guo C, Ding Z (2015) Personalized route recommendation using big trajectory data. In: 2015 IEEE 31st international conference on data engineering, pp 543–554
3. Dai J, Yang B, Guo C, Jensen CS, Hu J (2016) Path cost distribution estimation using trajectory data. Proc VLDB Endow 10(3):85–96
4. Ding Z, Yang B, Chi Y, Guo L (2016) Enabling smart transportation systems: a parallel spatio-temporal database approach. IEEE Trans Comput 65(5):1377–1391
5. Emrich T, Kriegel HP, Mamoulis N, Renz M, Züfle A (2012) Querying uncertain spatio-temporal data. In: 2012 IEEE 28th international conference on data engineering, pp 354–365
6. Eppstein D, Löffler M, Strash D (2010) Listing all maximal cliques in sparse graphs in near-optimal time. In: Algorithms and computation, no. 6506 in lecture notes in computer science, Springer Berlin Heidelberg, pp 403–414
7. Goh CH, Lu H, Ooi BC, Tan KL (1996) Indexing temporal data using existing B+-trees. Data Knowl Eng 18(2):147–165
8. Gudmundsson J, van Kreveld M (2006) Computing longest duration flocks in trajectory data. In: Proceedings of the 14th annual ACM international symposium on advances in geographic information systems, ACM, New York, NY, USA, pp 35–42
9. Gudmundsson J, van Kreveld M, Speckmann B (2004) Efficient detection of motion patterns in spatio-temporal data sets. In: Proceedings of the 12th annual ACM international workshop on geographic information systems, ACM, New York, NY, USA, pp 250–257
10. Guo C, Jensen CS, Yang B (2014) Towards total traffic awareness. SIGMOD Rec 43(3):18–23
11. Guo C, Yang B, Andersen O, Jensen CS, Torp K (2015) Ecomark 2.0: empowering eco-routing with vehicular environmental models and actual vehicle fuel consumption data. GeoInformatica 19(3):567–599
12. Guo C, Yang B, Hu J, Jensen CS (2018) Learning to route with sparse trajectory sets. In: IEEE 34th international conference on data engineering, pp 1073–1084
13. Güting RH, Valdés F, Damiani ML (2015) Symbolic trajectories. ACM Trans Spat Algorithms Syst 1(2):7:1–7:51

19. Je ng H, Yi ML, Zho X, Jen en CS, Shen HT (2008) Di co e of con o in t ajecto databa e . 1:1068 1080
20. Kie T, Yang B, G o C, Jen en CS (2018a) Di ting i hing t ajecto ie f om diffe ent d i e ing incom- letel labeled t ajecto ie . In: P oceeding of the 27th ACM inte national confe ence on info mation and kno ledge management, 863 872
21. Kie T, Yang B, Jen en CS (2018b) O tlie detection fo m ltidimen ional time e ie ing dee ne al net o k . In: IEEE 19th inte national confe ence on mobile data management, 125 134
22. Kie T, Yang B, G o C, Jen en CS (2019) O tlie detection fo time e ie ith ec ent a toencode en emble . In: 28th inte national joint confe ence on a tificial intelligence
23. Ko te B, V gen J (2012) Combinato ial o timi ation, algo ithm and combinato ic , ol 21. S inge , Be lin
24. Lee JG, Han J, Whang KY (2007) T ajecto cl te ing: a a tition-and-g o f ame o k. In: P oceeding of the 2007 ACM SIGMOD inte national confe ence on management of data, ACM, Ne Yo k, NY, USA, 593 604
25. Lee JG, Han J, Li X (2015) A nif ing f ame o k of mining t ajecto atte n of a io tem o al tightne . IEEE T an Kno l Data Eng 27(6):1478 1490
26. Li X, Ceik te V, Jen en C, Tan KL (2013) Effecti e online g o di co e in t ajecto databa e . IEEE T an Kno l Data Eng 25(12):2752 2766
27. Li Z, Ding B, Han J, Ka R (2010) S a m: mining ela ed tem o al mo ing object cl te . P oc VLDB Endo 3:723 734
28. Liao L, Patte on DJ, Fo D, Ka t H (2007) Lea ning and infe ing t an o tation o tine . A tif Intell 171(5 6):311 331
29. Sak MA, G ting RH (2011) S atiotem o al atte n e ie . GeoInfo matica 15(3):497 540
30. S acca iet a S, Pa ent C, Damiani ML, de Macedo JA, Po to F, Vangenot C (2008) A conce t al ie on t ajecto ie . Data Kno l Eng 65(1):126 146
31. S H, Zheng K, H ang J, Wang H, Zho X (2014) Calib ating t ajecto data fo atio-tem o al imila it anal i . VLDB J 24(1):93 116
32. S n J, Tao Y, Pa adia D, Kollio G (2006) S atio-tem o al join electi it . Inf S t 31(8):793 813
33. Tao Y, Pa adia D (2001) MV3 -T ee: A atio-tem o al acce method fo time tam and inte al e ie . In: P oceeding of the 27th inte national confe ence on e la ge data ba e , Mo gan Ka fmann P bli he Inc., San F anci co, CA, USA, 431 440
34. Tomita E, Tanaka A, Takaha hi H (2006) The o t-ca e time com le it fo gene ating all ma imal cli e and com tational e e iment . Theo Com t Sci 363(1):28 42
35. Vald F, G ting RH (2014) Inde - o ted atte n matching on mbolic t ajecto ie . In: P oceeding of the 22Nd ACM SIGSPATIAL inte national confe ence on ad ance in geog a hic info mation tem , ACM, Ne Yo k, NY, USA, 53 62
36. Vie a MR, Bakalo P, T ot a VJ (2009) On-line di co e of flock atte n in atio-tem o al data. In: P oceeding of the 17th ACM SIGSPATIAL inte national confe ence on ad ance in geog a hic info mation tem , ACM, Ne Yo k, NY, USA, 286 295
37. Wang L, Zheng Y, Xie X, Ma WY (2008) A fle ible atio-tem o al inde ing cheme fo la ge- cale GPS t ack et ie al. In: 9th inte national confe ence on mobile data management, IEEE, 1 8
38. W H, X e M, Cao J, Ka a P, Ng WS, Koo KK (2016) F t ajecto linking. In: IEEE 32nd inte national confe ence on data enginee ing, IEEE, 859 870
39. Xie K, Deng K, Zho X (2009) F om t ajecto ie to acti itie : a atio-tem o al join a oach. In: P oceeding of the 2009 inte national o k ho on location ba ed ocial net o k , ACM, Ne Yo k, NY, USA, 25 32
40. X J, G ting RH, Zheng Y (2015) The TM-RT ee: an inde on gene ic mo ing object fo ange e ie . GeoInfo matica 19(3):487 524
41. Yang B, Ma Q, Qian W, Zho A (2009) TRUSTER: t ajecto data oce ing on cl te . In: DASFAA, 768 771
42. Yang B, G o C, Jen en CS, Ka l M, Shang S (2014) Stocha tic k line o te lanning nde time- a ing nce taint . In: IEEE 30th inte national confe ence on data enginee ing, 136 147
43. Yang B, G o C, Ma Y, Jen en CS (2015) To a d e onali ed, conte t-a a e o ting. VLDB J 24(2):297 318
44. Yang B, Dai J, G o C, Jen en CS, H J (2018) PACE: a ath-cent ic a adigm fo tocha tic ath finding. VLDB J 27(2):153 178
45. Zheng K, Zheng Y, Y an N, Shang S, Zho X (2014) Online di co e of gathe ing atte n o e t ajec- to ie . IEEE T an Kno l Data Eng 26(8):1974 1988
46. Zheng Y (2015) T ajecto data mining: an o e ie . ACM T an Intell S t Technol 6(3):1 41

47. Zheng Y, Zhang L, Xie X, Ma WY (2009) Mining interesting locations and travel sequences from GPS trajectories. In: Proceedings of the 18th international conference on world wide web, ACM, New York, NY, USA, pp 791–800
48. Zheng Y, Xie X, Ma WY (2010) Geolife: a collaborative social networking service among user, location and trajectory. IEEE Data Eng Bull 33(2):32–39
49. Zhou P, Zhang D, Salzberg B, Cooperman G, Kollios G (2005) Close pair queries in moving object databases. In: Proceedings of the 13th annual ACM international workshop on geographic information systems, ACM, New York, NY, USA, pp 2–11

Qiang Lu received a B.S. degree from Shenyang University of Chemical Technology, Shenyang, China, in 2000 and a Ph.D. degree from China University of Petroleum-Beijing, China, 2006. From 2015 to 2016, he was a visiting scholar at the Department of Computer Science, Aalborg University, Denmark. He is currently an Associate Professor in the Department of Computer Science and Director of Computation Intelligence Center at China University of Petroleum, Beijing. He is also a faculty member in Beijing Key Lab of Petroleum Data Mining. His research interests include spatial-temporal data processing, evolutionary computing, and machine learning.

Rencai Wang received a B.S. degree from China University of Petroleum-East China, in 2014 and an M.S. degree from China University of Petroleum-Beijing, China, in 2017. He is currently working as a software engineer at IFLYTEK CO., LTD, responsible for data analysis and mining on education, and the development of the education cloud platform. His research interests include spatial temporal data management, trajectory computing, and data mining on behavior of the educational user.

Bin Yang is a Professor in the Department of Computer Science at Aalborg University, Denmark. He was at Aarhus University, Denmark and at Max Planck Institute for Informatics, Germany. He received the Ph.D. degree in computer science from Fudan University. His research interests include machine learning and data management. He was a PC co-chair of IEEE MDM 2018. He has served on program committees and as an invited reviewer for several international conferences and journals, including ICDE, IJCAI, TKDE, the VLDB Journal, and ACM Computing Surveys.

Zhiguang Wang received a B.S. degree in physics from Inner Mongolia Normal University in 1986, an M.S. degree in computer metering from Jilin University in 1994, and a Ph.D. degree in computer science from China University of Petroleum-Beijing. He is currently a Professor in the Department of Computer Science at China University of Petroleum, Beijing, serves as a Director of Research Group of Large Scale Data Processing and Visualization. He is also a faculty member in Beijing Key Lab of Petroleum Data Mining, and a council member in Beijing Education Federation. His current research interests include data management, distributed systems, and spatial-temporal data mining.